ICOT Technical Memorandum: TM-1160

TM-1160

Cu-Prolog 第三版処理系仕様書
(Cu-Prolog III System)

津田　宏、橋田　浩一、
白井　英俊（中京大学）

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

Institute for New Generation Computer Technology

cu-Prolog第三版 処理系仕様書
(cu-PrologIII system)

1992.2
ICOT TM-

津田 宏　(ICOT第三研究室 tsuda@icot.or.jp)
橋田 浩一 (ICOT第三研究室 hasida@icot.or.jp)
白井 英俊 (中京大学 sirai@sccs.chukyo-u.ac.jp)

<< 目次 >>

cu-Prolog 第三版ユーザーズマニュアル

津田 宏
(財) 新世代コンピュータ技術開発機構 第三研究室
〒 108 東京都港区三田 1-4-28 三田国際ビル 21 階
Tel : +81-3-3456-3069　　E-mail : tsuda@icot.or.jp

1992 年 2 月 5 日

1　概要

制約論理型言語 cu-Prolog[4, 3, 7] は、自然言語処理の単一化文法の特徴である宣言的な文法記述の実装を目的とした、記号的・組合せ的な制約を解くのに適した言語である。ここで言う制約とは、変数共有や変数束縛などによる依存関係をいう。cu-Prolog は、従来の CLP の多くによる代数方程式・不等式による制約に比して、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といえる。

cu-Prolog は ICOT の PSG-WG にて、JPSG(Japanese Phrase Structure Grammar: 日本語句構造文法) のパーザを実現する目的で考案された。JPSG では、情報を素性 (ラベル) とその値の対の不定個の並びによる素性構造により格納し、それらをノードとする句構造によって自然言語の構造を表現する。JPSG のような制約ベースの単一化文法では、自然言語の文法は全て制約で宣言的に記述される。JPSG における制約とは、二分木の各ノードにおける素性値の記号的・組み合わせ的関係である。

cu-Prolog は、Prolog のユーザ定義述語による項を制約として扱うことが出来るので、JPSG 等の自然言語処理における制約を自然に記述し、そのまま制約として実行することが可能である。cu-Prolog の制約解消系は、論理プログラムの unfold/fold 変換を基本操作とし、変数共有や束縛等によるヒューリスティックを加えた所に特徴がある。

cu-Prolog 第三版では、制約ベースの単一化文法の記述により適したものとするため、データ構造として部分項 (Partially Specified Term) を取り入れた。これにより、素性構造をそのまま取り扱うことが出来る。さらに、制約解消系も部分項の導入と共に自然に拡張を行い、単一化文法で重要となる選言的素性構造に対しても対応できるようになった。選言的素性構造の単一化は本来計算量の大きな問題であり、より実際的なアルゴリズムが研究されている [2]。cu-Prolog 第三版では、それがヒューリスティックスまでも含めて一般的な枠組で解決されている [6]。

本マニュアルは cu-Prolog 第三版のユーザーズマニュアルである。cu-Prolog 第二版は C 言語により記述され、UNIX 版 (4.2/3BSD 上にて動作) と、中京大学の白井英俊助教授 (sirai@sccs.chukyo-u.ac.jp) により開発された Macintosh 版とがある。

1.1　cu-Prolog 実行モジュールの作成法

1.1.1　モジュール構成

cu-Prolog 第三版は、C 言語による以下のモジュールからなる。

- ヘッダファイル

  `include.h` 構造体・マクロの定義

  `funclist.h` 他モジュールで使われる関数の外部参照

varset.h 大域変数の初期化

globalv.h 大域変数の外部参照

defsysp.h 組み込み述語変数の初期化

sysp.h 組み込み述語変数の外部参照

- システムの基本モジュール

  main.c 処理系トップレベル

  mainsub.c main.c のサブルーチン

  new.c メモリ管理

  read.c プログラム・項の読み込み

  print.c 項の表示ルーチン

- Prolog 処理系モジュール

  refute.c 処理系本体

  unify.c 単一化

- 組み込み述語に関するモジュール

  defsysp.c 組み込み述語の初期化

  syspred1.c 組み込み述語の定義.1

  syspred2.c 組み込み述語の定義.2

  jpsgsub.c JPSG パーザ用組み込み述語の定義

- 制約解消系に関するモジュール

  modular.c 制約解消系エントリ、共通ツール

  trans.c 制約解消系メイン

  tr_sub.c 制約解消系サブ

  tr_split.c 素式の分割ルーチン

これらのモジュールをコンパイル、リンクして実行ファイルを得る。例えば UNIX の場合は上記のファイルを一つのディレクトリに置き、

```
cc -o cuprolog *.c <cr>
```

とする。また添付の makefile を用いて、

```
make <cr>
```

としてもよい。

1.1.2 コンパイルの注意

使用する機種、コンパイラによっては、コンパイル前に include.h 中の最初の #define 文を書き換える必要がある。

- SUN4 の場合、#define SUN4 1
- UNIX4.2BSD のライブラリ times() がサポートされている場合、#define CPUTIME 60
- それ以外の場合 (CPU タイムは表示されない)、#define CPUTIME 0

1.1.3 ヒープオーバーフローが起こった場合

cu-Prolog には以下のデータ領域がある。

システムヒープ: プログラム節、新述語定義等。 大きさは SHEAP_SIZE (デフォルトは 20000)

ユーザヒープ: Prolog 反駁時の一時的な構造等。 大きさは HEAP_SIZE (デフォルトは 600000)

制約ヒープ: 制約・部分項の一時的な構造等。 大きさは CHEAP_SIZE (デフォルトは 25000)

環境スタック: Prolog 反駁時の環境。 大きさは ESP_SIZE (デフォルトは 500000)

ユーザスタック: Prolog 反駁時のポインタのつけかえ等。 大きさは USTACK_SIZE (デフォルトは 10000)

文字列ヒープ: 文字列の格納領域。 大きさは NAME_SIZE (デフォルトは 50000)

実行中にこれらのヒープ・スタックのオーバーフローが頻繁に起こるならば、サイズを大きくしてコンパイルし直すとよい。

Macintosh 版では、MacCUP アイコンのインフォメーションで使用するメモリの大きさを設定することで、上記の領域は適当に確保される。

1.1.4 他システムへの移植

cu-Prolog 第三版は UNIX4.2/3BSD(特に Sun3.Sun4.Symmetry) 上の C 言語により実装されている。OS に依存するライブラリは CPU 消費時間計測[1]のみなので、他システムの移植は容易であろう。

1.2 起動法・終了法

cu-Prolog を起動するには、OS のプロンプトから cuprolog と打ち込む。また起動と同時に初期プログラムを読み込む場合には

```
cuprolog ファイル名
```

とする。

cu-Prolog を終了するにはトップレベルから、%0 または:-halt. とする。

1.3 第二版からの改良点

cu-Prolog 第三版は、cu-Prolog 第二版 [5] と比べ主として以下のような点で改良がなされている。

- 制約変換のモジュール化・マニュアル制約変換導入
- 部分項 (PST) データ構造の導入, それに伴う制約解消系の拡張
- 組み込み述語の拡充 (含オペレータ)

2 cu-Prolog プログラムの記述

2.1 用語の説明

項: アトム、変数、複合項、部分項

アトム: 定数、ストリング、数値

定数: 英小文字で始まる文字列、' で囲まれた任意の文字列

[1] mainsub.c の settimer() および printtime() で、Sun4 の clock()、または BSD の times() を使用している。

ストリング: "で囲まれた任意の文字列

数値: 整数、 . :小数点

変数: 英大文字または _ で始まる文字列。 _ のみは無名変数と呼ばれ、任意の二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、 $t_1, t_2, \cdots, t_n$ を項としたとき、 $p(t_1, t_2, \cdots, t_n)$ を複合項と言う。他の項の引数に現れる複合項を関数と呼び、そのときの p をファンクタ、それ以外の複合項を述語と呼び、そのときの p を述語名と言う。なお、第三版より単項演算子 (not)、二項演算子 (==,<,> 等) もサポートされた。

部分項 (PST): 素性名 / 素性値の並びを {} で囲んだもの。素性名は英小文字で始まる文字列、素性値は項を取る。

コメント (行): %% で始まる行、または * と *\ で囲まれた文字列 (この場合コメントの入れ子は許さない)

2.2 制約付ホーン節 (Constraint Added Horn Clause: CAHC)

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. $H : C_1, \cdots, C_n.$ **(事実節)**
2. $H :- B_1, \cdots, B_m : C_1, \cdots, C_n.$ **(ルール節)**
3. $:- B_1, \cdots, B_n : C_1, \cdots, C_n.$ **(質問節)**

H は頭部 (ヘッド)、 $B_1, \cdots, B_n$ は本体、 $C_1, \cdots, C_n$ は制約部と呼ぶ。

なお、プログラムの本体で素式を表す変数も許している。これにより、

```
call(X) :- X.
not(X)  :- X, !, fail.
not(_).
```

と、 call/1, not/1 が定義できる。

2.3 部分項 (PST) の記述

cu-Prolog 第三版では部分項 (Partially Specified Term: PST) を項として使用することができる。部分項は次のように、ラベルと値を / をデリミタとした対の不定個の並びを中括弧で囲んだものである。ラベルはアトム、値は項をとる。

```
{l1/a, l2/X, l3/{f/b,g/c}}
```

部分項の導入により、単一化が拡張される。部分項 X,Y の単一化結果が部分項 Z の場合、以下が成り立つ。

- $\exists l. l/v \in X, l \notin Y \rightarrow l/v \in Z$
- $\exists l. l/v \in Y, l \notin X \rightarrow l/v \in Z$
- $\exists l. l/v \in X, l/u \in Y \rightarrow l/unify(u,v) \in Z$

例えば、 {l/a,m/X} と {m/b,n/c} の単一化結果は {l/a,m/b,n/c} となる。

同一の部分項がプログラム中に複数出現する場合には、媒介変数を用いて制約により表示される。例えば

```
f(X) :- g1(_p1,X),g2(_p2,X); _p1={f/a,g/c}.
```

のようになる。cu-Prolog の %w コマンドで書き出されたファイル中では、部分項を表す変数は上のように _p1, _p2, ··· と表される。このような媒介変数は %R コマンド (制約のプリプロセス) によって、部分項へのポインタへと書き換えられる。

2.4 制約の記述

制約は以下に定義されるモジュラーという標準形 (に書き換えられるもの) でなければならない。プログラムの制約部分を制約を標準形に書き換えるには、プリコンパイル機構 (システムの %P コマンド) が用意されている。

[定義] 1 (モジュラー) 素式列 $C_1, C_2, \ldots, C_m$ は以下の全ての条件を満たす時、モジュラーであるという。

1. C_i の全ての引数は変数 $(1 \leq i \leq m)$ である。
2. どの変数も二個所以上の空虚でない引数位置には現れない。
3. C_i はモジュラー定義述語から成る $(1 \leq i \leq m)$。 □

ただし、空虚な引数、およびモジュラー述語は以下に定義される。

[定義] 2 (空虚な引数位置) 述語 p の全ての定義節において p の第 i 引数が変数である時、p の第 i 引数は空虚であるという。述語 p による *unfolding* によって、第 i 引数はそれ自体で何かに束縛しないことを表している。 □

[定義] 3 (モジュラー述語) 述語 p の全ての定義節の本体が、モジュラーまたは *nil* の場合、p はモジュラー述語であるという。 □

PST の導入により、制約の標準形を一部拡張した。

[定義] 4 (引数の成分) 述語の各引数位置の成分 *(component)* とは、その引数が具体化し得る *PST* のラベルの集合である。通常項 *(アトム, リスト, 複合項)* はパス [] の *PST* と見なす。

述語 p の第 n 引数の成分を Cmp(p,n) と書き、また Cmp(T) により項 T のパスの集合を表す。また、特殊な X=t なる形の制約に対しては、変数 X は成分が Cmp(t) の引数位置にあるとみなす。

引数の成分はプログラムを静的に解析することにより得られる [6]。述語の引数成分は、%d コマンド (述語定義の表示) にて見ることが出来る。以下の例では、述語 p3/2 の第 1 引数の成分は {f,h}, 第 2 引数の成分は {g,h} である。

```
_%d p3
%d +---------( p3/2 )-----[f.h|g.h]--2/2--+
p3({f/a}, {g/b}).
p3({h/c}, {h/d}).
```

[定義] 5 (依存関係) 制約 *(素式列)* において、

1. 同一変数が共通の成分を持つ複数の引数位置に出現している場合、または
2. 同一変数が成分 {[]} の引数と、成分が *PST* のパスのみからなる引数位置に出現している場合、または
3. 成分が ϕ でない引数位置が具体化されている場合

は依存関係があると言う。 □

例えば、述語 p の第 1 引数成分が {f,g}, 述語 q の第 1 引数成分が {h} の場合、制約 p(X),q(X)} や p({u/a,v/b}) は依存関係を持たない。

[定義] 6 (モジュラー (再)) 依存関係の存在しない制約をモジュラーという。

3 BNF 記法による cu-Prolog の構文

以下は cu-Prolog の構文の BNF による記述である。

$$
\begin{aligned}
< char > &::= < capital > | < small > | < digit > \\
< charwhite > &::= < char > | < space > \\
< capital > &::= A|B|C|\ldots|X|Y|Z \\
< small > &::= a|b|c|\ldots|x|y|z \\
< digit > &::= 0|1|2|3|4|5|6|7|8|9 \\
< series > &::= < digit > | < digit >< series > \\
< number > &::= < series > | < series > . < series > \\
< charseq > &::= < char > | < char >< charseq > \\
< cwseq > &::= < charwhite > | < charwhite >< cwseq > \\
< string > &::= " < cwseq > " \\
< smallseq > &::= < small > | < small >< charseq > \\
< capitalseq > &::= < capital > | < capital >< charseq > \\
< term > &::= < var > | < atom > | < smallseq > (< term\_list >)| < PST > \\
< term\_list > &::= < term > | < term >, < term\_list > \\
< var > &::= < capitalseq > |\_ < charseq > |\_ \\
< atom > &::= < constant > | < string > | < number > \\
< constant > &::= < smallseq > |' < cwseq >' \\
< PST > &::= < pair\_list > \\
< pair\_list > &::= < pair > | < pair >, < pair\_list > \\
< pair > &::= < name > / < term > \\
< af > &::= < smallseq > (< term\_list >)| < smallseq > | < op\_term > | < var > \\
< af\_list > &::= < af > | < af >, < af\_list > \\
< HORN > &::= < af > | < af >:- < af\_list > |?- < af\_list > \\
< CAHC > &::= < horn > .| < horn >:< af\_list > . \\
< op\_term > &::= < op1 >< term > | < term >< op2 >< term > \\
< op1 > &::= not \\
< op2 > &::= <=> | = | = .. | > | >= | < | <= | ==
\end{aligned}
$$

4 システムコマンド一覧

cu-Prolog のトップレベルのコマンドには以下のものが用意されている。ここで、*predicate* は、*predicate_name* または *predicate_name/arity* を表すとする。

4.1 Prolog に関するコマンド

%h	ヘルプ
# *OS_command*	OS のコマンドを実行する
%d *predicate*	述語の定義を表示する
%d *	全述語の定義を表示する
%d /	述語名を表示する (簡約された述語も含む)
%d ?	述語名を表示する (簡約された述語は含まない)
	システム組み込み述語には +:recursive, ":functor
	ユーザ述語には *:spied,-:reduced,+:recursive,#: 制約変換による新述語
%f	システムヒープの使用状況を表示
%Q	cu-Prolog を終了する
%R	システムリセット
%G	(静的) ガーベジコレクション
%c 数値	推論の深さの上限を設定する。これより深いゴールは失敗とする

%u　　未定義述語をエラー (TURE) にするか、失敗 (FALSE) にするかのスイッチ

4.2 ファイル入出力についてのコマンド

"filename"	プログラムファイルをエコーバックなしで読み込む
"filename?	プログラムファイルをエコーバックつきで読み込む
%l *filename*	ログファイルを設定する
%l no	ログファイルへの記録を中止する
%w *filename*	プログラムをファイルに書き出す

4.3 デバッグコマンド

%p *predicate*	述語のスパイフラグを設定 / 解除するスイッチ
%p *	全述語のスパイフラグを設定する
%p .	全述語のスパイフラグを解消する
%p >	制約変換にスパイフラグを設定 / 解除するスイッチ
%p ?	スパイフラグが設定されている述語名を表示する
%t	ノーマルトレースモード on/off のスイッチ
%s	ステップトレースモード on/off のスイッチ

4.4 制約変換に関するコマンド

%L	制約変換で作られた新述語の導入定義節を表示する
%n *predicate*	制約変換で作られる新述語名を設定する (デフォルトは c)
%a	制約変換を全モジュラーモードにする
%o	制約変換を M-solvable モード[2]にする
%P *predicate*	述語の定義節の制約部を前処理する (標準形に書き換える)。
%P *	全ての述語の定義節の制約部を前処理する
%P ?	標準形でない制約を持つ定義節を表示する

4.5 その他のコマンド

%C　JPSG パーザ用組込みファンクタ cat() を再定義する。

4.6 Macintosh 版 cu-Prolog メニュー

Macintosh 版では、システムコマンドの多くがメニューから実行できる。

4.6.1 File メニュー

Open	cu-Prolog のプログラムファイルを読み込む
Save Predicates	プログラムをファイルに書き出す (%w)
Quit	cu-Prolog を終了する (%Q)

4.6.2 Command メニュー

Help	ヘルプを表示する (%h)
Show Predicate	述語定義を表示する (%d)
Set Log File	ログファイルを設定・解除する (%l)
Preprocess Constraint	制約を前処理する (%P)
Max # of Refutation	Prolog 推論の段数の最大を設定する (%c)
List New Predicates	制約変換で作られた新述語の導入節を表示する (%L)
Reset	cu-Prolog のリセット (%R)
Handling Undef	未定義述語をエラー (true), 失敗 (false) にするかを切り替える (%u)
Print Level	表示のレベルを設定する
Garbage Collection	静的ガーベジコレクションを行う (未サポート %G)
New Predicate Name	制約変換でできる新述語の名前を再定義する (%n)
Max # of Vars in Trans	制約変換で一制約中の変数の最大個数を設定する。これを越える制約は処理しない。

4.6.3 Trace メニュー

Step Trace	ステップトレースモードにする (%s)
Normal Trace	ノーマルトレースモードにする (%t)
Trace Off	ノートレースにする
Marking Spy	述語のスパイフラグを設定する (%p)

4.6.4 MODE メニュー

All Modular mode	制約を全部モジュラーに変換する (%a)
M-solvable mode	制約を M-solvable モードで変換する (定義節の少なくとも一つがモジュラーになった時点で制約変換を打ち切る)

5 cu-Prolog 組込み述語・ファンクタ

5.1 一般的な組込み述語

cu-Prolog 第三版では、以下の組込み述語が用意されている。引数のモードは、＋は具体化された項 (入力引数)、－は自由変数 (出力引数) を表す。引数のモードが満たされていない組み込み述語のゴールは直ちに失敗する。引数の PST は部分項を取ることを表している。

5.1.1 関数的な組み込み述語

解が一つ (バックトラックしない) の述語である。

述語	動作
!/0	カット
abolish/2	abolish(F+,A+) 述語 F/A の定義を消去する
arg/3	arg(Pos+,Fun+,Arg-) Fun の Pos 番目の引数を Arg にバインドする 例:arg(2,test(a,b,c),X) では X=b
assert/1,2,3	assert(Head+), assert(Head+,Body+), assert(Head+,Body+,Cstr+) 節 Head:-Body;Cstr をプログラムの最後に付け加える
asserta/1,2,3	節をプログラムの最初に付け加える
assertz/1,2,3	節をプログラムの最後に付け加える
close/1	close(F) ファイルポインタ F で指定されるファイルを閉じる
compare/3	compare(X+,Y+,C-) X,Y が共に数値または文字列の場合、それらの大小関係を返す。 C は>,==,<のいずれかに具体化する。
concat2/2	concat2(Str+,List-) 文字列 Str を一文字毎に分解したリストを List にバインドする concat2("ab",["a","b"])
count/1	count(X-) X に毎回異なる正整数 (0,1,2,…) をバインドする
default	default(X,Y+,Z+) 部分項 X が Y を含まなければ fail, そうでなければ X に Z が追加される。 例: :- X={pos/p,sem/Y},default(X,{pos/p},{ajn/□,refl/□}). ならば X={pos/p,ajn/□,refl/□,sem/Y} :- X={pos/n,sem/Y},default(X,{pos/p},{ajn/□,refl/□}). は fail
divstr/4	divstr(X+,N+,Y-,Z-) 文字列 X の N 番目から前を Y に後を Z にバインドする
equal/2	equal(X,Y) X と Y を unify する
eq/2	eq(X,Y) X が Y と等しいかチェックする
fail/0	常に失敗する
functor/3	functor(H+,F-,A-) H の述語が F/A
gensym/1	gensym(X-) LISP の gensym と同様、毎回異なるアトムと X をバインドする 初期値はコマンド %n で変えられる (デフォルトは c0)。
geq/2	geq(X+,Y+) 数値比較 (X>=Y)
greater/2	greater(X+,Y+) 数値比較 (X > Y)
halt/0	cu-Prolog を終了する
leq/2	leq(X+,Y+) 数値比較 (X=<Y)
less/2	less(X+,Y+) 数値比較 (X < Y)

ml/2	univ. 述語 ml(T+,L-) は T=..L と等価
	例: :-ml(f[a,b],U). ならば U=[f,a,b]
multiply/3	multiply(X+,Y+,Z-) 数値乗算 X * Y = Z
name/2(A,L)	name(A+,L-) 文字列 A を文字のリスト L に変換する
nl/0	'\n' を表示する
op/3	op(P+,T+,Op+) 演算子またはそのリスト Op を precedence を P、 type を T として定義する。
	例: :-op(700,xfx,'=').
open/3	ストリームを開く
pnames/2	pnames(PST+,Flist-) PST の素性名のリストを Flist にバインドする
	:-pnames(1/a,m/b,X). ならば X=m,1.
pvalue/3	pvalue(PST+,Fname+,V-) PST の素性 Fname の値を V にバインドする
read/1,2	ストリームファイルから読み込む
reset_timer/0	CPU タイマをリセットする。
see/1	see(F-) F が現在の入力ストリームになる
seen/0	現在の入力ストリームを閉じる
strcmp/3	strcmp(X+,Y+,C-) 文字列 X,Y の大小関係を C にバインドする (compare 参照)。
strlen/2	strlen(S+,L-) L に文字列 S の長さをバインドする
substring/3	substring(X+,N+,Y-) 文字列 X の N 番目からを Y にバインドする。
	(N が負の場合は文字列の末尾から数える)
substring/4	substring(X+,N+,L+,Y-) 文字列 X の N 番目から L 個の文字列を Y にバインドする。
sum/3	sum(X+,Y+,Z-) 数値加算 X + Y = Z
tab/0,1	タブを表示する
tell/1	tell(T-) T が現在の出力ストリームになる
timer/2	timer(X-,Y-)
	X には直前の reset_timer 以後の CPU 時間、
	Y にはそのうち制約変換に使われた CPU 時間をバインドする。
told/0	現在の出力ストリームを閉じる
true/0	常に成功する
unbreak/0	ステップトレースで break した後、:-unbreak. でその時点に戻る
var/1	var(T+) T が自由変数である
write/1	write(T+) 項 T を表示する

5.1.2 述語的な組み込み述語

解が複数ありうる (バックトラックする) 組み込み述語である。

述語	動作
clause/3	clause(Head+,Body-,Cstr-)
	Head とヘッドユニファイする節の本体と Body を、制約と Cstr をバインドする
concat/3	concat(S1,S2,S) 文字列 S1 と S2 を連結したものが S.
	例: concat("ab","cd","abcd")
execute/1	execute(Goal+) Goal(リスト) で与えられた素式列を順次実行する
	例: execute([memb(X,[a,b]),memb(X,[b,c])])
isop/3	定義されている演算子の (precedence,type,operator) の組を返す。
	例: :-isop(X,Y,Z). に対し X=900, Y=xfy, Z='/'
memb/2	member(Element,List) 組み込み版 member
or/2,3,4,5	複数のゴールを実行
retract/1,2,3	retract(Head+), retract(Head+,Body+), retract(Head+,Body+,Cstr+)
	Head:-Body;Cstr と単一化する節を一つ消去する

5.2 制約変換に関する組込み述語

述語	動作
condname/2	condname(Cstr+,Plist-) 制約 Cstr に含まれる述語のリストを Plist とバインドする
	condname([c0(X,Y),c1(Y,Z)],X) なら X=[c0,c1]

pcon/0	この時点で残っている制約を表示する
unify/2	unify(C+,NC-) C を制約変換したものを NC にバインドする C, NC は [c0(X,Y),c1(Y,a)] のような素式のリスト

5.3 JPSG パーザに関する組み込み述語・ファンクタ

cu-Prolog によりユニフィケーション文法の属性構造を記述する場合、部分項を用いるのが適当であるが、あらかじめ全ての素性のセットが分かっている場合には次に示す cat という特別なファンクタも便利である。

述語・ファンクタ	動作
cat/6	JPSG の文法範疇を表すファンクタ
t/3	t(M,L,R) 履歴を表すファンクタ
tree/1	tree(H+) 履歴 H を木構造で表示する

JPSG 等の文法範疇の実装に便利な cat() という特別なファンクタが用意されている。tree() により適当にプリティプリントされる。cat のアリティは次の %C コマンドでトップレベルから変更可能である。ただし %C コマンドは一度しか実行できない。

%C [素性名 1, 素性型 1, 素性名 2, 素性型 2,...]

と素性名と素性型をリストで囲んで指定する。

ただし、素性名は大文字から始まり 5 文字以内でなければならない。また、素性型は、1,2,3 の数値で指定する。ここで、

	素性型	
1	Normal	2,3 以外
2	SingleCat	値として文法範疇を一つ取る素性 (Adjacent 等)
3	SetCat	値として文法範疇の集合を取る素性 (Subcat 等)

デフォルトでは、[POS,1,FORM,1,AJA,2,AJN,2,SC,3,SEM,1] つまり

素性名	素性型	対応する JPSG の素性
POS	Normal	pos
FORM	Normal	gr.vform,pform, and so on
AJA	SingleCat	adjacent
AJN	SingleCat	adjunct
SC	SetCat	subcat
SEM	Normal	sem

となっている。

また tree コマンドにより cat ファンクタは

第 1 素性値 [第 2 素性値, 第 3 素性名: 第 3 素性値, ···]: 最終素性値

の順に、さらに素性値が [] の素性は省いてプリントされる。

t/3 は履歴を表す特別なファンクタである。履歴は以下のように定義される。

[定義] 7 (履歴) 文法範疇は履歴

- C と W が文法範疇の時 t(C,W,[ ]) は履歴
- L と R が履歴、M が文法範疇の時、t(M,L,R) は履歴
- L と R が履歴、C と W が文法範疇の時、t(t(C,W,[ ]),L,R) は履歴 □

履歴は tree() 述語により木構造で表示される。tree(t(C,W,[ ])) は

C--W

を出力し、tree(t(Mother,LeftDaughter,cat(n,ga,□,□,□,ken))) は

```
---Mother
 |
 |-LeftDaughter
 |
 |-n[ga]:ken
```

を出力する。

6 ファイルの入出力

6.1 プログラムの読み込み

- システム起動と同時に読み込むには、OS のプロンプトから

  cuprolog *filename*

  とする。エコーバックはない。
- トップレベルからエコーバックなしで読み込む

  "*filename*"
- トップレベルからエコーバックありで読み込む (デバッグ時など)

  "*filename*?

6.2 プログラムの保存

プログラムをファイルに保存するには、トップレベルから

%w *filename*

とする。

6.3 ログファイルの設定、解除

- ログファイル名を設定する。これ以後の画面はすべて設定されたファイルに格納される。

  %l *filename*
- ログの中止

  %l no

7 制約変換機構

7.1 @ モード

トップレベルから制約変換機構を単独で使うことができる (主にデバッグ用)。

```
@ member(X,[a,b,c]),member(X,[b,c,d]).
```

とすると、cu-Prolog はこれと等価でモジュラーな制約 (例えば c0(X)) と、制約変換の際に作られた新述語の定義を返す。

以下は @ モードの実行例である。

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

_@ member(X,[a,b,c]).       % モジュラーでない制約の入力 1

solution = c0(X_0)          % 変換された制約
c0(a).            % 制約変換で作られた新述語 c0 の定義
c0(b).
```

```
c0(c).

_@ member(A,X),append(X,Y,Z).        % 制約の入力 2
solution = c1(A_0, X_1, Y_2, Z_3)  % 変換された制約
                    % 制約変換で作られた新述語 c1,c2,c3 の定義
c3(V0_0, V3_3, V1_1, [V3_3 | V2_2]) :- append(V0_0, V1_1, V2_2).
c2(V0_0, V1_1, V4_4, V2_2, [V4_4 | V3_3]) :- c1(V0_0, V1_1, V2_2, V3_3).
c1(V1_1, [V1_1 | V0_0], V2_2, V3_3) :- c3(V0_0, V1_1, V2_2, V3_3).
c1(V0_0, [V2_2 | V1_1], V3_3, V4_4) :- c2(V0_0, V1_1, V2_2, V3_3, V4_4).
```

7.2 unify(C,NC)

組み込み述語 unify により Prolog 上で変換ルーチンを陽に使うこともできる。この場合、制約は次のように素式を要素とするリストで記述する。

```
[c0(X,[a,b,c]), c1(P,Q,R), c2(Q,S)]
```

unify(OldCond+, NewCond-) は、OldCond がリスト形式の制約に具体化されていて、NewCond が自由変数の場合にのみ制約変換を実行。NewCond には、OldCond と等価でモジュラーな制約がバインドされる。

7.3 制約変換の操作

制約変換でできる新述語の節の集合 (以下「節集合」と呼ぶ) として、以下の 3 つを用意する。

DEFINITION 新述語の導入節

NON-MODULAR 新定義節のうちモジュラーでない節

MODULAR 新定義節のうちモジュラーな節

モジュラーでない制約を C、C に含まれる変数を $X_1, \cdots, X_n$、p を n 引数の新述語名とする。制約変換ルーチンは

$p(X_1, \cdots, X_n)$==C.

を DEFINITION に付加し、以下の三つの基本操作を繰り返す。そして、DEFINITION および NON-MODULAR を空にすることができた場合には制約変換成功であり、途中で基本操作が続かなくなった場合には制約変換失敗となる。制約変換成功の場合、MODULAR 中の節が新述語の定義節で、C は $p(X_1, \cdots, X_n)$ に書き換えられたことになる。

なお現実装では、制約変換の前後で reduction 操作 (定義節が一つの述語は強制的に展開する) も行っている。

制約変換は次の三つの基本操作からなる。

1. *unfolding*

   NON-MODULAR または DEFINITION から一つの節 C を取り除き、節 C の本体から一つのリテラル L を選び、展開を行う。展開による新定義節のうちモジュラーでない節は NON-MODULAR に、モジュラー節は MODULAR に追加される。

2. *folding*

   NON-MODULAR の節 C の本体より、残りの本体と共有変数を持たないリテラル列 $L = l_1, \cdots, l_n$ を選ぶ。DEFINITION の中の一つの節 D の本体が変数の置換により L と同一なら、C の L を D の頭部 (の変数を置換したもの) と置換する。

3. *modularize*

   NON-MODULAR から一つの節 H:-B. を取り除き、B を変数による同値類 $B_1, \cdots, B_n$ に分割する。B_i に含まれる変数を $X_{i,1}, \cdots, X_{i,m}$、p_i を新述語として、$P_i = p_i(X_{i,1}, \cdots, X_{i,m})$ とする。$1 \leq i \leq n$ について P_i == B_i. を DEFINITION に追加し、MODULAR に H:-$P_1, \cdots, P_n$. を追加する。

7.4 制約変換のトレース

制約変換のトレースは、まず %p > によって制約変換にスパイをかけた後に、%t または %s により、ノーマルトレースかステップトレースかを指定する。

7.4.1 トレースの表示

DEFINITION に含まれる節 (新述語の導入節) は、以下の書式で表示される。

[節番号 (節状態. 述語定義数)] 導入節
`[1(d,0)] c0(X) <=> member(X,[a,b,c]).` (例)

ここで、節状態は以下の記号で表す。

r removed: unfolding により消去され (以下の f,g でない) 節
d derivation: 導入節で unfolding されていないもの
g registered: 新述語の定義に一つ以上の単位節が得られたもの
f false_registerd: 新述語の定義が空となったもの

NON-MODULAR および MODULAR に含まれる節 (新しく定義された節) は、以下の書式で表示される。

<節番号 (節状態. 述語定義数)> 新定義節
`<2(u)> c0(X):-member(X,[b,c]).` (例)

ここで、節状態は以下の記号で表す。

r removed: 簡約化操作、unfolding により消去された節
u untouched: NON-MODULAR の節
m modular: 本体に変数の共有関係がない節
i unit: 単位節

7.4.2 ステップトレースのユーザ入力

ステップトレースでは、トレース表示に次いでユーザの入力待ちになる。以下によりコントロールを指定する。

入力	動作
リターン	継続: デフォルトのヒューリスティックスで節とリテラルを選択し継続する
u CN LN	マニュアル unfolding:CN で節番号、LN で本体の何番目を展開するかを指定する。
s	トレースの中止: 以後の変換過程は表示されない。
n	ステップトレースの中止: 以後ノーマルトレースとなる。
q	制約変換の中断: その時点で新定義節のうち消去されていないものをアサートして制約変換を終了する。
z	制約変換の中止: 新定義節を消去後、制約変換を終了する。

7.5 制約変換のヒューリスティック

現行の cu-Prolog では制約変換を次の手順で行っている。

1. DEFINITION に節がある場合には、DEFINITION より一つの節 C を選び unfolding する。節 C は DEFINITION より取り除く。
2. DEFINITION が空の場合には、NON-MODULAR より節 N を選び、N の頭部において全ての引数が相異なる変数の場合には unfolding を行う。
3. そうでない場合は節 N の本体を folding または modularize する。
4. 以上の操作を、DEFINITION および NON-MODULAR が空になるまで繰り返す。

したがって、ヒューリスティックとしては

- DEFINITION からの節の選び方
- NON-MODULAR からの節の選び方
- unfolding において展開するリテラルの選び方

が考えられる。

7.5.1 DEFINITION からの節の選択

DEFINITION はスタックで実装しており、最も新しく作られたものを最初に選んでいる。

7.5.2 NON-MODULAR からの節の選択

NON-MODULAR はスタックで実装しており、最も新しく作られたものを最初に選んでいる。

7.5.3 unfolding におけるリテラルの選択

unfolding におけるリテラルの選択に関するヒューリスティックは、現行では以下の要素により各リテラルの活性係数を計算し、活性係数の最も高いものから展開している。

$$
\begin{aligned}
Arity &= \text{述語の引数の個数} \\
Const &= \text{引数のうち定数に具体化しているものの個数} \\
Vnum &= \text{引数に現れる自由変数が制約全体の中で何回出現しているかの個数} \\
Funct &= \text{引数のうち(変数を含む)複合項に具体化しているものの個数} \\
Rec &= \text{述語が再帰的なら1、そうでなければ0} \\
Defs &= \text{述語の定義節の個数} \\
Units &= \text{述語の定義節のうち単位節(本体が空)の個数} \\
Facts &= \text{述語定義がすべて単位節からなる場合に1、そうでなければ0} \\
\text{活性係数} &= 3*Const + 2*Funct + Vnum - Defs + Units - 2*Rec + 3*Facts
\end{aligned}
$$

活性係数は、cu-Prolog の制約変換に関して今まで経験的に得られていたヒューリスティックを含むように決定した。さらに要素を増やして実験を行うことでより有効なヒューリスティックも得られると思われる。

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。最初に述語にスパイフラグを設定してから、次のいずれかのトレースモードを選ぶ。

ノーマルトレース: (プロンプトは $ になる) 途中でストップしない

ステップトレース: (プロンプトは> になる) 実行を一時停止し、入力待ちになる。

制約変換のトレースについては「制約変換」の項を参考のこと。

8.1 スパイフラグの設定

%p *	全述語にスパイフラグを設定する
%p .	全述語のスパイフラグを解除する
%p *predicate*	述語のスパイフラグをスイッチする
%p >	制約変換のスパイフラグをスイッチする
%p ?	スパイされている述語名を表示する

8.2 トレースモード

トレースモードに入るスイッチは以下が用意されている。トレースモードから以下のコマンドを再び行うとノートレースモードに戻ることができる。

%s ステップトレースのスイッチ。プロンプトは'>' になる

%t ノーマルトレースのスイッチ。プロンプトは'$' になる

ステップトレースモードでは、ゴールを表示してからユーザの入力待ちになる。以下によりコマンドを指定する。

入力	動作
リターン	実行を継続する
s	最左ゴールが終了するまでトレースを省略する
a	親ゴールを表示する
b	ブレーク。一時的にトップレベルに移る。:-unbreak によりこの時点に戻ることができる。
f	現在のゴールを fail させる。
l	リープ。このゴールの終了まで飛ぶ。
z	実行を中止する

9 JPSG パーザ

JPSG(Japanese Phrase Structure Grammar) 等の単一化文法のパーザを、cu-Prolog の CAHC で記述することを考えよう。JPSG における制約は、局所的な枝分かれにおける文法範疇 (親、左右娘) の各素性の関係として宣言的に記述されている。cu-Prolog では辞書や句構造規則を表すプログラム節に、ユーザ定義述語による制約を直接付加できるので、JPSG の制約を自然に記述することが可能である。パーザの処理効率を考えると、効率の良いアルゴリズムの分かっている構文木作成部分はプログラムの本体で記述し、曖昧性に関わる部分を制約で記述するのが望ましい。また、部分項により選言を含まない素性構造を表し、選言的な部分は制約で付加することで、disjunctive feature structure も cu-Prolog で容易に記述できる。

ここでは JPSG に基づく簡単な日本語パーザにおいて、CAHC の二種類の効果的な使い方を紹介する。

9.1 制約による曖昧性の処理

制約により同音異義語等の語彙的曖昧性の処理を簡単に行うことができる。例えば、名詞 "はし" は、状況に応じて橋、箸、端などの意味を取る。このような曖昧性を持つ語彙も、CAHC では次のように一つの辞書エントリで表すことができる。

```
lexicon(hasi,{pos/n,form/n,sem/SEM}):hasi_sem(SEM).
```

ここで、制約 hasi_sem は、次のように定義されているとする。

```
hasi_sem({type/structure,obj/bridge}).
hasi_sem({type/tool,obj/chopsticks}).
```

意味素性の値には変数 SEM が含まれており、それには制約 hasi_sem(SEM) が課せられている。これは次の disjunctive feature structure に相当する。

$$\left[\begin{array}{l} pos : n \\ form : n \\ sem : \left\{ \left[\begin{array}{l} type : structure \\ obj : bridge \end{array}\right] \left[\begin{array}{l} type : tool \\ obj : chopsticks \end{array}\right] \right\} \end{array}\right]$$

辞書の段階ではこのように曖昧でも、処理が進んでいく間に変数に別の制約が課せられ、それらの制約を解消することで曖昧性が減少していく。先の「はし」の例では、文の別の部分の処理において変数 OBJ が tool に具体化したとすると、その時点で制約が解消され、変数 TYPE も chopsticks に具体化することになる。従来の Prolog 等では、同音語の辞書はそれぞれの意味により別々の辞書エントリとして記述され、一つの可能性が失敗する度にバックトラックを行うことになり効率が悪い。同様にして、動詞の活用語尾や、後置詞の接続の記述も扱っている。

9.2 JPSG の制約の記述

もう一つは、JPSG の句構造規則における構造原理を制約として埋め込むことができる。前述のように、局所的な枝分かれにおける文法範疇の各素性の関係を制約として記述できればよい。

JPSG では [1]、FFP(束縛素性原理) は次のようになっている。

> 局所的な枝分かれにおいて、親の束縛素性の値は、子の束縛素性の値の和と単一化する。

CAHC を用いると、この原理は次のように句構造規則に埋め込むことができる。

```
psr([foot(MS)],[foot(LDS)],[foot(RDS)]):union(LDS,RDS,MS).
```

しかし、Prolog でこの種の制約を記述した場合に、実行は手続き的 (psr か union のどちらかが先に実行される) になってしまい、変数の具体化の状況によっては効率が悪くなる。

9.3 素性

JPSG パーザでは、JPSG の以下の素性を取り扱っている。

(素性名)	(素性型)	(取り得る値)	(用途)
pos	主辞素性	n,v,p	品詞
infl	主辞素性	vc,vv,···	品詞の活用の種類
form	主辞素性	root, sbj,obj, ···	品詞の活用形
gr	主辞素性	ga,wo,···	名詞の文法関係
mod	主辞素性	+,-	修飾可能性
dep	主辞素性	カテゴリの主辞素性	修飾するカテゴリ
view	主辞素性	asp(B,E,D)	視野・アスペクト・参照時間
adjacent	Subcat 素性	カテゴリ	直前に来るカテゴリ
sc	Subcat 素性	カテゴリ	補語
slash	束縛素性	カテゴリの主辞素性	文のギャップに相当
pslash	束縛素性	カテゴリの主辞素性	resumptive pronoun にて生じる
refl	束縛素性	カテゴリの主辞素性	「自分」にて生じる
sem			意味

9.4 実行例

以下は、cu-Prolog による JPSG パーザ (第 0 版) の実行例である。「健が奈緒美を愛する」には曖昧性がないので対応する文法範疇に付随する制約は空である。「健が愛する」は「健が (t を) 愛する」というギャップのある文、または「健が愛する (t)」という関係節、の二通りの読みがある。これらの曖昧性はトップレベルに付随する制約の複数の解として表されている。

```
tsuda#icot21[5]% cup3 % cu-Prolog の立ち上げ

*****  cu - Prolog   Ver. III  *****
Copyright: Institute for New Generation Computer Technology,  Japan 1989-91
in Cooperation with SIRAI@sccs.chukyo-u.ac.jp
All Modular mode (help ->  %h)

_"jpsg/j4.p" % JPSG パーザファイルを読み込む
=== open 'jpsg/j4.p'

 ****** end of file *******
CPU time = 2.050 sec
_:-p([ken,ga,naomi,wo,ai,suru]). % 「健が奈緒美を愛する」の解析
% 構文木が返る

{sem/[love,ken,naomi], core/{form/Form_1913, pos/v}, sc/[], refl/[],
 slash/[], psl/[], ajn/[], ajc/[]}---[suff_p]
 |
 |--{sem/[love,ken,naomi], core/{pos/v}, sc/[], refl/[], slash/[],
    psl/[], ajn/[], ajc/[]}---[subcat_p]
 |   |
 |   |--{sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[],
        psl/[], ajn/[], ajc/[]}---[adjacent_p]
 |   |   |
 |   |   |--{sem/ken, core/{form/n, pos/n}, sc/[], refl/[], slash/[],
            psl/[], ajn/[], ajc/[]}---[ken]
 |   |   |
 |   |   |__{sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[],
            psl/[], ajn/[], ajc/[{sem/ken, core/{pos/n}, sc/[],
            refl/ReflAC_70}]}---[ga]
 |   |
 |   |__{sem/[love,ken,naomi], core/{pos/v}, sc/[{sem/ken,
        core/{form/ga, pos/p}, sc/[], refl/[], slash/[], psl/[], ajn/[],
        ajc/[]}], refl/[], slash/[], psl/[], ajn/[], ajc/[]}---[subcat_p]
```

```
 |       |
 |       |--{sem/naomi, core/{form/wo, pos/p}, sc/[], refl/[],
 |           slash/[], psl/[], ajn/[], ajc/[]}---[adjacent_p]
 |       |   |
 |       |   |--{sem/naomi, core/{form/n, pos/n}, sc/[], refl/[],
 |           |   slash/[], psl/[], ajn/[], ajc/[]}---[naomi]
 |       |   |
 |       |   |__{sem/naomi, core/{form/wo, pos/p}, sc/[], refl/[],
 |           |   slash/[], psl/[], ajn/[], ajc/[{sem/naomi, core/{pos/n}, sc/[],
 |           |   refl/ReflAC_504}]}---[wo]
 |       |
 |       |__{sem/[love,ken,naomi], core/{form/vs2, pos/v}}---[ai]
 |
 |__{sem/[love,ken,naomi], core/{form/Form_1913, pos/v}, sc/[],
     refl/[], slash/[], psl/[], ajn/[], ajc/[{sem/[love,ken,naomi],
     core/{form/vs1, pos/v}, sc/[], refl/ReflAC_1929}]}---[suru]

category= {sem/[love,ken,naomi], core/{form/Form_3670, pos/v},
sc/[], refl/[], slash/[], psl/[], ajn/[], ajc/[]}
% トップレベルのカテゴリ
constraint= syu_ren(Form_3670) % それに付随する制約（終止形または連体
      % 形に対応する）
true.
CPU time = 2.200 sec (Constraints Handling = 1.900 sec)


_:-p([ken,ga,ai,suru]). %「健が愛する」の構文解析

{sem/[love,V7_1482,V6_1481], core/{form/Form_833, pos/v}, sc/V1_1476,
 refl/[], slash/V3_1478, psl/[], ajn/[], ajc/[]}---[suff_p]
 |
 |--{sem/[love,V7_1482,V6_1481], core/{pos/v}, sc/V0_1475, refl/[],
     slash/V2_1477, psl/[], ajn/[], ajc/[]}---[subcat_p]
 |   |
 |   |--{sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[],
 |       psl/[], ajn/[], ajc/[]}---[adjacent_p]
 |   |   |
 |   |   |--{sem/ken, core/{form/n, pos/n}, sc/[], refl/[], slash/[],
 |   |       psl/[], ajn/[], ajc/[]}---[ken]
 |   |   |
 |   |   |__{sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[],
 |   |       psl/[], ajn/[], ajc/[{sem/ken, core/{pos/n}, sc/[],
 |   |       refl/ReflAC_70}]}---[ga]
 |   |
 |   |__{sem/[love,V7_1482,V6_1481], core/{form/vs2, pos/v}}---[ai]
 |
 |__{sem/[love,V7_1482,V6_1481], core/{form/Form_833, pos/v}, sc/[],
     refl/[], slash/[], psl/[], ajn/[], ajc/[{sem/[love,V7_1482,V6_1481],
     core/{form/vs2, pos/v}, sc/[], refl/ReflAC_945}]}---[suru]

category= {sem/[love,V7_1482,V6_1481], core/{form/Form_833, pos/v},
    sc/V1_1476, refl/[], slash/V3_1478, psl/[], ajn/[], ajc/[]}
% トップレベルのカテゴリ

constraint= c58(V0_1475, V1_1476, V2_1477, V3_1478, V4_1479, V5_1480,
    {sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[], psl/[],
    ajn/[], ajc/[]}, V6_1481, {sem/V6_1481, core/{form/wo, pos/p}},
    V7_1482, {sem/V7_1482, core/{form/ga, pos/p}}),  syu_ren(Form_833)
% トップレベルカテゴリに付随する制約
true.
CPU time = 1.817 sec (Constraints Handling = 1.567 sec)
```

```
_:-c58(_, SC, _, SL, _, _:_ , Obj,_, Sbj,_).     % トップレベルの制約を解く
  SC = []  SL = [{sem/V0_4}]  Obj = V0_4  Sbj = ken;    % 解1
  SC = [{sem/V0_4, core/{form/wo, pos/p}}]  SL = []  Obj = V0_4  Sbj = ken; % 解2
% 解1では subcat の内容が slash に移っている。解2ではまだ subcat が残っている

CPU time = 0.000 sec (Constraints Handling = 0.000 sec)
```

参考文献

[1] Takao GUNJI. *Japanese Phrase Structure Grammar*. Reidel, Dordrecht, 1986.

[2] Robert T. Kasper. A Unification Method for Disjunctive Feature Descriptions. In *25th Annual Meeting of ACL*, pages 235–242, July 1987.

[3] Hiroshi TSUDA, Koiti HASIDA, and Hidetosi SIRAI. cu-Prolog and its application to a JPSG parser. In K.Furukawa, H.Tanaka, and T.Fujisaki, editors, *Logic Programming '89*, pages 134–143. Springer-Verlag LNAI-485, 1989.

[4] Hiroshi TSUDA, Koiti HASIDA, and Hidetosi SIRAI. JPSG Parser on Constraint Logic Programming. In *Proc. of 4th ACL European Chapter*, pages 95–102, 1989.

[5] 津田 宏, 橋田 浩一, 白井 英俊, and 安川 秀樹. cu-Prolog 第二版処理系仕様書. Technical Report ICOT-TM952, ICOT Technical Memorandum, 1990.

[6] 津田宏. cu-prolog による選言的素性構造. In 日本ソフトウェア科学会第 *8* 回大会論文集, pages 505–508, 1991.

[7] 津田宏, 橋田浩一, and 白井英俊. 制約充足としての構文解析 — 制約論理型言語 cu-prolog の応用. In 日本ソフトウェア科学会第 *6* 回大会論文集, pages 257–260, 1989.

<< cu-Prolog第三版 関数一覧 >>

関数・マクロのアルファベット順の一覧である。

関数エントリ	モジュール
#define ARG_EQ	<tr_sub.c>
#define ARG_FALSE	<tr_sub.c>
#define ARG_TRUE	<tr_sub.c>
#define ATOMIC_TYPE	<include.h>
#define Arg(T,N)	<include.h>
#define Arg1(T)	<include.h>
#define Arg2(T)	<include.h>
#define Arg3(T)	<include.h>
#define BACKTRACK	<include.h>
#define BL	<print.c>
#define BL	<read.c>
#define BL	<varset.h>
#define BR	<print.c>
#define BR	<read.c>
#define BR	<varset.h>
#define BRACKET	<include.h>
#define CATMAX	<jpsgsub.c>
#define CHEAP_SIZE	<include.h>
#define CLAUSE_TYPE	<include.h>
#define CM	<print.c>
#define CM	<read.c>
#define CM	<varset.h>
#define CO	<print.c>
#define CO	<read.c>
#define CO	<varset.h>
#define COMMA	<include.h>
#define COMPONENT_CHECKED	<include.h>
#define CONSTANT_TERM	<include.h>
#define CONST_LIST_TYPE	<include.h>
#define CONST_MARK	<include.h>
#define COPYRIGHT	<main.c>
#define CPUTIME	<include.h>
#define CT	<print.c>
#define CT	<read.c>
#define CT	<varset.h>
#define CTnormal	<include.h>
#define CTnotrace	<include.h>
#define CTstep	<include.h>
#define C_BLINK	<include.h>
#define C_CLS	<include.h>
#define C_HIGHLIGHT	<include.h>
#define C_LOAD	<include.h>
#define C_NORMAL	<include.h>
#define C_REVERSE	<include.h>
#define C_SAVE	<include.h>

```
#define C_UNDER          <include.h>
#define CatSet           <jpsgsub.c>
#define CatSingle        <jpsgsub.c>
#define CnstMax          <jpsgsub.c>
#define Component(F,N)   <include.h>
struct term *CtoL(nbuf,flag)    <syspred1.c>
#define DEBUG            <new.c>
#define DEBUG            <print.c>
#define DEBUG            <tr_split.c>
#define DEBUG            <tr_sub.c>
#define DEBUG            <trans.c>
#define DERIVATION       <include.h>
#define DOWN             <include.h>
#define DUMMY_DEF        <include.h>
#define Def1(F,N,A,P)    <defsysp.c>
#define Def1Red(F,N,A,P)<defsysp.c>
#define Def2(F,N,A,P)    <defsysp.c>
#define Def2Red(F,N,A,P)<defsysp.c>
#define Defatom(T,N)     <defsysp.c>
#define Deftemp(F,N,A)   <defsysp.c>
#define ECLAUSE_TYPE     <include.h>
#define ESP_SIZE         <include.h>
#define ETERNAL          <include.h>
#define EXTRACT          <unify.c>
#define FALSE            <include.h>
#define FALSE_REGISTERED<include.h>
#define FILE_POINTER     <include.h>
#define FILE_TYPE        <include.h>
#define FINITEFUN        <include.h>
#define FLOAT_NUM        <include.h>
#define FLT_EPSILON      <include.h>
#define FROM_CONC        <syspred1.c>
#define FROM_CONC        <syspred2.c>
#define FROM_NAME        <syspred1.c>
#define FROM_NAME        <syspred2.c>
#define FULLSTOP         <include.h>
#define FX               <defsysp.c>
#define FY               <defsysp.c>
#define HASH_SIZE        <include.h>
#define HEAP_SIZE        <include.h>
#define INFIX            <include.h>
#define INPUT            <syspred1.c>
#define INT_NUM          <include.h>
#define Is_Leap          <include.h>
#define Is_Modular       <include.h>
#define Is_Msolvable     <include.h>
#define Is_Normaltrace   <include.h>
#define Is_Notrace       <include.h>
#define Is_Steptrace     <include.h>
#define Is_Trace         <include.h>
```

```
#define Is_ctnormal     <include.h>
#define Is_ctnotrace    <include.h>
#define Is_ctstep       <include.h>
#define KEYIN           <include.h>
#define LC              <print.c>
#define LC              <read.c>
#define LC              <varset.h>
#define LIST_TYPE       <include.h>
#define Leap_mode       <include.h>
int     Llevel(t,e,nv)  <syspred1.c>
void    LtoC(t,e,pos,flag)      <syspred1.c>
void    LtoP(t,e,tt,depth)      <syspred1.c>
#define         MAIN    <main.c>
#define MEDIUM          <include.h>
#define MEMORY_ALLOC(X,Y,F)     <include.h>
#define MODULAR_DEFINED         <include.h>
#define Modmax_def              <include.h>
#define Modular_mode            <include.h>
#define Msolvable_mode          <include.h>
#define N               <print.c>
#define N               <read.c>
#define N               <varset.h>
#define NAME            <include.h>
#define NAME_MAX        <include.h>
#define NAME_SIZE       <include.h>
#define NEW             <new.c>
#define NEWPRED         <include.h>
#define NL              <include.h>
#define NOEXTRACT       <unify.c>
#define NONFUNC         <include.h>
#define NON_UNFOLDABLE  <include.h>
#define NOREDUCED_CLAUSE        <mainsub.c>
#define NOT_CONSTANT_TERM       <include.h>
#define NUMBER          <include.h>
struct clause   *Nclause(head,body,flag)        <new.c>
struct cset     *Ncset(flag)    <tr_sub.c>
struct eclause  *Neclause(val,env,tail,flag)    <new.c>
struct pair     *Nenv(n)        <new.c>
struct node     *Newnode(goal,icons,env,nlink,nlast)            <refute.c>
struct term     *Nfile(x)       <new.c>
struct func     *Nfunc(ftype,n,a)       <new.c>
struct clause   *Nlist(head,body,flag)  <new.c>
struct term     *Nnum(nbuf,flag)        <new.c>
struct term     *Nnum_val(x,flag)       <new.c>
#define Normal          <jpsgsub.c>
#define Normaltrace_mode<include.h>
#define Notrace_mode    <include.h>
struct pst      *Npst(flag)     <new.c>
struct term     *Npst_item(p,pobj,next) <new.c>
#define Npstobj(Head,Env,Tail,Flag)     <defsysp.c>
```

```
#define Npstobj(Head,Env,Tail,Flag)     <include.h>
#define Npstobj(Head,Env,Tail,Flag)     <unify.c>
struct term     *Nstr(x,flag)           <new.c>
struct term     *Nterm(n,flag)          <new.c>
struct term     *Nvar(nbuf,flag)        <new.c>
#define OUTPUT          <syspred1.c>
void    PCat(t,e,f)     <jpsgsub.c>
#define POSTFIX         <include.h>
#define PRED_IN_LINE    <mainsub.c>
#define PREFIX          <include.h>
#define PST_ITEM_TYPE   <include.h>
#define PST_TYPE        <include.h>
void    P_csnumber(cs,mode)             <tr_sub.c>
void    P_dclause(cl,e) <print.c>
void    P_hclause(cl,e) <print.c>
void    P_hclause_sub(cl,e)             <print.c>
void    P_status()      <tr_sub.c>
void    P_var(vlist)    <print.c>
int     Panswer(root,vlist)             <refute.c>
void    Pbinding(vlist,env)             <refute.c>
void    Pcahc_core(c,cst,e)             <print.c>
void    Pclause(c,e)    <print.c>
void    Pclause_core(c,e)               <print.c>
void    Pcmp(cmp)       <trans.c>
void    Pcset_cstr(cs)  <tr_sub.c>
void    Pcset_def(cs)   <tr_sub.c>
void    Peclause(ec)    <print.c>
void    Peclause_core(ec,d)             <print.c>
void    Penergy(cl)     <tr_sub.c>
void    Pfunctor(t,e,d) <print.c>
void    Pgoal(n)        <print.c>
void    Ppst(t,e,d)     <print.c>
void    Ppst_content(ptt,d)             <print.c>
void    Ppst_content2(ptt,env,d)                <print.c>
#define Pred(T)         <include.h>
struct func     *Predicate(fname,arity)         <new.c>
#define Predname(T)     <include.h>
void    Psequence(t,e,d)<print.c>
void    Psubcat(t,e)    <jpsgsub.c>
void    Pterm(t,e)      <print.c>
void    Pterm_core(t,e,d)               <print.c>
struct clause   *PtoL(t)<syspred1.c>
void    Ptree(t,e)      <jpsgsub.c>
void    Pvar(t,n)       <print.c>
void    Pvariant(va)    <tr_split.c>
void    Pvpair(va)      <tr_split.c>
#define Q               <print.c>
#define Q               <read.c>
#define Q               <varset.h>
#define REDUCEDFUN      <include.h>
```

```
#define REDUCED_DEF     <include.h>
#define REFMAX          <include.h>
#define REGISTERED      <include.h>
#define REMOVED         <include.h>
struct term     *Rlist(flag)            <read.c>
struct term     *Rpst(flag)             <read.c>
struct term     *Rterm(n,flag)          <read.c>
struct term     *Rterm_half(n,flag,m)           <read.c>
struct term     *Rterm_leftover(n,m,flag,t)             <read.c>
int     Rtoken()        <read.c>
#define SAFE            <unify.c>
#define SG              <print.c>
#define SG              <read.c>
#define SG              <varset.h>
#define SHEAP_SIZE      <include.h>
#define SP              <print.c>
#define SP              <read.c>
#define SP              <varset.h>
#define SPECIFIED       <syspred1.c>
#define SPYFUN          <include.h>
#define STAY_IF         <include.h>
#define STAY_IF_FALSE_PRED              <include.h>
#define STAY_IF_TRUE_PRED               <include.h>
#define STB(F)          <include.h>
#define STE             <include.h>
#define STINGY          <include.h>
#define STRING          <include.h>
#define SUSPEND         <include.h>
#define SYSFAIL         <include.h>
#define SYSFUN          <include.h>
#define SYSNO           <include.h>
#define SYSPRED         <defsysp.c>
#define SYSTRUE         <include.h>
void    Showfunc(f)     <print.c>
void    Showhorn(c,cst,e)               <print.c>
void    Shownewfunc()   <print.c>
#define Steptrace_mode  <include.h>
#define TB              <include.h>
#define TE              <include.h>
#define TEMPFUN         <include.h>
#define TEMPORAL        <include.h>
#define TEMPORAL_DEFINED<include.h>
#define TNTB            <include.h>
#define TNTE            <include.h>
#define TREEMAX         <jpsgsub.c>
#define TRUE            <include.h>
#define TSTB            <include.h>
#define TSTE            <include.h>
#define TTB             <include.h>
#define TTE             <include.h>
```

```
#define TYPE1SYS          <include.h>
#define TYPE1SYS_REDUCED<include.h>
#define TYPE2SYS          <include.h>
#define TYPE2SYS_REDUCED<include.h>
#define Termalloc(a)      <new.c>
#define Termalloc(a)      <new.c>
void    Trace_Answer(root)      <refute.c>
void    Trace_False(n)  <refute.c>
void    Trace_False2(n) <refute.c>
int     Trace_Goal(n)   <refute.c>
void    Trace_True(n)   <refute.c>
void    Trace_True2(n)  <refute.c>
void    Trace_Unification(n,s)          <refute.c>
#define UC                <print.c>
#define UC                <read.c>
#define UC                <varset.h>
#define UL                <print.c>
#define UL                <read.c>
#define UL                <varset.h>
#define UNIT_DEFINED      <include.h>
#define UNSAFE            <unify.c>
#define UNTOUCHED         <include.h>
#define UP                <include.h>
#define USERFUN           <include.h>
#define USTACK_SIZE       <include.h>
#define VACUITY_NOCHECK <include.h>
#define VARNAME           <include.h>
#define VAR_GLOBAL_TYPE <include.h>
#define VAR_PST_TYPE      <include.h>
#define VAR_VOID_TYPE     <include.h>
#define VERSION           <main.c>
#define VMAX              <include.h>
#define XF                <defsysp.c>
#define XFX               <defsysp.c>
#define XFY               <defsysp.c>
#define YF                <defsysp.c>
#define YFX               <defsysp.c>
void    abandon_transformation()        <trans.c>
int     abolish_pred(t,e)               <syspred1.c>
int     abomb_pred(t,e)         <defsysp.c>
void    add_clause(c,vlist,anum)        <tr_sub.c>
void    add_component_pst(f,a,ec)       <mainsub.c>
void    add_cs_to_set(cs,flag)          <tr_sub.c>
void    add_label(f,a,l,flag)           <mainsub.c>
void    add_set(s,flag) <new.c>
void    add_to_set()    <trans.c>
void    addpst(t,e)     <print.c>
void    adv()           <read.c>
#define advance         <include.h>
int     alldigit(c)     <read.c>
```

```
int     *alloc(n)       <new.c>
void    allspy(n)       <mainsub.c>
#define alpha           <read.c>
#define alphabet(CH)    <print.c>
int     app_str(x,y,z,ez)               <syspred2.c>
int     apply(target,head,rest,anum)            <trans.c>
int     apply_add_clause(head,e0,ec)            <trans.c>
int     arg_pred(t,e)   <syspred1.c>
int     arg_type(a)     <tr_sub.c>
int     assert_pred(t,e)        <syspred1.c>
int     assertz_pred(t,e)       <syspred1.c>
#define atomic_equal(u,t)       <include.h>
void    attach(c,vl,anum)       <tr_split.c>
void    attach_arg(arg,c,vl)    <tr_split.c>
int     attach_pred(t,e,n,m,status)             <syspred1.c>
struct node     *backtrack_node(n)              <refute.c>
#define bracket(C)              <read.c>
int     calc_1(x,y,z,e,op)      <syspred2.c>
int     calc_2(x,z,y,e,op)      <syspred2.c>
void    calc_all_var(f)         <mainsub.c>
void    calc_component()        <mainsub.c>
int     calc_pred(t,e,op)       <syspred2.c>
int     *challoc(n)             <new.c>
int     check(c)                <read.c>
int     check_INITDEF()         <trans.c>
void    check_all_unit(fl)      <mainsub.c>
void    check_constant_term(t)  <modular.c>
int     check_modularity(cst)   <mainsub.c>
void    check_recursion()       <mainsub.c>
void    check_unitpred(f)       <mainsub.c>
int     clause_pred(t,e,n,status)               <syspred1.c>
void    clear_predicate(f)      <syspred1.c>
void    clear_up_DEF()          <trans.c>
int     close_pred(t,e)         <syspred1.c>
int     cmp_clause(a1,a2)       <tr_sub.c>
int     cmp_cplxt(a1,a2)        <tr_sub.c>
int     cmp_flt(a1,a2)          <tr_sub.c>
int     cmp_fp(a1,a2)           <tr_sub.c>
int     cmp_int(a1,a2)          <tr_sub.c>
int     cmp_label(l1,l2)        <mainsub.c>
int     cmp_list(a1,a2)         <tr_sub.c>
int     cmp_pst(a1,a2)          <tr_sub.c>
int     cmp_str(a1,a2)          <tr_sub.c>
int     cmp_var(a1,a2)          <tr_sub.c>
int     cname_pred(t,e,nn)      <jpsgsub.c>
#define cnew(s)         <include.h>
struct term     *cnlistmake(n)  <jpsgsub.c>
int     compare_pred(t,e)       <syspred2.c>
#define component_checked(F)    <include.h>
#define component_not_checked(F)        <include.h>
```

```
struct clause   *compress_clause(cst)   <mainsub.c>
int     concat2_pred(t,e)               <syspred2.c>
int     concat_pred(t,e,n,status)       <syspred2.c>
struct clause   *convert_list_to_clause(t,e,tt,ee,p,emsg)       <syspred1.c>
struct term     *copy_arg(t,i,flag)     <tr_split.c>
struct clause   *copy_clause(cl,flag)   <tr_split.c>
struct term     *copy_pst(pt,flag)      <tr_split.c>
struct term     *copy_term(t,flag)      <tr_split.c>
int     count_pred(t,e)         <syspred2.c>
int     cs_status_type(st)      <tr_sub.c>
int     cu(t,e)         <modular.c>
int     cunify_pred(t,e)        <syspred1.c>
int     cut_pred(t,e,n)         <defsysp.c>
int     decode_pname(fname)     <mainsub.c>
void    decrement_vacuous(t)    <new.c>
int     default_pred(t,e)       <defsysp.c>
void    defclause()             <main.c>
void    defnewfunc()            <main.c>
void    defsyspred()            <defsysp.c>
void    delete_constraint(vl)   <tr_split.c>
void    delete_tmp()            <mainsub.c>
#define delimitchar(C)          <read.c>
int     diff_str(x,z,y,e,first) <syspred2.c>
void    disp_func_def(f_from,f_to)              <mainsub.c>
void    divide_consts(cl)       <tr_split.c>
int     divstr_pred(t,e)        <syspred2.c>
#define down(p,t,e)             <include.h>
struct pair     *ealloc(n)      <new.c>
struct eclause  *eclause_append(head,tail)      <modular.c>
struct eclause  *eclause_conc(ec1,ec2)          <tr_sub.c>
void    edit_predicate()        <main.c>
void    end_unfoldfold()        <trans.c>
int     energy(tm)              <tr_sub.c>
struct pair     *env(t,e)       <unify.c>
int     eq_pred(t,e)            <syspred1.c>
int     eq_pred_sub(x,y,ex,ey)  <syspred1.c>
int     equal_pred(t,e)         <syspred1.c>
int     equalpred(t1,e1,t2,e2)  <syspred1.c>
void    error(s)                <main.c>
void    error_detail(t,e,s)     <main.c>
struct func     *exist_fname(fname)             <new.c>
struct term     *exist_termpair(t)              <tr_split.c>
struct term     *exist_vpair(t) <tr_split.c>
struct node     *extend(n,status)       <refute.c>
int     extend_apply(target,head,rest,e0,f,s)   <trans.c>
int     fail_pred(t,e)          <defsysp.c>
int     file_open_pred(t,e,openmode)    <syspred1.c>
#define filep_value(Term)               <include.h>
void    filewrite(n)            <mainsub.c>
struct pst_item *find_pstitem(t,e)      <new.c>
```

```
#define finitefun(F)                <include.h>
int     firsthalf(h,w)              <syspred2.c>
int     foldunfold()                <trans.c>
void    freeheap()                  <mainsub.c>
struct cset     *from_to(s1,s2)     <trans.c>
#define funcalloc(a)                <new.c>
#define funcalloc(a)                <new.c>
struct func     *funcsearch(fname,arity)            <new.c>
int     functor_pred(t,e)           <syspred1.c>
void    garbagecollect()            <main.c>
void    general_assert(t,e,flag)<syspred1.c>
int     gensym_pred(t,e)            <syspred2.c>
int     geq_pred(t,e)               <syspred2.c>
int     greater_arg(a1,a2)          <tr_sub.c>
int     greater_pred(t,e)           <syspred2.c>
int     greater_term(t1,t2)         <tr_sub.c>
int     halt_pred(t,e)              <defsysp.c>
int     has_common_label(ec,cm)     <mainsub.c>
int     has_no_pst(t)               <tr_split.c>
int     has_no_var(t)               <tr_sub.c>
int     hash(fname)                 <new.c>
int     have_nextgoal(n)            <refute.c>
#define head_of_list(Term)          <include.h>
void    helpmenu()                  <mainsub.c>
#define in_cheap(X)                 <modular.c>
#define in_sheap(X)                 <include.h>
#define in_upper_heap(X,Y)          <modular.c>
void    index_func(fnew)            <new.c>
void    index_funclist(flist)       <new.c>
struct itrace   *index_newflist(fl,it)          <new.c>
void    index_op(f,type,prec)       <defsysp.c>
void    index_set(chead,con,flag)               <new.c>
void    init_atoms()                <defsysp.c>
void    init_category()             <jpsgsub.c>
void    init_froles()               <defsysp.c>
void    init_heap_max()             <mainsub.c>
void    init_operator()             <defsysp.c>
void    init_pp()                   <print.c>
struct set      *init_set(n)        <refute.c>
void    init_status()               <main.c>
void    init_syspred()              <defsysp.c>
void    init_system_component(f,a)      <defsysp.c>
void    init_unfoldfold()               <trans.c>
struct clause   *insert_clause(ct,cl)   <tr_sub.c>
void    insert_cs(cs,newcs)             <trans.c>
struct eclause  *insert_pstobj(val,tail,flag)           <read.c>
#define is_atomic(Term)             <include.h>
int     is_body_finite(f)           <mainsub.c>
#define is_clause(Term)             <include.h>
#define is_component_checked(F) <include.h>
```

```
#define is_component_not_checked(F)     <include.h>
#define is_dead(n)                      <refute.c>
#define is_eclause(Term)                <include.h>
#define is_file(Term)                   <include.h>
#define is_funcsys(F)                   <include.h>
#define is_functor(Term)                <include.h>
#define is_int(Term)                    <include.h>
#define is_list(Term)                   <include.h>
#define is_lower(CH)                    <print.c>
int     is_modular_clause(cl)           <tr_sub.c>
int     is_modular_head(t)              <trans.c>
int     is_modular_literal(t)           <tr_sub.c>
#define is_nofuncsys(F)                 <include.h>
#define is_num(Term)                    <include.h>
#define is_pst(Term)                    <include.h>
#define is_pstitem(Term)                <include.h>
#define is_readable(FP)                 <include.h>
#define is_root(n)                      <refute.c>
#define is_string(Term)                 <include.h>
int     is_term_end(c)                  <read.c>
#define is_tip(n)                       <refute.c>
#define is_unitclause(Set)              <include.h>
#define is_varname(X)                   <read.c>
#define is_voidvar(t)                   <include.h>
#define is_writable(FP)                 <include.h>
#define isallunit(F)                    <include.h>
#define isatom(Term)                    <include.h>
#define isconst(Term)                   <include.h>
#define isconst_functor(Term)           <include.h>
#define isconst_list(L)                 <read.c>
#define isdigit(X)                      <read.c>
#define isfinite(F)                     <include.h>
#define isfunc(F)                       <include.h>
#define isnewpred(F)                    <include.h>
#define isnonfunc(F)                    <include.h>
#define isnoreduced(F)                  <include.h>
#define isnospy(F)                      <include.h>
#define isnotnewpred(F)                 <include.h>
int     isop_pred(t,e,n,status)         <defsysp.c>
#define isrecursive(F)                  <include.h>
#define isreduced(F)                    <include.h>
#define isspy(F)                        <include.h>
#define issystem(F)                     <include.h>
#define isuser(F)                       <include.h>
#define isvar(t)                        <include.h>
#define isxdigit(X)                     <read.c>
#define kanzi(CH)                       <print.c>
#define kanzi(CH)                       <read.c>
int     keyread(a)                      <read.c>
int     leq_pred(t,e)                   <syspred2.c>
```

```
int     pcon_pred(t,e,n)          <syspred1.c>
int     pickname(t,e)             <jpsgsub.c>
int     pnames_pred(t,e)          <defsysp.c>
int     pp_number(ec)             <print.c>
int     pred_compare(f1,f2)       <new.c>
int     prefix_is_atom(m)         <read.c>
void    prepare()                 <main.c>
void    preprocess_all_unit(fl,flag)          <mainsub.c>
void    preprocess_constr_sub(flag)           <mainsub.c>
void    preprocess_constraints(fn)            <mainsub.c>
void    preprocess_unit(f,flag) <mainsub.c>
void    print_ancestors(n)        <refute.c>
void    print_constant()          <main.c>
void    print_hash_table()        <new.c>
void    print_pp(d)               <print.c>
void    printtime()               <mainsub.c>
void    printtime()               <mainsub.c>
void    printtime()               <mainsub.c>
struct node     *proceed_node(n,btnode)       <refute.c>
void    prolog_execution()        <main.c>
void    pst_add_unify(t,e,u,f)  <defsysp.c>
void    pst_add_unify_sub(entry,ol,e)         <defsysp.c>
void    pst_unify(t,e,u,f,safeflag)           <unify.c>
void    push_log(oldt,oldenv,newt)            <modular.c>
void    push_pstlog(oldt,oldenv,newt)         <modular.c>
void    putcursor()               <mainsub.c>
int     pvalue_pred(t,e)          <defsysp.c>
void    quit_prolog()             <mainsub.c>
void    quit_transformation()   <trans.c>
int     quote_needed(f)           <print.c>
#define quotesign                 <read.c>
void    read_comments()           <read.c>
void    read_digits(i)            <read.c>
void    read_hexa(i)              <read.c>
int     read_pred(t,e)            <syspred1.c>
void    read_spechar(i)           <read.c>
void    readfile()                <mainsub.c>
#define readword(S)               <include.h>
void    rec_to_finite()           <mainsub.c>
void    recalc_component()        <mainsub.c>
void    recalc_voccur_sub(t,v)  <new.c>
void    recalc_voccurrence(cl,v)        <new.c>
struct eclause  *record_pstlists(ptt,e) <unify.c>
struct pst_item *record_pstobjects(t,e) <unify.c>
#define recursivefun(F)           <include.h>
struct eclause  *reduce_clause(cl,e)    <tr_sub.c>
struct eclause  *reduce_clause_m(cl,e)  <tr_sub.c>
struct clause   *reduce_cstr(cst,vlist,anum,env)      <mainsub.c>
struct clause   *reduce_substitute(cst,e)             <mainsub.c>
#define reducedfun(F)             <include.h>
```

```
int     refute(Root,n,Status)   <refute.c>
void    remove_from_CSTR(f)     <trans.c>
struct clause   *remove_modular_literals(cl)          <tr_split.c>
struct pst_item *remove_pstitem(t,e)    <unify.c>
void    rename_var_names(v)     <main.c>
void    renum_pvars(pvs,vnum)   <main.c>
struct clause   *reorder(cl,tc) <trans.c>
void    reorder_clause(cl,tc)   <tr_sub.c>
void    replace_terms(c1,c2,vl) <tr_split.c>
void    reset_component()       <mainsub.c>
int     reset_timer_pred(t,e)   <defsysp.c>
void    reset_voccurrence(v)    <new.c>
int     resolve(n0,n,sliteral,env)      <refute.c>
int     retract_pred(t,e,n,status)      <syspred1.c>
void    safe_unify(t,e,u,f,extflag)     <unify.c>
int     *salloc(n)              <new.c>
int     satisfiable(cl,anum)    <tr_sub.c>
void    scanpst_clause(t,e)     <print.c>
void    scanpst_eclause(ec)     <print.c>
void    scanpst_functor(t,e)    <print.c>
void    scanpst_term(t,e)       <print.c>
struct term     *search_log(t,e)        <modular.c>
struct term     *search_pstlog(t,e)     <modular.c>
int     see_pred(t,e)           <syspred1.c>
int     seen_pred(t,e)          <syspred1.c>
void    set_body_component(ff)  <mainsub.c>
void    set_category()          <jpsgsub.c>
void    set_eof()               <mainsub.c>
void    set_head_component(f)   <mainsub.c>
void    set_inputfile(n)        <mainsub.c>
void    set_new_def(c,vl,anum)  <trans.c>
void    set_temporal_def(f)     <trans.c>
struct set      *setconcat(slist,s)             <new.c>
void    settimer()              <mainsub.c>
void    settimer()              <mainsub.c>
void    settimer()              <mainsub.c>
void    show_category()         <jpsgsub.c>
void    show_heap_max()         <mainsub.c>
void    show_newdefs()          <tr_sub.c>
void    show_pred_def(f)        <mainsub.c>
void    show_pred_roles(f)      <mainsub.c>
void    show_syspred_name()     <mainsub.c>
void    show_syspred_status(f)  <mainsub.c>
void    show_userpred_name()    <mainsub.c>
void    showdef(fname)          <mainsub.c>
void    showvar(v)              <print.c>
int     skip(c)                 <read.c>
void    skip_cr()               <tr_sub.c>
#define skipline                <include.h>
#define snew(s)                 <include.h>
```

```
struct clause   *sort_clause(cl)<tr_sub.c>
#define specialchar(C)          <read.c>
struct compartment      *split(clist,vlist,anum)        <tr_split.c>
#define spychange(F)            <include.h>
#define spyfun(F)               <include.h>
void    spyswitch(fname)        <mainsub.c>
struct clause   *startmodular(clist,vlist,anum)         <trans.c>
int     stay_pred(t,e)          <defsysp.c>
int     step_asking()           <tr_sub.c>
void    stepswitch()            <mainsub.c>
void    store_termpair(told,tnew)       <tr_split.c>
void    store_vpair(told,tnew)  <tr_split.c>
#define str_value(Term)         <include.h>
int     strcmp_pred(t,e)        <syspred2.c>
#define streq(p,q)              <include.h>
int     strlen_pred(t,e)        <syspred2.c>
int     substr_pred(t,e)        <syspred2.c>
int     subsume(t,e,u,f,flag)   <defsysp.c>
int     subsume_pst(t,e,u,f,flag)       <defsysp.c>
int     subsume_pstlist(x,y,e,flag)     <defsysp.c>
int     sum_pred(t,e)           <syspred2.c>
struct clause   *surface_copy_clause(cl,flag)           <tr_sub.c>
int     system_function(t,e,n)          <defsysp.c>
int     system_pred(t,e,n,m,status)     <defsysp.c>
void    systemcommand(c)                <main.c>
int     tab_pred(t,e)                   <syspred1.c>
#define tail_of_list(Term)              <include.h>
struct clause   *target_literal(cl)     <tr_sub.c>
struct clause   *target_literal(cl)     <tr_sub.c>
int     tell_pred(t,e)          <syspred1.c>
#define tempterm(a)             <new.c>
#define tempterm(a)             <new.c>
char    *termname(t,e)          <jpsgsub.c>
int     termnumber(t)           <modular.c>
struct term     *termset(t,e,flag)              <modular.c>
struct eclause  *termset_pstobj(pobj,flag)      <modular.c>
struct eclause  *termset_pstobj_sub(pobj,e,flag)<modular.c>
int     timer_pred(t,e)         <defsysp.c>
int     told_pred(t,e)          <syspred1.c>
struct term     *tolist(c,flag) <modular.c>
#define tprint0(X)              <include.h>
#define tprint1(X,V)            <include.h>
#define tprint2(X,V1,V2)        <include.h>
#define tprint3(X,V1,V2,V3)     <include.h>
#define tprint4(X,V1,V2,V3,V4)  <include.h>
#define tputc(X)                <include.h>
void    traceswitch()           <mainsub.c>
void    trans_routine()         <main.c>
struct eclause  *transform(precond,newc,newenv)         <modular.c>
int     tree_pred(t,e)          <jpsgsub.c>
```

```
void    treeprint(t,e,n)          <jpsgsub.c>
int     true_pred(t,e,n,status) <defsysp.c>
void    truncate_varname(n,nbuf)<main.c>
struct term     *try_fold(c,n)  <modular.c>
int     tunify(t,e,u,f,flag)    <unify.c>
int     type_pred(t,e)          <defsysp.c>
int     unbreak_pred(t,e)       <defsysp.c>
void    undo(u)                 <new.c>
void    unfold_cstr(cs)         <trans.c>
void    unfold_derivation(cs)   <trans.c>
void    unify(t,e,u,f)          <unify.c>
void    unify_merge_psts(target,object,safeflag)        <unify.c>
void    unify_pst_extract(t,e,u,f)                      <unify.c>
void    unify_pstlist_objects(entry,ol,e,safeflag)      <unify.c>
struct term     *up_atomic(t,flag)              <modular.c>
struct term     *up_const(t,flag)               <modular.c>
struct term     *up_const_functor(t,flag)       <modular.c>
struct clause   *up_eclause(ec,flag)            <modular.c>
void    up_init()               <modular.c>
struct clause   *up_itrace_clause(cl,anum)      <modular.c>
struct term     *up_pst(pt,e,flag)              <modular.c>
void    up_restore()            <modular.c>
void    upush(p)                <new.c>
#define userfun(F)              <include.h>
int     var_pred(t,e)           <syspred1.c>
struct term     *var_trans(v,flag)              <tr_split.c>
struct variant  *variant(cl,flag)               <tr_split.c>
struct variant  *variant_v(cl,flag)             <tr_split.c>
struct term     *varsearch(varname)             <new.c>
#define vcomponent(t)           <include.h>
#define vconstraint(t)          <include.h>
#define vdecrement(t)           <include.h>
#define vheadoccurrence(t)      <include.h>
#define vincrement(t)           <include.h>
#define vlink(t)                <include.h>
#define vname(t)                <include.h>
#define vnumber(t)              <include.h>
#define voccurrence(t)          <include.h>
int     vpair_length(vp)        <tr_split.c>
#define white                   <read.c>
int     write_pred(t,e)         <syspred1.c>
void    writenewfunc()          <print.c>
```

<< cu-Prolog第三版 関数クロスリファレンス >>

92.1.24

```
function_name+          一回しか現れない関数
function_name.          他の関数を呼び出していない関数
function_name~          二度目以上の表示
[function_name]         主ルーチン

[main]---------------------------------- メインループ
.   [prepare]           システム初期設定
.   prolog_execution    prologトップレベルループ
.   .   [systemcommand] システムコマンド(%)実行
.   .   [check_recursion]       プログラムstaticアナライザ
.   .   defnewfunc+             新述語導入節($で始まる)読み込み&セット
.   .   .   rename_var_names
.   .   .   .   truncate_varname+
.   .   .   literalnumber.
.   .   defclause+      プログラム節読み込み&セット
.   .   .   rename_var_names~
.   .   printtime       CPUタイム表示
.   .   oscommand+      UNIX(OS)のコマンド実行
.   .   putcursor+      カーソル表示
.   .   questionclause+ 質問節読み込み&セット&実行
.   .   .   [refute]    推論処理エントリ
.   .   .   Panswer+    答の表示
.   .   .   .   Pbinding        変数の束縛表示
.   .   .   .   Trace_Answer
.   .   .   no+
.   .   readfile+       ファイル読み込み(fpセット)
.   .   set_eof         ファイル末尾
.   .   .   clearerr.
.   .   trans_routine+  制約解消系(@コマンド)
.   .   .   modular+    @コマンドエントリ
.   .   .   .   [startmodular]  制約解消系メイン
.   .   .   .   nil+
.   .   .   .   show_newdefs+   新述語表示
.   on_interrupt+
.   settimer    タイマセット
.   .   times.
.   signal+

[refute]---------------------------------- 推論処理メインルーチン
.   backtrack_node      バックトラック先のノード
.   .   Trace_False2
.   Trace_False         ゴール失敗時のトレース
.   Trace_Goal          ゴールトレース表示
.   .   print_ancestors+
.   Trace_True          ゴール成功時のトレース
.   is_dead.            それ以上ORのないリーフ
```

```
.   extend                展開
.   .   [system_function]       組み込み述語処理
.   .   init_set
.   .   resolve+          ヘッドunification&ゴール置換
.   .   .   [tunify]      ユニファイア
.   .   .   Trace_Unification
.   is_tip.               ゴールが空のノード
.   proceed_node          次のandゴールの存在するノード
.   .   Trace_True2.
.   .   next_goal+
.   .   have_nextgoal+

[prepare]---------------------------------- システム初期化
.   init_status           大域変数初期化
.   .   init_syspred+     組み込み述語・アトム他エントリ初期化
.   .   .   defsyspred+ 組み込み述語初期化
.   .   .   .   Def1
.   .   .   .   Def1Red
.   .   .   .   Def2+
.   .   .   .   Def2Red+
.   .   .   .   Deftemp+
.   .   .   init_atoms+ 組み込みアトム初期化
.   .   .   .   Def1~
.   .   .   .   Defatom
.   .   .   init_froles+        組み込み述語モード設定
.   .   .   .   init_system_component+
.   .   .   init_operator+      オペレータ初期化
.   .   .   .   Def1~
.   .   .   .   Def1Red~
.   .   .   .   Defatom~
.   .   init_heap_max+  ヒープポインタ初期化
.   isatty+
.   open_title+           オープニングタイトル

[systemcommand]---------------------------------- +組み込み述語処理
.   [prepare]   システム初期化
.   quit_prolog cu-Prolog終了
.   .   delete_tmp        一時ファイル消去
.   .   keyread
.   readword+
.   set_category+         JPSG用カテゴリファンクタ設定
.   .   init_category.
.   .   list_to_cat+
.   .   number+
.   edit_predicate+       プログラムエディット(子プロセスでエディタ使用)
.   .   set_inputfile
.   .   system.
.   filewrite   プログラムをファイルに書き込む(%w)
.   .   writenewfunc+
.   freeheap    ヒープ残領域表示 (%f)
```

```
.   garbagecollect        静的GC
.   .   set_inputfile~
.   .   delete_tmp~
.   print_constant+       システムアトム表示
.   Shownewfunc+          新述語導入節表示
.   helpmenu+             ヘルプ(%h)
.   .   show_category+
.   loghandle+            ログファイル設定&解除
.   preprocess_constraints+      制約の前処理(%P)
.   .   preprocess_unit
.   .   .   [termset]     項のコピー
.   .   .   check_modularity+
.   .   .   .   is_modular_literal
.   .   .   reduce_cstr+
.   .   .   .   [startmodular]
.   .   .   .   [termset]
.   .   .   .   reduce_substitute+
.   .   .   .   .   [tunify]
.   .   .   .   .   compress_clause+
.   .   preprocess_constr_sub+
.   .   .   preprocess_all_unit+
.   print_hash_table+     ハッシュテーブル表示
.   show_heap_max         ヒープ表示(デバッグ用)
.   showdef+     プログラム定義表示(%d)
.   .   [check_recursion]
.   .   show_pred_def+    ユーザ述語表示
.   .   .   isnewpred.
.   .   .   show_pred_roles
.   .   .   show_syspred_status+
.   .   show_syspred_name+       組み込み述語表示
.   .   show_userpred_name+
.   .   .   isnewpred~
.   spyswitch+   スパイフラグ設定(%s)
.   .   isspy.
.   .   allspy+
.   .   .   nospyfun+
.   .   .   spyfun+
.   .   spychange+
.   stepswitch+           ステップトレースモード設定
.   traceswitch+          ノーマルトレースモード設定

[check_recursion]------------------------- プログラムの静的解析
.   calc_component+       成分の計算
.   .   calc_all_var
.   .   .   recalc_voccurrence
.   .   .   .   decrement_vacuous+
.   .   .   .   .   vdecrement+
.   .   .   .   recalc_voccur_sub+
.   .   .   .   reset_voccurrence+
.   .   component_checked.
```

```
.   .   set_body_component
.   .   .   merge_component
.   .   .   .   cmp_label.
.   .   set_head_component
.   .   .   add_component_pst
.   .   .   add_label
.   .   .   .   cmp_label~
.   .   .   merge_component~
.   check_all_unit
.   .   check_unitpred
.   .   .   recursivefun+
.   rec_to_finite
.   .   finitefun.
.   .   is_body_finite
.   .   isfinite+
.   reset_component+
.   .   vlink
.   .   .   int+
.   .   vcomponent.

[tunify]---------------------------------- 単一化
.   safe_unify  safe_unificationメインルーチン
.   .   atomic_equal
.   .   .   fabs+
.   .   ocheck+
.   .   pst_unify       PSTの単一化ルーチン
.   .   .   record_pstobjects
.   .   .   .   record_pstlists
.   .   .   .   Npst_item+
.   .   .   .   .   cnew
.   .   .   .   .   .   challoc
.   .   .   remove_pstitem
.   .   .   unify_merge_psts+
.   .   .   unify_pstlist_objects+
.   .   .   .   record_pstlists~
.   .   unify_pst_extract
.   .   .   record_pstobjects~
.   unify       unsafe unificationルーチン
.   .   atomic_equal~
.   .   pst_unify~
.   .   unify_pst_extract~

[startmodular]---------------------------------- 制約解消系エントリ
.   abandon_transformation      制約解消失敗(全新述語消去)
.   .   index_newflist
.   .   .   in_sheap.
.   .   .   up_itrace_clause+
.   .   .   .   [variant]
.   .   reducedfun+
.   add_to_set+         新述語をシステムに登録
```

```
. . index_newflist~
. . add_cs_to_set
. . . [variant]
. . . add_set
. . . . setconcat+
. clear_up_DEF 不要な定義節消去
. . remove_from_CSTR
. end_unfoldfold+ 制約解消系後処理
. . recalc_component+
. . . is_component_not_checked+
. foldunfold+ fold/unfold変換エントリ
. . P_status+
. . . P_csnumber+
. . . Pcset_cstr+
. . . . cs_status_type.
. . . Pcset_def+
. . . . tprint3+
. . . . cs_status_type~
. . set_temporal_def
. . step_asking+ 制約変換ステップトレース処理
. . . [apply]
. . . nth_cset+
. . . nth_literal+
. . . quit_transformation+
. . . reorder_clause+
. . . skip_cr+
. . check_INITDEF+
. . is_modular_head+
. . . vheadoccurrence.
. . unfold_cstr+ 非モジュラー節展開
. . . [apply]
. . . [target_literal]
. . . from_to+
. . . insert_cs+
. . . reorder.
. . unfold_derivation+ 新述語導入節展開
. . . [apply]
. . . [target_literal]
. . . literalnumber~
. . . reorder~
. modular_form 制約のモジュラー形
. . satisfiable 制約のsatisfiabilityチェック
. . . [refute]
. . split+ 変数の共有関係で制約を分割
. . . vconstraint.
. . . attach+
. . . . is_modular_literal~
. . . . attach_arg+
. . . . . novar
. . . . . replace_terms+
```

```
. . . . has_no_var+
. . . . . novar~
. . . divide_consts
. . . . has_no_pst+
. . . delete_constraint+
. . . remove_modular_literals+
. . . . is_modular_literal~
. . surface_copy_clause+
. . Pcmp+
. . new_constraint+ 新述語による制約
. . . try_fold+ fold変換
. . . . match+
. . . . . match_term+
. . . . termnumber
. . . variant_v+
. . . . Pvariant+
. . . new_pred_set+
. . . . newpred+
. . . . set_new_def+
. . . . vpair_length+
. . . variables+
. . set_const_pst+
. init_unfoldfold+ 制約解消系前処理

[apply]---------------------------- 制約の展開
. [system_function] 組み込み述語処理
. Newnode
. . init_set~
. system_pred
. isfunc.
. reduce_clause 節の簡約(定義が一つの述語を展開する)
. . reduce_clause_m+
. . . [tunify]
. . . [system_function]
. . . Neclause
. . . one_def_literal+ 定義が一つの述語かチェック
. apply_add_clause 節のコピー
. . [termset]
. . up_eclause
. . . [termset]
. . satisfiable~
. . add_clause+
. . . recalc_voccurrence~
. . . Ncset
. extend_apply+ 展開
. . [tunify]
. . eclause_conc.

[target_literal]---------------------------- 展開するリテラルの選択
. Penergy+ 活性度表示
```

```
.   energy      活性度の計算
.   .   isconst
.   .   voccurrence.
.   .   isallunit+
.   greater_term        項の比較
.   .   greater_arg
.   .   .   cmp_clause+
.   .   .   cmp_cplxt+
.   .   .   cmp_flt+
.   .   .   cmp_fp+
.   .   .   cmp_int+
.   .   .   cmp_list+
.   .   .   cmp_var
.   .   .   cmp_pst+
.   .   .   cmp_str+
.   .   .   arg_type+

[system_function]--------------------------- 組み込み述語処理
.   equalpred
.   .   [tunify]
.   stay_pred+
.   attach_pred+
.   .   transform
.   .   .   [startmodular]
.   .   .   eclause_append+
.   .   convert_list_to_clause
.   strcmp_pred+
.   substr_pred+
.   divstr_pred+
.   .   Nnum_val
.   .   firsthalf+
.   compare_pred+
.   timer_pred+
.   .   Nnum_val~
.   .   times~
.   reset_timer_pred+
.   .   times~
.   default_pred+
.   .   pst_add_unify+
.   .   .   pst_add_unify_sub+
.   .   .   .   record_pstlists~
.   .   .   record_pstobjects~
.   .   .   remove_pstitem~
.   .   subsume
.   .   .   subsume_pst+
.   .   .   .   subsume_pstlist+
.   abomb_pred+
.   halt_pred+
.   isop_pred+
.   .   [tunify]
```

```
. . Nnum_val~
. not_pred+
. . [refute]
. op_pred+
. . index_op
. pnames_pred+
. pvalue_pred+
. type_pred+
. unbreak_pred+
. cname_pred+
. . cnlistmake+
. . pickname+
. . . termname+
. tree_pred+
. . Ptree+
. . . treeprint+
. . . . PCat
. . . . . Psubcat+
. . . . . null_or_nil+
. . . . oldlink+
. abolish_pred+
. . clear_predicate+
. arg_pred+
. assert_pred+
. . general_assert
. . . [termset]
. . . list_to_clause+
. . . . [termset]
. assertz_pred+
. . general_assert~
. clause_pred+
. . [tunify]
. . tolist
. close_pred+
. . filep_value.
. cunify_pred+
. . cu+
. . . [startmodular]
. . . [termset]
. . . vnumber.
. eq_pred+
. . eq_pred_sub+
. equal_pred+
. functor_pred+
. . make_func+
. . match_func+
. . . [tunify]
. makelist_pred+
. . Llevel+
. . LtoP+
```

```
.   .   PtoL+
.   memb_pred+
.   .   [tunify]
.   name_pred+
.   .   alldigit+
.   .   .   isdigit
.   .   .   numeric+
.   .   CtoL
.   .   LtoC
.   nl_pred+
.   .   filep_value~
.   open_pred+
.   .   file_open_pred
.   or_pred+
.   .   init_set~
.   .   convert_list_to_clause~
.   pcon_pred+
.   read_pred+
.   .   check
.   .   filep_value~
.   .   is_readable+
.   retract_pred+
.   .   [tunify]
.   .   tolist~
.   .   is_unitclause.
.   see_pred+
.   .   file_open_pred~
.   seen_pred+
.   tab_pred+
.   .   filep_value~
.   tell_pred+
.   .   file_open_pred~
.   told_pred+
.   var_pred+
.   write_pred+
.   .   filep_value~
.   concat2_pred+
.   .   CtoL~
.   .   LtoC~
.   concat_pred+
.   .   app_str+
.   .   diff_str+
.   count_pred+
.   .   Nnum_val~
.   gensym_pred+
.   geq_pred+
.   .   numcomp_pred
.   greater_pred+
.   .   numcomp_pred~
.   leq_pred+
```

```
.   .   numcomp_pred~
.   less_pred+
.   .   numcomp_pred~
.   multiply_pred+
.   .   calc_pred
.   .   .   calc_1+
.   .   .   calc_2+
.   strlen_pred+
.   .   Nnum_val~
.   .   substring+
.   sum_pred+
.   .   calc_pred~

[termset]---------------------- 項のコピー(環境付き)
.   isconst_functor.    変数を含まない複合項
.   check_constant_term+        項が変数を含むかチェックする
.   .   isconst~
.   push_log+
.   search_log+
.   up_atomic
.   .   in_upper_heap
.   .   .   in_cheap+
.   .   .   in_sheap~
.   up_const_functor
.   .   in_upper_heap~
.   .   up_const+
.   up_pst+
.   .   push_pstlog+
.   .   termset_pstobj+
.   .   .   [termset]
.   .   .   Npstobj
.   .   termset_pstobj_sub+
.   .   .   [termset]
.   .   .   Npstobj~
.   .   search_pstlog+

[variant]---------------------------------- 項のコピー(環境なし)
.   copy_clause
.   .   copy_term
.   .   .   Npstobj~
.   .   .   in_sheap~
.   .   .   copy_arg+
.   .   .   .   has_common_label
.   .   .   .   store_vpair
.   .   .   exist_termpair.
.   .   .   exist_vpair.
.   .   .   copy_pst+
.   .   .   .   Npst
.   .   .   .   store_vpair~
.   .   .   .   copy_eclause+
```

```
.   .   .   store_termpair
.   .   .   var_trans+
.   .   .   .   store_vpair~
```

==========subroutines==========

```
<up_init>.          termsetの初期設定
<up_restore>        termsetの後処理
<Nvar>              var構造体セット
<nalloc>            name string heap(nheap□)へのalloc
.   alloc
.   sizeof.
<Nstr>              文字列セット
<Nnum>              数値セット
.   atof+
<is_modular_clause>     節のモジュラー性判定
<index_set>         CAHCセット
<index_func>        述語をハッシュテーブルに登録
.   pred_compare+
<funcsearch>        ハッシュテーブルから述語を検索
<decode_pname>.     predname/number --> predname + name に分解
<exist_fname>       述語名がすでに登録されているかチェック
<isatom>            アトムかどうか
<isuser>.           ユーザ定義述語かどうか
<isrecursive>.      再帰述語かどうか
<is_num>            数値かどうか
<is_file>           ファイル変数かどうか
<is_writable>.      ファイルが書き込み可能かどうか
<is_funcsys>        述語が関数的かどうか
.   isfunc~
<is_nofuncsys>      述語が関数的でないか
.   isnonfunc+
<isreduced>.        述語が簡約されているか
<isnoreduced>.      述語が簡約されていないか
<advance>           一文字読み込み
.   adv
<skipline>          CRまで読み飛ばす
<tputc>.            tee putchar
<Pterm>             項の表示
.   init_pp.
.   print_pp
.   .   Ppst_content
.   scanpst_term
.   .   addpst+
.   .   scanpst_functor+
<Psequence>         項の並びの表示
.   Pvar
.   .   voccurrence~
<Pclause>           項の並びの表示(+制約)
.   Pclause_core
```

. init_pp~
. print_pp~
. scanpst_clause
<Peclause> 項の並び(eclause)の表示
. Peclause_core
. init_pp~
. print_pp~
. scanpst_eclause
<P_hclause> horn clauseの表示
. init_pp~
. print_pp~
. scanpst_clause~
. P_hclause_sub+
<P_dclause> 新述語導入節の表示
. Pclause_core~
. init_pp~
. print_pp~
. scanpst_clause~
<Showhorn> CAHCの表示
. Pcahc_core
. init_pp~
. print_pp~
. scanpst_clause~
<Showfunc> 述語定義の表示
<Pgoal> ゴールの表示
. Peclause_core~
. init_pp~
. print_pp~
. scanpst_clause~
. scanpst_eclause~
<sscanf>.
<hash>
<streq>.
<strlen>.
<strcat>.
<strncpy>.
<Arg3>
<Predicate> 述語探索(ない場合には新規登録)
<Predname>. 述語名
<new> user heapへのalloc
. alloc~
<Nfunc> func構造体のセット
. funcalloc+
<Nterm> term構造体のセット
. Termalloc+
. mediterm+
. tempterm+
<tprint0> tee print (no argument)
<Arg>. 項の引数
<down> deref

. is_voidvar+
<error_detail>
<is_pst>. 項がPSTかどうか
<snew> system heapへのalloc
. salloc
<Nclause> clause構造体のセット
<MEMORY_ALLOC> 各ヒープへのalloc
<Component>. 述語の引数成分
<Arg1> 項の第1引数
<Arg2> 項の第2引数
<undo> user stackのrestore
. heap+
<is_clause>. 項がclauseかどうか
<is_functor>. 項が複合項かどうか
<Pred>. 項の述語
<head_of_list>. リストのcar
<tail_of_list>. リストのcdr
<is_int> 項がintegerかどうか
<is_list>. 項がリストかどうか
<num_value>. 数値の値
<Nlist> list構造体のalloc
<find_pstitem> pst構造を検索
<upush> user stackにpush
<error> エラーメッセージ表示
. freeheap~
. garbagecollect~
<next> 一文字読み込み
. feof+
. ferror+
<is_atomic>. 項がatomかどうか
<is_string> 項が文字列かどうか
<str_value>. 文字列の値
<isvar>. 項が変数かどうか
<issystem>. 組み込み述語かどうか
<tprint1> tee print (1 argument)
<tprint2> tee print (2 arguments)
<vname>. 変数名
<Rterm> 項の読み込み
. Rterm_half+
. . check~
. . skip+
. . Rlist+
. . . notconst_list+
. . Rpst+
. . . insert_pstobj+
. . isconst~
. . notconst
. . Nfile+
. . op_search
. . varsearch+

```
.   .   Rtoken
.   .   .   bracket+
.   .   .   is_varname+
.   .   .   isdigit~
.   .   .   isxdigit
.   .   .   read_comments+
.   .   .   read_digits+
.   .   .   read_hexa+
.   .   .   read_spechar+
.   .   .   specialchar
.   .   prefix_is_atom+
.   .   .   is_term_end+
.   Rterm_leftover+
.   .   isconst~
.   .   notconst~
.   .   op_search~
.   .   Rtoken~
<renum_pvars>.  PST変数の番号付け替え
<Nenv>  変数環境alloc
.   ealloc+
<Pterm_core>.   項の表示
.   Peclause_core~
.   Pfunctor+
.   .   quote_needed+
.   .   .   is_lower+
.   .   .   alphabet+
.   Ppst+
.   .   pp_number.
.   .   Ppst_content~
.   .   Ppst_content2+
.   Pvar~
<kanzi>. .


==========not called=========
().
(NL)
(STB)
(tprint4)
(disp_func_def)
(mark_component_checked_all)
.   set_all_body_component+
(NEW)
.   sizeof~
(index_funclist)
(DEBUG)
.   Pcahc_core~
.   Pclause_core~
.   Peclause_core~
.   init_pp~
```

```
.   print_pp~
.   scanpst_clause~
.   scanpst_eclause~
.   scanpst_term~
(P_var)
.   Pclause_core~
(showvar)
(alpha)
(delimitchar)
(quotesign)
(white)
(is_root)
.   Trace_False~
.   Trace_Goal~
.   Trace_True~
.   extend~
.   Psolution+
.   on_interrupt_refute+
.   proceed_node~
(Pvpair)
(sort_clause)
.   insert_clause+
(merge_pst_objects)
.   env+
.   safe_unify~
.   unify~
```

◎cu-Prolog第三版 関数インデックス
関数名によりソートされている。

```
タイプ  関数名          モジュール名
#define ARG_EQ          <tr_sub.c>
#define ARG_FALSE               <tr_sub.c>
#define ARG_TRUE                <tr_sub.c>
#define ATOMIC_TYPE             <include.h>
#define Arg(T,N)                <include.h>
#define Arg1(T)         <include.h>
#define Arg2(T)         <include.h>
#define Arg3(T)         <include.h>
#define BACKTRACK               <include.h>
#define BL              <print.c>
#define BL              <read.c>
#define BL              <varset.h>
#define BR              <print.c>
#define BR              <read.c>
#define BR              <varset.h>
#define BRACKET         <include.h>
#define CATMAX          <jpsgsub.c>
#define CHEAP_SIZE              <include.h>
```

```
#define EXTRACT          <unify.c>
#define FALSE            <include.h>
#define FALSE_REGISTERED                 <include.h>
#define FILE_POINTER             <include.h>
#define FILE_TYPE                <include.h>
#define FINITEFUN                <include.h>
#define FLOAT_NUM                <include.h>
#define FLT_EPSILON              <include.h>
#define FROM_CONC                <syspred1.c>
#define FROM_CONC                <syspred2.c>
#define FROM_NAME                <syspred1.c>
#define FROM_NAME                <syspred2.c>
#define FULLSTOP                 <include.h>
#define FX               <defsysp.c>
#define FY               <defsysp.c>
#define HASH_SIZE                <include.h>
#define HEAP_SIZE                <include.h>
#define INFIX            <include.h>
#define INPUT            <syspred1.c>
#define INT_NUM          <include.h>
#define Is_Leap          <include.h>
#define Is_Modular               <include.h>
#define Is_Msolvable             <include.h>
#define Is_Normaltrace           <include.h>
#define Is_Notrace               <include.h>
#define Is_Steptrace             <include.h>
#define Is_Trace                 <include.h>
#define Is_ctnormal              <include.h>
#define Is_ctnotrace             <include.h>
#define Is_ctstep                <include.h>
#define KEYIN            <include.h>
#define LC               <print.c>
#define LC               <read.c>
#define LC               <varset.h>
#define LIST_TYPE                <include.h>
#define Leap_mode                <include.h>
int      Llevel(t,e,nv)          <syspred1.c>
void     LtoC(t,e,pos,flag)              <syspred1.c>
void     LtoP(t,e,tt,depth)              <syspred1.c>
#define          MAIN            <main.c>
#define MEDIUM           <include.h>
#define MEMORY_ALLOC(X,Y,F)              <include.h>
#define MODULAR_DEFINED          <include.h>
#define Modmax_def               <include.h>
#define Modular_mode             <include.h>
#define Msolvable_mode           <include.h>
#define N                <print.c>
#define N                <read.c>
#define N                <varset.h>
#define NAME             <include.h>
```

```
#define NAME_MAX                <include.h>
#define NAME_SIZE               <include.h>
#define NEW             <new.c>
#define NEWPRED         <include.h>
#define NL              <include.h>
#define NOEXTRACT               <unify.c>
#define NONFUNC         <include.h>
#define NON_UNFOLDABLE          <include.h>
#define NOREDUCED_CLAUSE                <mainsub.c>
#define NOT_CONSTANT_TERM               <include.h>
#define NUMBER          <include.h>
struct_clause*  Nclause(head,body,flag)         <new.c>
struct_cset*    Ncset(flag)             <tr_sub.c>
struct_eclause* Neclause(val,env,tail,flag)             <new.c>
struct_pair*    Nenv(n)         <new.c>
struct_node*    Newnode(goal,icons,env,nlink,nlast)             <refute.c>
struct_term*    Nfile(x)                <new.c>
struct_func*    Nfunc(ftype,n,a)                <new.c>
struct_clause*  Nlist(head,body,flag)           <new.c>
struct_term*    Nnum(nbuf,flag)         <new.c>
struct_term*    Nnum_val(x,flag)                <new.c>
#define Normal          <jpsgsub.c>
#define Normaltrace_mode                <include.h>
#define Notrace_mode            <include.h>
struct_pst*     Npst(flag)              <new.c>
struct_term*    Npst_item(p,pobj,next)          <new.c>
#define Npstobj(Head,Env,Tail,Flag)             <defsysp.c>
#define Npstobj(Head,Env,Tail,Flag)             <include.h>
#define Npstobj(Head,Env,Tail,Flag)             <unify.c>
struct_term*    Nstr(x,flag)            <new.c>
struct_term*    Nterm(n,flag)           <new.c>
struct_term*    Nvar(nbuf,flag)         <new.c>
#define OUTPUT          <syspred1.c>
void    PCat(t,e,f)             <jpsgsub.c>
#define POSTFIX         <include.h>
#define PRED_IN_LINE            <mainsub.c>
#define PREFIX          <include.h>
#define PST_ITEM_TYPE           <include.h>
#define PST_TYPE                <include.h>
void    P_csnumber(cs,mode)             <tr_sub.c>
void    P_dclause(cl,e)         <print.c>
void    P_hclause(cl,e)         <print.c>
void    P_hclause_sub(cl,e)             <print.c>
void    P_status()              <tr_sub.c>
void    P_var(vlist)            <print.c>
int     Panswer(root,vlist)             <refute.c>
void    Pbinding(vlist,env)             <refute.c>
void    Pcahc_core(c,cst,e)             <print.c>
void    Pclause(c,e)            <print.c>
void    Pclause_core(c,e)               <print.c>
```

```
void    Pcmp(cmp)                   <trans.c>
void    Pcset_cstr(cs)              <tr_sub.c>
void    Pcset_def(cs)               <tr_sub.c>
void    Peclause(ec)                <print.c>
void    Peclause_core(ec,d)                 <print.c>
void    Penergy(cl)                 <tr_sub.c>
void    Pfunctor(t,e,d)             <print.c>
void    Pgoal(n)                    <print.c>
void    Ppst(t,e,d)                 <print.c>
void    Ppst_content(ptt,d)                 <print.c>
void    Ppst_content2(ptt,env,d)                        <print.c>
#define Pred(T)             <include.h>
struct_func*    Predicate(fname,arity)          <new.c>
#define Predname(T)                 <include.h>
void    Psequence(t,e,d)                    <print.c>
void    Psubcat(t,e)                <jpsgsub.c>
void    Pterm(t,e)                  <print.c>
void    Pterm_core(t,e,d)                   <print.c>
struct_clause*  PtoL(t)             <syspred1.c>
void    Ptree(t,e)                  <jpsgsub.c>
void    Pvar(t,n)                   <print.c>
void    Pvariant(va)                <tr_split.c>
void    Pvpair(va)                  <tr_split.c>
#define Q                   <print.c>
#define Q                   <read.c>
#define Q                   <varset.h>
#define REDUCEDFUN                  <include.h>
#define REDUCED_DEF                 <include.h>
#define REFMAX              <include.h>
#define REGISTERED                  <include.h>
#define REMOVED             <include.h>
struct_term*    Rlist(flag)                 <read.c>
struct_term*    Rpst(flag)                  <read.c>
struct_term*    Rterm(n,flag)               <read.c>
struct_term*    Rterm_half(n,flag,m)                <read.c>
struct_term*    Rterm_leftover(n,m,flag,t)                      <read.c>
int     Rtoken()                    <read.c>
#define SAFE                <unify.c>
#define SG                  <print.c>
#define SG                  <read.c>
#define SG                  <varset.h>
#define SHEAP_SIZE                  <include.h>
#define SP                  <print.c>
#define SP                  <read.c>
#define SP                  <varset.h>
#define SPECIFIED                   <syspred1.c>
#define SPYFUN              <include.h>
#define STAY_IF             <include.h>
#define STAY_IF_FALSE_PRED                  <include.h>
#define STAY_IF_TRUE_PRED                   <include.h>
```

```
#define STB(F)              <include.h>
#define STE                 <include.h>
#define STINGY              <include.h>
#define STRING              <include.h>
#define SUSPEND             <include.h>
#define SYSFAIL             <include.h>
#define SYSFUN              <include.h>
#define SYSNO               <include.h>
#define SYSPRED             <defsysp.c>
#define SYSTRUE             <include.h>
void    Showfunc(f)                 <print.c>
void    Showhorn(c,cst,e)                   <print.c>
void    Shownewfunc()               <print.c>
#define Steptrace_mode              <include.h>
#define TB                  <include.h>
#define TE                  <include.h>
#define TEMPFUN             <include.h>
#define TEMPORAL                    <include.h>
#define TEMPORAL_DEFINED                    <include.h>
#define TNTB                <include.h>
#define TNTE                <include.h>
#define TREEMAX             <jpsgsub.c>
#define TRUE                <include.h>
#define TSTB                <include.h>
#define TSTE                <include.h>
#define TTB                 <include.h>
#define TTE                 <include.h>
#define TYPE1SYS                    <include.h>
#define TYPE1SYS_REDUCED                    <include.h>
#define TYPE2SYS                    <include.h>
#define TYPE2SYS_REDUCED                    <include.h>
#define Termalloc(a)                <new.c>
#define Termalloc(a)                <new.c>
void    Trace_Answer(root)                  <refute.c>
void    Trace_False(n)              <refute.c>
void    Trace_False2(n)             <refute.c>
int     Trace_Goal(n)               <refute.c>
void    Trace_True(n)               <refute.c>
void    Trace_True2(n)              <refute.c>
void    Trace_Unification(n,s)              <refute.c>
#define UC                  <print.c>
#define UC                  <read.c>
#define UC                  <varset.h>
#define UL                  <print.c>
#define UL                  <read.c>
#define UL                  <varset.h>
#define UNIT_DEFINED                <include.h>
#define UNSAFE              <unify.c>
#define UNTOUCHED                   <include.h>
#define UP                  <include.h>
```

```
#define USERFUN          <include.h>
#define USTACK_SIZE              <include.h>
#define VACUITY_NOCHECK          <include.h>
#define VARNAME          <include.h>
#define VAR_GLOBAL_TYPE          <include.h>
#define VAR_PST_TYPE             <include.h>
#define VAR_VOID_TYPE            <include.h>
#define VERSION          <main.c>
#define VMAX             <include.h>
#define XF               <defsysp.c>
#define XFX              <defsysp.c>
#define XFY              <defsysp.c>
#define YF               <defsysp.c>
#define YFX              <defsysp.c>
void     abandon_transformation()                <trans.c>
int      abolish_pred(t,e)               <syspred1.c>
int      abomb_pred(t,e)         <defsysp.c>
void     add_clause(c,vlist,anum)                <tr_sub.c>
void     add_component_pst(f,a,ec)               <mainsub.c>
void     add_cs_to_set(cs,flag)          <tr_sub.c>
void     add_label(f,a,l,flag)           <mainsub.c>
void     add_set(s,flag)         <new.c>
void     add_to_set()            <trans.c>
void     addpst(t,e)             <print.c>
void     adv()           <read.c>
#define advance          <include.h>
int      alldigit(c)             <read.c>
int*     alloc(n)                <new.c>
void     allspy(n)               <mainsub.c>
#define alpha            <read.c>
#define alphabet(CH)             <print.c>
int      app_str(x,y,z,ez)               <syspred2.c>
int      apply(target,head,rest,anum)            <trans.c>
int      apply_add_clause(head,e0,ec)            <trans.c>
int      arg_pred(t,e)           <syspred1.c>
int      arg_type(a)             <tr_sub.c>
int      assert_pred(t,e)                <syspred1.c>
int      assertz_pred(t,e)               <syspred1.c>
#define atomic_equal(u,t)                <include.h>
void     attach(c,vl,anum)               <tr_split.c>
void     attach_arg(arg,c,vl)            <tr_split.c>
int      attach_pred(t,e,n,m,status)             <syspred1.c>
struct_node*     backtrack_node(n)               <refute.c>
#define bracket(C)               <read.c>
int      calc_1(x,y,z,e,op)              <syspred2.c>
int      calc_2(x,z,y,e,op)              <syspred2.c>
void     calc_all_var(f)         <mainsub.c>
void     calc_component()                <mainsub.c>
int      calc_pred(t,e,op)               <syspred2.c>
int*     challoc(n)              <new.c>
```

```
int     check(c)                <read.c>
int     check_INITDEF()         <trans.c>
void    check_all_unit(fl)              <mainsub.c>
void    check_constant_term(t)          <modular.c>
int     check_modularity(cst)           <mainsub.c>
void    check_recursion()               <mainsub.c>
void    check_unitpred(f)               <mainsub.c>
int     clause_pred(t,e,n,status)               <syspred1.c>
void    clear_predicate(f)              <syspred1.c>
void    clear_up_DEF()          <trans.c>
int     close_pred(t,e)         <syspred1.c>
int     cmp_clause(a1,a2)               <tr_sub.c>
int     cmp_cplxt(a1,a2)                <tr_sub.c>
int     cmp_flt(a1,a2)          <tr_sub.c>
int     cmp_fp(a1,a2)           <tr_sub.c>
int     cmp_int(a1,a2)          <tr_sub.c>
int     cmp_label(l1,l2)                <mainsub.c>
int     cmp_list(a1,a2)         <tr_sub.c>
int     cmp_pst(a1,a2)          <tr_sub.c>
int     cmp_str(a1,a2)          <tr_sub.c>
int     cmp_var(a1,a2)          <tr_sub.c>
int     cname_pred(t,e,nn)              <jpsgsub.c>
#define cnew(s)         <include.h>
struct_term*    cnlistmake(n)           <jpsgsub.c>
int     compare_pred(t,e)               <syspred2.c>
#define component_checked(F)            <include.h>
#define component_not_checked(F)                <include.h>
struct_clause*  compress_clause(cst)            <mainsub.c>
int     concat2_pred(t,e)               <syspred2.c>
int     concat_pred(t,e,n,status)               <syspred2.c>
struct_clause*  convert_list_to_clause(t,e,tt,ee,p,emsg)                <syspred1↓
↑.c>
struct_term*    copy_arg(t,i,flag)              <tr_split.c>
struct_clause*  copy_clause(cl,flag)            <tr_split.c>
struct_term*    copy_pst(pt,flag)               <tr_split.c>
struct_term*    copy_term(t,flag)               <tr_split.c>
int     count_pred(t,e)         <syspred2.c>
int     cs_status_type(st)              <tr_sub.c>
int     cu(t,e)         <modular.c>
int     cunify_pred(t,e)                <syspred1.c>
int     cut_pred(t,e,n)         <defsysp.c>
int     decode_pname(fname)             <mainsub.c>
void    decrement_vacuous(t)            <new.c>
int     default_pred(t,e)               <defsysp.c>
void    defclause()             <main.c>
void    defnewfunc()            <main.c>
void    defsyspred()            <defsysp.c>
void    delete_constraint(vl)           <tr_split.c>
void    delete_tmp()            <mainsub.c>
#define delimitchar(C)          <read.c>
```

```
int     diff_str(x,z,y,e,first)         <syspred2.c>
void    disp_func_def(f_from,f_to)              <mainsub.c>
void    divide_consts(cl)               <tr_split.c>
int     divstr_pred(t,e)                <syspred2.c>
#define down(p,t,e)             <include.h>
struct_pair*    ealloc(n)               <new.c>
struct_eclause* eclause_append(head,tail)               <modular.c>
struct_eclause* eclause_conc(ec1,ec2)           <tr_sub.c>
void    edit_predicate()                <main.c>
void    end_unfoldfold()                <trans.c>
int     energy(tm)              <tr_sub.c>
struct_pair*    env(t,e)                <unify.c>
int     eq_pred(t,e)            <syspred1.c>
int     eq_pred_sub(x,y,ex,ey)          <syspred1.c>
int     equal_pred(t,e)         <syspred1.c>
int     equalpred(t1,e1,t2,e2)          <syspred1.c>
void    error(s)                <main.c>
void    error_detail(t,e,s)             <main.c>
struct_func*    exist_fname(fname)              <new.c>
struct_term*    exist_termpair(t)               <tr_split.c>
struct_term*    exist_vpair(t)          <tr_split.c>
struct_node*    extend(n,status)                <refute.c>
int     extend_apply(target,head,rest,e0,f,s)           <trans.c>
int     fail_pred(t,e)          <defsysp.c>
int     file_open_pred(t,e,openmode)            <syspred1.c>
#define filep_value(Term)               <include.h>
void    filewrite(n)            <mainsub.c>
struct_pst_item*        find_pstitem(t,e)               <new.c>
#define finitefun(F)            <include.h>
int     firsthalf(h,w)          <syspred2.c>
int     foldunfold()            <trans.c>
void    freeheap()              <mainsub.c>
struct_cset*    from_to(s1,s2)          <trans.c>
#define funcalloc(a)            <new.c>
#define funcalloc(a)            <new.c>
struct_func*    funcsearch(fname,arity)         <new.c>
int     functor_pred(t,e)               <syspred1.c>
void    garbagecollect()                <main.c>
void    general_assert(t,e,flag)                <syspred1.c>
int     gensym_pred(t,e)                <syspred2.c>
int     geq_pred(t,e)           <syspred2.c>
int     greater_arg(a1,a2)              <tr_sub.c>
int     greater_pred(t,e)               <syspred2.c>
int     greater_term(t1,t2)             <tr_sub.c>
int     halt_pred(t,e)          <defsysp.c>
int     has_common_label(ec,cm)         <mainsub.c>
int     has_no_pst(t)           <tr_split.c>
int     has_no_var(t)           <tr_sub.c>
int     hash(fname)             <new.c>
int     have_nextgoal(n)                <refute.c>
```

```
#define head_of_list(Term)              <include.h>
void    helpmenu()              <mainsub.c>
#define in_cheap(X)             <modular.c>
#define in_sheap(X)             <include.h>
#define in_upper_heap(X,Y)              <modular.c>
void    index_func(fnew)                <new.c>
void    index_funclist(flist)           <new.c>
struct_itrace*  index_newflist(fl,it)           <new.c>
void    index_op(f,type,prec)           <defsysp.c>
void    index_set(chead,con,flag)               <new.c>
void    init_atoms()            <deisysp.c>
void    init_category()         <jpsgsub.c>
void    init_froles()           <defsysp.c>
void    init_heap_max()         <mainsub.c>
void    init_operator()         <defsysp.c>
void    init_pp()               <print.c>
struct_set*     init_set(n)             <refute.c>
void    init_status()           <main.c>
void    init_syspred()          <defsysp.c>
void    init_system_component(f,a)              <defsysp.c>
void    init_unfoldfold()               <trans.c>
struct_clause*  insert_clause(ct,cl)            <tr_sub.c>
void    insert_cs(cs,newcs)             <trans.c>
struct_eclause* insert_pstobj(val,tail,flag)            <read.c>
#define is_atomic(Term)         <include.h>
int     is_body_finite(f)               <mainsub.c>
#define is_clause(Term)         <include.h>
#define is_component_checked(F)         <include.h>
#define is_component_not_checked(F)             <include.h>
#define is_dead(n)              <refute.c>
#define is_eclause(Term)                <include.h>
#define is_file(Term)           <include.h>
#define is_funcsys(F)           <include.h>
#define is_functor(Term)                <include.h>
#define is_int(Term)            <include.h>
#define is_list(Term)           <include.h>
#define is_lower(CH)            <print.c>
int     is_modular_clause(cl)           <tr_sub.c>
int     is_modular_head(t)              <trans.c>
int     is_modular_literal(t)           <tr_sub.c>
#define is_nofuncsys(F)         <include.h>
#define is_num(Term)            <include.h>
#define is_pst(Term)            <include.h>
#define is_pstitem(Term)                <include.h>
#define is_readable(FP)         <include.h>
#define is_root(n)              <refute.c>
#define is_string(Term)         <include.h>
int     is_term_end(c)          <read.c>
#define is_tip(n)               <refute.c>
#define is_unitclause(Set)              <include.h>
```

```
#define is_varname(X)           <read.c>
#define is_voidvar(t)           <include.h>
#define is_writable(FP)         <include.h>
#define isallunit(F)            <include.h>
#define isatom(Term)            <include.h>
#define isconst(Term)           <include.h>
#define isconst_functor(Term)           <include.h>
#define isconst_list(L)         <read.c>
#define isdigit(X)              <read.c>
#define isfinite(F)             <include.h>
#define isfunc(F)               <include.h>
#define isnewpred(F)            <include.h>
#define isnonfunc(F)            <include.h>
#define isnoreduced(F)          <include.h>
#define isnospy(F)              <include.h>
#define isnotnewpred(F)         <include.h>
int     isop_pred(t,e,n,status)         <defsysp.c>
#define isrecursive(F)          <include.h>
#define isreduced(F)            <include.h>
#define isspy(F)                <include.h>
#define issystem(F)             <include.h>
#define isuser(F)               <include.h>
#define isvar(t)                <include.h>
#define isxdigit(X)             <read.c>
#define kanzi(CH)               <print.c>
#define kanzi(CH)               <read.c>
int     keyread(a)              <read.c>
int     leq_pred(t,e)           <syspred2.c>
int     less_pred(t,e)          <syspred2.c>
void    list_to_cat(t0,n)               <jpsgsub.c>
struct_clause*  list_to_clause(t,e)             <syspred1.c>
int     literalnumber(c)                <new.c>
void    loghandle(fname)                <mainsub.c>
void    main(argc,argv)         <main.c>
int     make_func(f,a,t,e)              <syspred1.c>
int     makelist_pred(t,e)              <syspred1.c>
void    mark_component_checked_all()            <mainsub.c>
int     match(clo,clt,e)                <modular.c>
int     match_func(t,e,f,ef,a,ea)               <syspred1.c>
void    match_term(t1,t2,e)             <modular.c>
#define mediterm(a)             <new.c>
#define mediterm(a)             <new.c>
int     memb_pred(t,e,n,status)         <syspred1.c>
struct_component*       merge_component(ca,cb,flag)             <mainsub.c>
struct_eclause* merge_pst_objects(target,e,object,f,safeflag)           <unify.c>
void    modular(c,vlist,anum)           <modular.c>
struct_clause*  modular_form(clist,vlist,anum)          <trans.c>
int     multiply_pred(t,e)              <syspred2.c>
char*   nalloc(n,flag)          <new.c>
int     name_pred(t,e)          <syspred1.c>
```

```
#define new(s)          <include.h>
struct_clause*  new_constraint(cmp)             <trans.c>
struct_clause*  new_pred_set(cc)                <trans.c>
#define newpred(F)              <include.h>
void    next()          <read.c>
struct_node*    next_goal(m,oldnode,btnode)             <refute.c>
int     nl_pred(t,e)            <syspred1.c>
#define nospyfun(F)             <include.h>
int     not_pred(t,e)           <defsysp.c>
#define notconst(Term)          <include.h>
#define notconst_list(L)                <read.c>
#define novar(Term)             <include.h>
struct_cset*    nth_cset(n)             <tr_sub.c>
struct_clause*  nth_literal(cl,n)               <tr_sub.c>
int     null_or_nil(t,e)                <jpsgsub.c>
#define num_value(Term)         <include.h>
int     numcomp_pred(t,e,op)            <syspred2.c>
#define numeric(X)              <read.c>
void    ocheck(p,t,e)           <unify.c>
void    oldlink(n)              <jpsgsub.c>
void    on_interrupt()          <main.c>
struct_set*     one_def_literal(f)              <tr_sub.c>
int     op_pred(t,e)            <defsysp.c>
struct_operator*        op_search(fname,otype)          <new.c>
int     open_pred(t,e)          <syspred1.c>
void    open_title()            <main.c>
int     or_pred(t,e,n,m,status)         <syspred1.c>
void    oscommand()             <mainsub.c>
int     pcon_pred(t,e,n)                <syspred1.c>
int     pickname(t,e)           <jpsgsub.c>
int     pnames_pred(t,e)                <defsysp.c>
int     pp_number(ec)           <print.c>
int     pred_compare(f1,f2)             <new.c>
int     prefix_is_atom(m)               <read.c>
void    prepare()               <main.c>
void    preprocess_all_unit(fl,flag)            <mainsub.c>
void    preprocess_constr_sub(flag)             <mainsub.c>
void    preprocess_constraints(fn)              <mainsub.c>
void    preprocess_unit(f,flag)         <mainsub.c>
void    print_ancestors(n)              <refute.c>
void    print_constant()                <main.c>
void    print_hash_table()              <new.c>
void    print_pp(d)             <print.c>
void    printtime()             <mainsub.c>
void    printtime()             <mainsub.c>
void    printtime()             <mainsub.c>
struct_node*    proceed_node(n,btnode)          <refute.c>
void    prolog_execution()              <main.c>
void    pst_add_unify(t,e,u,f)          <defsysp.c>
void    pst_add_unify_sub(entry,ol,e)           <defsysp.c>
```

```
void    pst_unify(t,e,u,f,safeflag)             <unify.c>
void    push_log(oldt,oldenv,newt)              <modular.c>
void    push_pstlog(oldt,oldenv,newt)           <modular.c>
void    putcursor()             <mainsub.c>
int     pvalue_pred(t,e)                <defsysp.c>
void    quit_prolog()           <mainsub.c>
void    quit_transformation()           <trans.c>
int     quote_needed(f)         <print.c>
#define quotesign               <read.c>
void    read_comments()         <read.c>
void    read_digits(i)          <read.c>
void    read_hexa(i)            <read.c>
int     read_pred(t,e)          <syspred1.c>
void    read_spechar(i)         <read.c>
void    readfile()              <mainsub.c>
#define readword(S)             <include.h>
void    rec_to_finite()         <mainsub.c>
void    recalc_component()              <mainsub.c>
void    recalc_voccur_sub(t,v)          <new.c>
void    recalc_voccurrence(cl,v)                <new.c>
struct_eclause* record_pstlists(ptt,e)          <unify.c>
struct_pst_item*        record_pstobjects(t,e)          <unify.c>
#define recursivefun(F)         <include.h>
struct_eclause* reduce_clause(cl,e)             <tr_sub.c>
struct_eclause* reduce_clause_m(cl,e)           <tr_sub.c>
struct_clause*  reduce_cstr(cst,vlist,anum,env)         <mainsub.c>
struct_clause*  reduce_substitute(cst,e)                <mainsub.c>
#define reducedfun(F)           <include.h>
int     refute(Root,n,Status)           <refute.c>
void    remove_from_CSTR(f)             <trans.c>
struct_clause*  remove_modular_literals(cl)             <tr_split.c>
struct_pst_item*        remove_pstitem(t,e)             <unify.c>
void    rename_var_names(v)             <main.c>
void    renum_pvars(pvs,vnum)           <main.c>
struct_clause*  reorder(cl,tc)          <trans.c>
void    reorder_clause(cl,tc)           <tr_sub.c>
void    replace_terms(c1,c2,vl)         <tr_split.c>
void    reset_component()               <mainsub.c>
int     reset_timer_pred(t,e)           <defsysp.c>
void    reset_voccurrence(v)            <new.c>
int     resolve(n0,n,sliteral,env)              <refute.c>
int     retract_pred(t,e,n,status)              <syspred1.c>
void    safe_unify(t,e,u,f,extflag)             <unify.c>
int*    salloc(n)               <new.c>
int     satisfiable(cl,anum)            <tr_sub.c>
void    scanpst_clause(t,e)             <print.c>
void    scanpst_eclause(ec)             <print.c>
void    scanpst_functor(t,e)            <print.c>
void    scanpst_term(t,e)               <print.c>
struct_term*    search_log(t,e)         <modular.c>
```

```
struct_term*    search_pstlog(t,e)              <modular.c>
int     see_pred(t,e)           <syspred1.c>
int     seen_pred(t,e)          <syspred1.c>
void    set_body_component(ff)          <mainsub.c>
void    set_category()          <jpsgsub.c>
void    set_eof()               <mainsub.c>
void    set_head_component(f)           <mainsub.c>
void    set_inputfile(n)                <mainsub.c>
void    set_new_def(c,vl,anum)          <trans.c>
void    set_temporal_def(f)             <trans.c>
struct_set*     setconcat(slist,s)              <new.c>
void    settimer()              <mainsub.c>
void    settimer()              <mainsub.c>
void    settimer()              <mainsub.c>
void    show_category()         <jpsgsub.c>
void    show_heap_max()         <mainsub.c>
void    show_newdefs()          <tr_sub.c>
void    show_pred_def(f)                <mainsub.c>
void    show_pred_roles(f)              <mainsub.c>
void    show_syspred_name()             <mainsub.c>
void    show_syspred_status(f)          <mainsub.c>
void    show_userpred_name()            <mainsub.c>
void    showdef(fname)          <mainsub.c>
void    showvar(v)              <print.c>
int     skip(c)         <read.c>
void    skip_cr()               <tr_sub.c>
#define skipline                <include.h>
#define snew(s)         <include.h>
struct_clause*  sort_clause(cl)         <tr_sub.c>
#define specialchar(C)          <read.c>
struct_compartment*     split(clist,vlist,anum)         <tr_split.c>
#define spychange(F)            <include.h>
#define spyfun(F)               <include.h>
void    spyswitch(fname)                <mainsub.c>
struct_clause*  startmodular(clist,vlist,anum)          <trans.c>
int     stay_pred(t,e)          <defsysp.c>
int     step_asking()           <tr_sub.c>
void    stepswitch()            <mainsub.c>
void    store_termpair(told,tnew)               <tr_split.c>
void    store_vpair(told,tnew)          <tr_split.c>
#define str_value(Term)         <include.h>
int     strcmp_pred(t,e)                <syspred2.c>
#define streq(p,q)              <include.h>
int     strlen_pred(t,e)                <syspred2.c>
int     substr_pred(t,e)                <syspred2.c>
int     subsume(t,e,u,f,flag)           <defsysp.c>
int     subsume_pst(t,e,u,f,flag)               <defsysp.c>
int     subsume_pstlist(x,y,e,flag)             <defsysp.c>
int     sum_pred(t,e)           <syspred2.c>
struct_clause*  surface_copy_clause(cl,flag)            <tr_sub.c>
```

```
int     system_function(t,e,n)          <defsysp.c>
int     system_pred(t,e,n,m,status)             <defsysp.c>
void    systemcommand(c)                <main.c>
int     tab_pred(t,e)           <syspred1.c>
#define tail_of_list(Term)              <include.h>
struct_clause*  target_literal(cl)              <tr_sub.c>
struct_clause*  target_literal(cl)              <tr_sub.c>
int     tell_pred(t,e)          <syspred1.c>
#define tempterm(a)             <new.c>
#define tempterm(a)             <new.c>
char*   termname(t,e)           <jpsgsub.c>
int     termnumber(t)           <modular.c>
struct_term*    termset(t,e,flag)               <modular.c>
struct_eclause* termset_pstobj(pobj,flag)               <modular.c>
struct_eclause* termset_pstobj_sub(pobj,e,flag)         <modular.c>
int     timer_pred(t,e)         <defsysp.c>
int     told_pred(t,e)          <syspred1.c>
struct_term*    tolist(c,flag)          <modular.c>
#define tprint0(X)              <include.h>
#define tprint1(X,V)            <include.h>
#define tprint2(X,V1,V2)                <include.h>
#define tprint3(X,V1,V2,V3)             <include.h>
#define tprint4(X,V1,V2,V3,V4)          <include.h>
#define tputc(X)                <include.h>
void    traceswitch()           <mainsub.c>
void    trans_routine()         <main.c>
struct_eclause* transform(precond,newc,newenv)          <modular.c>
int     tree_pred(t,e)          <jpsgsub.c>
void    treeprint(t,e,n)                <jpsgsub.c>
int     true_pred(t,e,n,status)         <defsysp.c>
void    truncate_varname(n,nbuf)                <main.c>
struct_term*    try_fold(c,n)           <modular.c>
int     tunify(t,e,u,f,flag)            <unify.c>
int     type_pred(t,e)          <defsysp.c>
int     unbreak_pred(t,e)               <defsysp.c>
void    undo(u)         <new.c>
void    unfold_cstr(cs)         <trans.c>
void    unfold_derivation(cs)           <trans.c>
void    unify(t,e,u,f)          <unify.c>
void    unify_merge_psts(target,object,safeflag)                <unify.c>
void    unify_pst_extract(t,e,u,f)              <unify.c>
void    unify_pstlist_objects(entry,ol,e,safeflag)              <unify.c>
struct_term*    up_atomic(t,flag)               <modular.c>
struct_term*    up_const(t,flag)                <modular.c>
struct_term*    up_const_functor(t,flag)                <modular.c>
struct_clause*  up_eclause(ec,flag)             <modular.c>
void    up_init()               <modular.c>
struct_clause*  up_itrace_clause(cl,anum)               <modular.c>
struct_term*    up_pst(pt,e,flag)               <modular.c>
void    up_restore()            <modular.c>
```

```
void    upush(p)                <new.c>
#define userfun(F)              <include.h>
int     var_pred(t,e)           <syspred1.c>
struct_term*    var_trans(v,flag)               <tr_split.c>
struct_variant* variant(cl,flag)                <tr_split.c>
struct_variant* variant_v(cl,flag)              <tr_split.c>
struct_term*    varsearch(varname)              <new.c>
#define vcomponent(t)           <include.h>
#define vconstraint(t)          <include.h>
#define vdecrement(t)           <include.h>
#define vheadoccurrence(t)              <include.h>
#define vincrement(t)           <include.h>
#define vlink(t)                <include.h>
#define vname(t)                <include.h>
#define vnumber(t)              <include.h>
#define voccurrence(t)          <include.h>
int     vpair_length(vp)                <tr_split.c>
#define white           <read.c>
int     write_pred(t,e)         <syspred1.c>
void    writenewfunc()          <print.c>


--------end----------
```

```
/*-----------------------------------------------------------------------
*              cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                        1989 - 91
*      Programmed by:  Hiroshi Tsuda (tsuda@icot.or.jp)
*                      Hidetosi Sirai (sirai@sccs.chukyo-u.ac.jp)
*
* header files: include.h, funclist.h, varset.h, globalv.h, sysp.h,
*               syspdef.h
* Prolog: main.c, mainsub.c, new.c, read.c, print.c, refute.c, unify.c
* embedded predicates: defsysp.c, syspred1.c, syspred2.c, jpsqsub.c
* Constraint: modular.c, trans.c, tr_sub.c, tr_split.c
-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------
*                        << include.h >>
*     (define structures, macros, etc)
*
* 91.12 cu-Prolog III
-----------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

/*      CPUTIME : print CPU time for UNIX 4.2 BSD
*       if your system has times() function   #define CPUTIME 60
*       if your system is SUN4                #define SUN4 1
*              else                           #define CPUTIME 0
*/

#define CPUTIME   60

#define HEAP_SIZE       100000   /* user heap size */
#define SHEAP_SIZE      600000   /* system heap size */
#define ESP_SIZE         25000   /* environment heap size */
#define CHEAP_SIZE      500000   /* constraints/pst heap size */
#define USTACK_SIZE      10000   /* user stack size */
#define NAME_SIZE        50000   /* name string size */

/* Tee print macro */
#define tpute(X)        {if (wfp) pute(X,wfp);if (lfp) pute(X,lfp);}
#define tprint0(X)      {if (wfp) fprintf(wfp,X);if (lfp) fprintf(lfp,X);}
#define tprint1(X,V)    {if (wfp) fprintf(wfp,X,V);if (lfp) fprintf(lfp,X,V);}
#define tprint2(X,V1,V2)        {if (wfp) fprintf(wfp,X,V1,V2);\
                                 if (lfp) fprintf(lfp,X,V1,V2);}
#define tprint3(X,V1,V2,V3)     {if (wfp) fprintf(wfp,X,V1,V2,V3);\
                                 if (lfp) fprintf(lfp,X,V1,V2,V3);}
#define tprint4(X,V1,V2,V3,V4)  {if (wfp) fprintf(wfp,X,V1,V2,V3,V4);\
                                 if (lfp) fprintf(lfp,X,V1,V2,V3,V4);}
#define NL              tpute('\n')

#define readword(S)     fscanf(fp,"%s",S);if (lfp) fprintf(lfp,"%s",S);
#define skipline        while (cbuf != '\n') next()
#define KEYIN           (fp == stdin)
#define advance         (next(), adv())

/* string equal */
#define streq(p,q)      (*(p) == *(q) && strcmp(p,q) == 0)

/* type of token */
#define NAME            0
#define NUMBER          1
#define STRING          2
#define FILE_TYPE       3
#define VARNAME         4
#define BRACKET         5   /* ()[]{} */
#define COMMA           6   /* , */
#define FULLSTOP        7   /* . */
#define CONST_MARK      8   /* ; */

/* VT-100 Escape Sequence */
#define C_HIGHLIGHT "\033[01m"
#define C_UNDER     "\033[04m"
#define C_BLINK     "\033[05m"
#define C_REVERSE   "\033[07m"
#define C_NORMAL    "\033[0m"
#define C_SAVE   "\033[s"
#define C_LOAD   "\033[u"
#define C_CLS    "\033[2J"

/* storage type */
#define TEMPORAL        0
#define MEDIUM          1
#define ETERNAL         2
#define STINGY          3

/* flag for checking constant term used in Rterm and termset */
#define CONSTANT_TERM   1
#define NOT_CONSTANT_TERM 0

/* discrimination of term */
#define VAR_VOID_TYPE   1
#define VAR_PST_TYPE    2
#define VAR_GLOBAL_TYPE 3
#define ATOMIC_TYPE     4
#define PST_TYPE        5
#define PST_ITEM_TYPE   6
#define CLAUSE_TYPE     7
#define EQUATION_TYPE   8
#define LIST_TYPE       9
#define CONST_LIST_TYPE 10

struct term {                   /* atomic formula (literal) */
        union {
            int ident;           /* descriminated accordint to this */
            struct func *t_func; /* functor(predicate) name */
          } type;
        int t_arity;            /* arity. when < 0 complex const  */
        union {
          struct term *t_arg[1];        /* args */
          float n_value;
```

```
        char *s_value;
        FILE *f_value;
      } tag;
};

#define FLOAT_NUM    0
#define INT_NUM      1
/* #define STRING       2 */
#define FILE_POINTER 3

 /* num,string,file */
#define is_atomic(Term)    (Term->type.ident == ATOMIC_TYPE)

#define is_num(Term)       (is_atomic(Term) && (Term->t_arity <= INT_NUM))
#define is_string(Term)    (is_atomic(Term) && (Term->t_arity == STRING))
#define is_int(Term)       (is_atomic(Term) && (Term->t_arity == INT_NUM))
#define is_file(Term)      (is_atomic(Term) && (Term->t_arity == FILE_POINTER))
#define is_psi(Term)       ((Term)->type.ident == PSI_TYPE)
#define is_psiitem(Term)   ((Term)->type.ident == PSI_ITEM_TYPE)
#define is_list(Term)      (((Term)->type.ident == LIST_TYPE) ||\
                            ((Term)->type.ident == CONST_LIST_TYPE))
#define is_clause(Term)    ((Term)->type.ident == CLAUSE_TYPE)
#define is_eclause(Term)    ((Term)->type.ident == EXCLAUSE_TYPE)

#define num_value(Term)    ((Term)->tag.n_value)
#define str_value(Term)    ((Term)->tag.s_value)
#define filep_value(Term)  ((Term)->tag.f_value)
#define head_of_list(Term) (((struct clause *)Term)->c_term)
#define tail_of_list(Term) ((struct term *)((struct clause *)Term)->c_link)

#define is_readable(FP) (FP->_flag & _IOREAD)
#define is_writable(FP) (FP->_flag & _IOWRT)

#define is_functor(Term) ((Term)->type.ident > CONST_LIST_TYPE)
#define isconst_functor(Term)  ((Term)->t_arity <= 0)
#define isconst(Term)    (is_atomic(Term) |  \
                          (is_functor(Term) && (isconst_functor(Term))))
#define notconst(Term)  (isvar(Term) || is_psi(Term) ||\
                         ((! is_atomic(Term)) && (term->t_arity > 0)))
#define isatom(Term)     (is_atomic(Term) |  (Term->t_arity == 0))

#define Arg(T,N)    (T)->tag.t_arg[N] /* N+1 th argument of term T */
#define Arg1(T)     Arg(T,0)
#define Arg2(T)     Arg(T,1)
#define Arg3(T)     Arg(T,2)
#define Pred(T)     (T)->type.t_func  /* predicate symbol */
#define Predname(T)     (T)->type.t_func->f_name  /* predicate symbol name */

#ifndef FLT_EPSILON             /* =2^-23 */
/* #define FLT_EPSILON 1.19209290E-07 */
#define FLT_EPSILON 2E-07
#endif

#define atomic_equal(u,t) \
        (is_int(u) && ((int)num_value(t)==(int)num_value(u))) ||\
        (is_atomic(t) && (t->t_arity == u->t_arity) &&\
         ( (is_int(t) && ((int)num_value(t) == (int)num_value(u))) || \
           (is_num(t) && \
             ((float)fabs(num_value(t)-num_value(u)) \
              <= FLT_EPSILON*fabs(num_value(t)))) ||\
           (is_string(t) && (strcmp(str_value(t),str_value(u)) == 0)) ||\
               (is_file(t) && (filep_value(t) == filep_value(u)))))

struct var {
        int v_type;             /*  v_type = VAR_TYPE */
        int v_number;
        char *v_name;
        struct var *v_link;
        struct clause *v_constraint; /* constraint of CNIC */
        struct component *v_component; /* used in the component check */
        short int v_head_occur; /* var occurrence in head */
        short int v_occurrence; /* var occurrence */
};

#define isvar(t)       ( ((struct var *)t)->v_type <= VAR_GLOBAL_TYPE \
                         && ((struct var *)t)->v_type >= VAR_VOID_TYPE )
#define novar(Term)    (is_atomic(Term) ||\
                          (Term->type.ident == CONST_LIST_TYPE) ||\
                          (! isvar(Term)) && (Term->t_arity <= 0)))
#define is_voidvar(t)   (((struct var *)t)->v_type == VAR_VOID_TYPE)
#define vname(t)        (((struct var *)t)->v_name)
#define vnumber(t)      (((struct var *)t)->v_number)
#define voccurrence(t)   (((struct var *)t)->v_occurrence)
#define vheadoccurrence(t)    (((struct var *)t)->v_head_occur)
#define vincrement(t)   ((struct var *)t)->v_occurrence++
#define vdecrement(t)   ((struct var *)t)->v_occurrence--
#define vconstraint(t)        (((struct var *)t)->v_constraint)
#define vcomponent(t)        (((struct var *)t)->v_component)
#define vlink(t)        ((struct term *)(((struct var *)t)->v_link))

typedef int (*SYSFUNC)();       /* for system predicate */

struct component                /* component of arguments */
{
        struct func *c_label;
        struct component *c_next;
};

#define Component(F,N)   (F)->f_component[N] /* N+1 th argument of func F */

struct func {                   /* predicate (functor) */
        unsigned short int f_arity, f_number, f_mark;
        unsigned short int f_setcount;  /* number of definitions */
        char *f_name;
        union {
          struct set *f_set;    /* definition clauses */
          SYSFUNC f_sysfunc;    /* system function */
        } def;
        struct func *f_link;
        struct itrace *f_integ; /* integrate() history  */
```

```
        int f_unitcount;        /* number of unit defs */
        struct component *f_component[1], /* component of arguments */
};

/*  predicate(functor) type definition  */
#define USERFUN          0      /*  user function, default value */
#define SYSFUN           1      /*  system pred */
#define SPYFUN           2      /*  spy fun. or not */
#define REDUCEDFUN       4      /*  reduced fun. or not */
#define FINITEFUN        8      /*  finete fun. or not */
#define TEMPFUN         16      /*  temporary func */
#define NEWPRED         32      /*  new predicate */
#define NONFUNC         64      /*  non-functional (many solutions) */
#define TYPE1SYS         9      /*  system (functional) pred */
#define TYPE1SYS_REDUCED 13 /* reduced # of arguments system pred */
#define TYPE2SYS        65      /*  system (non functional) pred */
#define TYPE2SYS_REDUCED 69 /* reduced # of arguments system non functional */
#define NON_UNFOLDABLE   128 /* non unfoldable pred at the unfold/fold */
#define STAY_IF          256 /* stay if TRUE/FALSE for NON_UNFOLDABLE pred */
#define STAY_IF_TRUE_PRED 384
#define STAY_IF_FALSE_PRED 128
#define VACUITY_NOCHECK 512     /*  vacuity non check flag */
#define COMPONENT_CHECKED 1024  /* component checked */

/* #define      systemfun(F)    (F->f_mark) != SYSFUN
#define userfun(F)      (F->f_mark) &= (~SYSFUN)  */

#define issystem(F)     ( ((F->f_mark) & SYSFUN) != 0 )
#define isuser(F)       ( ((F->f_mark) & SYSFUN) == 0 )
#define isnonfunc(F)    ( ((F->f_mark) & NONFUNC) != 0 )

#define isfunc(F)       ( ((F->f_mark) & NONFUNC) == 0 )
#define is_funcsys(F)   (issystem(F) && isfunc(F))
#define is_nofuncsys(F) (issystem(F) && isnonfunc(F))
#define spyfun(F)       (F->f_mark) |= SPYFUN
#define nospyfun(F)     (F->f_mark) &= (~SPYFUN)
#define spychange(F)    (F->f_mark) ^= SPYFUN
#define isspy(F)        ( ((F->f_mark) & SPYFUN) != 0 )
#define isnospy(F)      ( ((F->f_mark) & SPYFUN) == 0 )
#define reducedfun(F)   (F->f_mark) |= REDUCEDFUN
#define isreduced(F)    ( ((F->f_mark) & REDUCEDFUN) != 0)
#define isnoreduced(F)  ( ((F->f_mark) & REDUCEDFUN) == 0)
#define finitefun(F)    (F->f_mark) |= FINITEFUN
#define recursivefun(F) (F->f_mark) &= (~FINITEFUN)
#define isfinite(F)     ( ((F->f_mark) & FINITEFUN) != 0)
#define isrecursive(F)  ( ((F->f_mark) & FINITEFUN) == 0)
#define newpred(F)      (F->f_mark) |= NEWPRED
#define isnewpred(F)    ( ((F->f_mark) & NEWPRED) != 0)
#define isnotnewpred(F) ( ((F->f_mark) & FINITEFUN) == 0)
#define isallunit(F)    (F->f_setcount == F->f_unitcount)
#define component_checked(F)    (F->f_mark) |= COMPONENT_CHECKED
#define component_not_checked(F)        (F->f_mark) &= (~COMPONENT_CHECKED)
#define is_component_checked(F) ( ((F->f_mark) & COMPONENT_CHECKED) != 0 )
#define is_component_not_checked(F) ( ((F->f_mark) & COMPONENT_CHECKED) == 0 )
```

```
/* type of operator */
#define PREFIX  0100
#define POSTFIX 0200
#define INFIX   0300

struct operator {
  struct func *o_func;
  int o_prec; /* precedence of operator 0-1200 */
  int o_type; /* type of operator: xf,yf,fx,fy,xfy,yfx,xfx */
     /*   bit pattern of o_type is: PREFIX=0100, POSTFIX=0200, INFIX=0300,
          leftdown = 0010, rightdown = 0001
          xf -- 0210, yf -- 0200, fx -- 0101, fy -- 0100,
          xfy -- 0310, yfx -- 0301, xfx -- 0311 */
  struct operator *o_link;  /* link to another operator */
};

struct clause {                 /* sequence of atomic formula */
        int c_type; /* CLAUSE_TYPE or LIST_TYPE */
        struct term *c_form;    /* atomic formula */
        struct clause *c_link;
};

struct set {                    /* definition horn clause */
        unsigned short int s_anumber;
        unsigned short int s_bodynumber;        /* number of body literals */
        struct clause *s_clause; /* Horn clause */
        struct set *s_link;
        struct term *s_vlist;
        struct clause *s_constraint; /* constraint clause */
};

struct cset{                    /* clause stack */
        struct clause *cs_clause;
        struct term *cs_vlist;
        unsigned short int cs_anumber;  /* v_number | p_number */
        unsigned short int cs_status;   /* 0: not unfolded      1: unfolded */
        unsigned short int cs_enum; /* the number of literals in the body */
        unsigned short int cs_number;   /* set number */
        struct cset *cs_mother; /* mother derivation clause */
        struct cset *cs_link;
};

#define is_unitclause(Set)  (Set->s_bodynumber == 0)
/* dummy definition (for non functional system predicate) */
#define DUMMY_DEF (struct set *)1

struct pair {                           /* environment for SS */
        struct term *p_body;    /* term */
        struct pair *p_env;     /* environment */
};

struct ustack {                 /* user stack */
        int *u_addr;            /* address */
        int u_val;              /* content */
};
```

```
struct node {                   /* node for Prolog refutation */
        struct clause *n_clause; /* goal */
        struct pair *n_env;     /* variable environment */
        struct set *n_set;      /* OR program clauses */
        struct node *n_link, *n_last;
        struct eclause *n_constraint; /* constraint of CANC */
        unsigned short int n_count, n_spy, n_tmp, n_scount;
        int *n_hp;
        struct pair *n_ep;
        struct ustack *n_usp;
};

struct eclause {        /*  environment + clause(copy) */
        int e_type;
        struct term     *e_form;        /* atomic formula */
        struct eclause  *e_link;        /* equiv eclause link */
        struct pair     *e_env;         /* formular environment */
};

struct itrace{          /* integrate trace */
        unsigned short int it_anumber, it_cnumber; /* #of var,literals (key) */
        struct clause   *it_clause; /* Horn clause (history) */
        struct itrace   *it_link;
};

struct pst {            /* Partial Specified Term */
        int type;       /* PST_TYPE */
        struct term *p_var; /* var comes here */
        struct eclause *p_lists; /* property lists */
};

struct pstvar {
        int v_type;             /*  v_type = VAR_PST_TYPE */
        int v_number;
        char *v_name;
        struct term *v_link;
        struct term *old_var;
};

struct pst_item {
        struct pair *p_var;  /* PST var */
        struct eclause *p_lists; /* property lists */
        struct pst_item *p_link; /* link to other items */
};

/*  deref: if (t,e) is var ,then p != NULL, else p == NULL  */
/*  t,p,e must be variables !!!!  */
#define down(p, t, e)\
        while(1){ \
            if(isvar(t)) {\
              if (is_voidvar(t)) {\
                   p=Anonymous_env; break; }\
              p = &e[vnumber(t)];\
              if(p->p_body == NULL) break;\
              t = p->p_body;\
              e = p->p_env;\
             }\
            else { p = NULL; break; } }

/* various modes of cu-Prolog */
#define Notrace_mode       tflag = 0
#define Normaltrace_mode   tflag = 1
#define Steptrace_mode     tflag = 2
#define Leap_mode          tflag = 3
#define Is_Notrace         (tflag == 0)
#define Is_Normaltrace     (tflag == 1)
#define Is_Steptrace     (tflag == 2)
#define Is_Leap          (tflag == 3)
#define Is_Trace         (tflag != 0)
#define Msolvable_mode sflag = 0
#define Modular_mode   sflag = 1
#define Is_Msolvable     (sflag == 0)
#define Is_Modular     (sflag == 1)

#define Is_ctnotrace (CTmode == 0)
#define Is_ctnormal (CTmode == 1)
#define Is_ctstep (CTmode == 2)
#define CTnotrace CTmode = 0
#define CTnormal CTmode = 1
#define CTstep CTmode = 2
/* c.t. trace(normal, step) begin&end */
#define TTB   if (CTmode != 0){
#define TTE   TE
/* c.t. step trace begin&end */
#define TSTB  if Is_ctstep {
#define TSTE  TE
/* c.t. normal trace begin&end */
#define TNTB  if Is_ctnormal {
#define TNTE  TE


/* trace begin & end */
#define TB              if (Is_Trace) {
#define STB(F)          if (Is_Trace && isspy(F) ) {
#define TE              NL; }
#define STE             TE

/* modularize fail */
/* #define MFAIL                (struct clause *)1 */

/* return value of the execution of system predicate */
#define SYSNO   1               /*      is not system pred.   */
#define SYSTRUE 2               /*      system pred. success    */
#define SYSFAIL 3               /*      system pred. fail       */
#define SUSPEND 4

#define TRUE 1
#define FALSE 0
```

```
/* refutation search staus flag  used in syspred.c, refute.c */
#define DOWN 1
#define UP   2
#define BACKTRACK 3

/* predicate symbol hash table size */
#define HASH_SIZE 253

#define NAME_MAX  256           /* size of name buffer */
#define REFMAX  10000           /* refutation max (Refcount) default */
#define Modmax_def  50  /* modularize max (MODULAR_MAX) default */

/* struct allocation macro   s:struct name */
#define snew(s)     (struct s *)salloc(sizeof (struct s) / sizeof (int))
#define cnew(s)     (struct s *)challoc(sizeof (struct s) / sizeof (int))
#define new(s)      (struct s *)alloc(sizeof (struct s) / sizeof (int))

#define MEMORY_ALLOC(X,Y,F) \
        switch (F) { \
            case TEMPORAL: \
              X=new(Y); break;\
            case MEDIUM: \
              X=cnew(Y); break; \
            default:\
              X=snew(Y); }

#define in_sheap(X) (( &sheap[0] <= ((int *)X)) && (((int *)X) < shp))
#define Npstobj(Head,Env,Tail,Flag)  Neclause(Head,Env,Tail,Flag)

/*   the maximum number of variables   */
#define VMAX 30

/*   for constraint transformation (trans.c tr_sub.c tr_split.c)   */
/*  values of cs->cs_status */
#define REMOVED 1
/* for CSTR_list */
#define UNTOUCHED 0
#define MODULAR_DEFINED 2
#define UNIT_DEFINED 3
/* for DEF_list */
#define DERIVATION 4
#define REGISTERED 5
#define FALSE_REGISTERED 6
#define REDUCED_DEF 7
/* for M-solvability */
#define TEMPORAL_DEFINED 8

struct compartment{
        struct clause *cmp_clause;
        struct compartment *cmp_link;
};

struct vpair{                   /* link between v1 and v2 */
        struct term *v1;        /* original var or pst */
        struct term *v2;        /* variant var or pst */
        struct vpair *v_link;
};

struct variant
{
        struct clause *v_clause; /* variant clause */
        struct term *v_var;      /* variable list in v_clause */
        struct vpair *v_pair;    /* variable correspondence */
        int v_anum;              /* # of v! # of pst */
};


/***************  global functions  *************/
#include "funclist.h"

/***************  global vars  *****************/
#if     MAIN == 1
#include "varset.h"
#else
#include "globalv.h"
#endif

/***************  system predicate  *************/
#if     SYSPRED == 1
#include "syspdef.h"
#else
#include "sysp.h"
#endif

/***************  heap, stack  *****************/
#if     NEW == 1
        int sheap[SHEAP_SIZE];  /* system heap */
        int *shp = sheap;
        int *SHEAPTOP = &sheap[SHEAP_SIZE];
        int heap[HEAP_SIZE];    /* user heap */
        int *Heap_Max = heap;
        int *hp = heap;
        int *HEAPTOP = &heap[HEAP_SIZE];
        int cheap[CHEAP_SIZE];  /* constraints/pst heap */
        int *CHEAPTOP = &cheap[CHEAP_SIZE];
        int *Cheap_Max = cheap;
        int *chp = cheap;
        struct pair eheap[ESP_SIZE];  /* environment heap */
        struct pair *ep = eheap;
        struct pair *Esp_Max = eheap;
        struct pair *ESPTOP = &eheap[ESP_SIZE];
        struct ustack ustack[USTACK_SIZE];      /* user stack */
        struct ustack *usp = ustack;
        struct ustack *STACKTOP = &ustack[USTACK_SIZE];
        struct ustack *Stack_Max = ustack;
        char nheap[NAME_SIZE];   /* name string heap */
        char *nhp = nheap;
        char *NHEAPTOP = &nheap[NAME_SIZE];
        struct func *hash_list[HASH_SIZE]; /* predicate hash table */
#else
```

```
        extern int sheap[], *shp, *SHEAPTOP;            /* system heap */
        extern int heap[], *hp, *Heap_Max, *HEAPTOP;    /* local heap */
        extern int cheap[], *chp, *Cheap_Max, *CHSAPTOP; /* constrains heap */
        extern struct pair cheap[], *cp, *Csp_Max, *CSPTOP;
        extern struct ustack ustack[], *usp, *Stack_Max, *STACKTOP;
/* user stack */
        extern char uheap[], *uhp, *UHEAPTOP;
        extern struct func *hash_list[];
#endif
```

```
/*************************************************************
*     cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 - 91
*************************************************************/
/*-------------------------------------------------------------
*                      <<< funclist.h >>>
*   external function definitions
-------------------------------------------------------------*/

/*  extern function def  */
/*      main.c          */
void prepare(),error_detail(), error(),systemcommand();
void oscommand(),set_inputfile(),readfile(),set_eof(),trans_routine();
void questionclause(),defclause(),push_status(),pop_status(),init_status();
void garbagecollect(),edit_predicate();
void open_file(), prolog_execution();
void defnewfunc(), rename_var_names(), truncate_varname();
void renum_pvars(), preprocess_constraints();

/*      mainsub.c       */
void traceswitch(),stepswitch(),spyswitch(),showdef(),loghandle();
void check_recursion();
void recalc_f_roles();
int not_vacuous();
void reduceswitch(),putcursor(),disp_func_def();
void allspy(),helpmenu(),filewrite();
void freeheap();
void printline(),settimer(),quit_prolog();
void delete_tmp();

/*      defsysp.c       */
void init_syspred();
void init_atoms();
void init_operator();
void defsyspred();
int system_function();
int cut_pred();
int fail_pred();
int halt_pred();
int abomb_pred();
int true_pred();
int op_pred();
int isop_pred();
void index_op();
int not_pred();
int unbreak_pred();
int pnames_pred(), pvalue_pred(), type_pred();
int default_pred(), subsume(), subsume_pst();
int subsume_pstlist();
void pst_add_unify(), pst_add_unify_sub();
int reset_timer_pred(), timer_pred();
int stay_pred();

/*   syspred1.c  */
int monb_pred();
int execute_pred();
int or_pred();
int read_pred();
int open_pred();
int see_pred();
int seen_pred();
int tell_pred();
int told_pred();
int close_pred();
int pcon_pred();
int attach_pred();
int cunify_pred();
int write_pred();
int nl_pred();
int tab_pred();
int var_pred();
int equal_pred();
int eq_pred();
int eq_pred_sub();
int equalpred();
int assertz_pred();
int assert_pred();
void general_assert();
struct clause *list_to_clause();
int retract_pred();
int retract_pred_sub();
void clear_predicate();
int abolish_pred();
int makelist_pred();
int Llevel();
void LtoP();
struct clause *PtoL();
int name_pred();
void LtoC();
struct term *CtoL();
int arg_pred();
int functor_pred();
int make_func();
int match_func();
int clause_pred();
struct term *Clause_to_List();

/* syspred2.c */
int sum_pred();
int multiply_pred();
int calc_pred();
int calc_1();
int calc_2();
int greater_pred();
int less_pred();
int geq_pred();
int leq_pred();
int numcomp_pred();
int compare_pred();
```

```
int concat_pred();
int app_str();
int diff_str();
int concat2_pred();
int divstr_pred();
int strlen_pred();
int stremp_pred();
int count_pred();
int gensym_pred();
int default_pred();
int substr_pred();

/* jpsgsub.c */
void show_category();
void init_category();
void list_to_cat();
void set_category();
int tree_pred();
void Ptree();
void oldlink();
void treeprint();
int null_or_nil();
void PCal();
void Psubcat();
char *termname();
int pickname();
struct term *enlistmake();
int cname_pred();

/*      new.c           */
int *galloc(),*alloc();
struct pair *ealloc();
char *nalloc();
int hash();
void print_hash_table();

/* name flag for nalloc() */
char *nalloc();
void index_func(), index_funclist();
struct itrace *index_newflist();
void reset_voccurrence(),recale_voccurrence();
void recale_pred_value();
struct operator *op_search();
struct term *Nvar(),*Nterm();
struct term *Nnum(), *Nnum_val(), *Nstr(), *Nfile();
struct pst *Npst();
struct term *Npst_item();
struct pst_item *find_pstitem();
struct clause *Nlist(), *Nclause();
struct func *Nfunc(),*funcsearch();
struct func *Predicate(),*exist_fname();
struct term *varsearch();
struct pair *Nenv();
struct node *Nnode();
struct cnode *Ncnode();
struct eclause *Neclause();
struct allvar *Nallvars();
void check_pred_defcount();
void upush(),undo(),add_set(),index_set();
struct term *literalcopy();
void show_heap_max(),init_heap_max();


/*      print.c         */
void Pterm(),Pterm_core(),Psequence(),Pallvar(),Pvar(), Pfunctor();
void writenewfunc(), Ppst();
void Pclause(),P_var(), P_dclause();
void Showborn(),Showfunc(),Shownewfunc();
void Pgoal();
int quote_needed();
void Peclause();


/*      read.c          */
void adv();
int check(), skip(), keyread(), alldigit();
void period(), next(), read_hexa(), read_digits();
void read_comments(), read_spechar();
int Rtoken(), is_term_end(), prefix_is_atom();
struct term *Rlist(),*Rterm(), *Rterm_half(), *Rterm_leftover();
struct term *Rfunc(),*Rhead();
struct clause *Rclause();
struct clause *Rconstraint();
struct term *Rliteral(), *Rvar(), *Rpst();
void register_psttable();
struct eclause *insert_pstobj();

/*      modular.c       */
void modular();
int cu();
struct term *tolist(),*termset(), *up_pst();
struct eclause *termset_pstobj(), *termset_pstobj_sub();

void up_init();
void up_restore();
struct clause *up_eclause();
struct clause *up_list_to_clause();
struct clause *up_itrace_clause();
struct set *up_func_def();
struct eclause *onestep_reduce(), *eclause_append();

struct term *up_atomic(), *up_const(), *up_const_functor();
struct term *try_fold();
struct eclause *transform();
void match_term();

/*      refute.c        */
int refute();
struct node *Newnode();
struct node *backtrack_node();
```

```
struct set *init_set();
int Panswer();

/*      unify.c         */
int tunify();
void ocheck(),unify(),safe_unify();
void pst_unify();
struct eclause *record_pstlists();
void unify_pstlist_objects(), unify_merge_psts();
struct pst_item *remove_pstitem(), *record_pstobjects();

/* transform.c */
int check_INITDEF();
void clear_up_DEF();
void add_to_set();
struct clause *startmodular();
struct clause *modular_form();
struct compartment *Ncomp();
struct clause *copy_clause();
struct term *copy_term();
struct term *var_trans();
struct clause *new_constraint();
struct clause *new_pred_def();
void set_new_def();
int need_trans();
/* struct clause *restore_head(); */
struct term *restore_term();
struct term *var_reverse();
struct cset *target_clause();
struct cset *target_def();
void delete_csets();
int foldunfold();
int unfold();
int apply();
struct set *one_def_literal();
void register_newpred();

void Pvpair();
void Pcomp();
int stat();
void Pcset_def();
void Pcset_cstr();
void P_csnumber();
void P_status();
struct vpair *Nvpair();
struct cset *Ncset();
void add_clause();
void add_cs_to_set();
struct eclause *reduce_clause();
void reorder_clause();
int satisfiable();
struct clause *target_literal();
int energy();
struct clause *surface_copy_clause();
struct clause *ctail();
struct eclause *ectail();
int has_pred();
int is_vacuous();
int is_modular_clause();
int is_modular_literal();
int has_no_var();
/* struct clause *Nhornclause(); */
struct eclause *eclause_conc();
struct clause *sort_clause();
struct clause *insert_clause();
int greater_term();
int arg_type();
int greater_arg();
int cmp_var();
int cmp_cplxt();
int cmp_list();
int cmp_flt();
int cmp_int();
int cmp_str();
int cmp_fp();
int have_def();
void init_unfoldfold();
void end_unfoldfold();
int step_asking();
struct cset *nth_cset();
struct clause *nth_literal();
void abandon_transformation();
void quit_transformation();
void skip_cr();
void show_newdefs();

struct compartment *split();
void clear_vconstraint();
void delete_constraint();
void attach();
void attach_term();
void attach_arg();
void replace_terms();
```

```
/*-----------------------------------------------------------------------
*                cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 -91
*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------
*                <<<<   varset.h    >>>>
*                initialize global vars
-----------------------------------------------------------------------*/

long CONSTRAINT_HANDLING_TIME = 0L;

FILE *fp,*wfp,*lfp; /* read file pointer, write fp, log fp */
int lty;
int cbuf;                /* character buffer */
struct ustack *uTop;
int ECHO_BACK = 0;
int Handle_Undefined = FALSE; /* fail return */
int Print_Depth = 12;

int tflag;   /* trace flag  0->off, 1->on 2->step trace on */
int sflag;   /* solution mode flag  1->all solutions, 0 >one solution */
int CTmode;                      /* trace mode 0,1,2 */
int refute_node_count = 1;       /* refute node counter in c.t. */

/* Classification of Characters */
#define BL  001  /* blank */
#define UC  002  /* Upper Character */
#define LC  003  /* Lower Character */
#define UL  004  /* Undef Line */
#define N   005  /* Numeric */
#define SG  006  /* sign, +- */
#define SP  007  /* special character */
#define Q   010  /* quote */
#define CT  011  /* Cut */
#define CM  012  /* comment character */
#define BR  013  /* Brackets, Commas */
#define CO  014  /* Constraint Marker */

int char_type[128] = {
/* 00, 01, 02, 03, 04, 05, 06, 07, 10, 11, 12, 13, 14, 15, 16, 17 */
   BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL,
/* 20, 21, 22, 23, 24, 25, 26, 27, 30, 31, 32, 33, 34, 35, 36, 37 */
   BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL, BL,
/* sp,  !,  "   #   $   %   &   '   (   )   *   +   ,   -   .   / */
   BL, CT,  Q, SP, SP, CM, SP,  Q, BR, BR, SP, SG, BR, SG, SP, SP,
/*  0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ? */
    N,  N,  N,  N,  N,  N,  N,  N,  N,  N, SP, CO, SP, SP, SP, SP,
/*  @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O */
   SP, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC,
/*  P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _ */
   UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, UC, BR, SP, BR, SP, UL,
/*  `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o */
   SP, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC,
/*  p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ del */
   LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, LC, BR, BR, BR, SP, BL };

char nbuf[NAME_MAX];     /* name buffer */
int  v_number = 0;               /* temporary var number */
int  p_number = 0;
struct term *v_list = NULL;              /* temporary var list */
struct term *pv_list = NULL;
struct func *f_list = NULL;       /* new function list entry */
/* struct node *n_last   NULL;  */ /* node list */
struct operator *o_list = NULL;
struct itrace *newf_list = NULL;         /* new function definition  */
struct pst_item *pstable;

int Def_Modified = 0;            /* def modified flag */

/*  system predicates in cu prolog  */
struct func *LIST,*UNIFY;
struct term *NIL,*FAIL, *END_OF_FILE;
struct clause *MFAIL;
struct term *XF_P, *YF_P, *FX_P, *FY_P, *XFX_P, *XFY_P, *YFX_P;
struct term *S_GLOBAL_VAR, *S_VAR, *S_INTEGER, *S_FLOAT;
struct term *S_STRING, *S_FILE_POINTER, *S_PST, *S_CLAUSE;
struct term *S_LIST, *S_FUNCTOR, *S_ATOM, *S_PSTOBJ;
struct term *S_EQ, *S_GREATER, *S_LESS;
struct term *Anonymous_var;
struct pair *Anonymous_env;

int Refcount;   /* maximum of refute counter */
int MODULARMAX; /* maximum number of Variables in Transformation */

struct node  *Last_BT, *Last_SKIP;

char    genname[8] = "c",        /* generate function name c0,c1,... */
        logfile[32] = "no",      /* no log  */
        Anonymous_VarName[4]="_";
int tokentype, reread;
int GENSYM = 0;

#include <setjmp.h>
jmp_buf reset;           /* error recovery */
jmp_buf unbreak_reset;           /* trace --- unbreak */
```

```
/*-------------------------------------------------------------------
*               cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                        1989--91
-------------------------------------------------------------------  */
/*-------------------------------------------------------------------
*               << globalv.h >>
*               global variable external reference
-------------------------------------------------------------------  */

extern long CONSTRAINT_HANDLING_TIME;

extern FILE *fp,*wfp,*lfp; /* read file pointer, write fp, log fp */
extern int tty;
extern int cbuf;        /* character buffer */
extern struct ustack *utop;     /* save user stack pointer */
extern int ECHO_BACK;
extern int Handle_Undefined;  /* handling undefined predicates */
extern int Print_Depth;  /* maximum depth of printing */

extern int tflag;   /* trace flag  0 > off, 1 > on 2 >step trace on */
extern int sflag;   /* solution mode flag  1 >all solutions, 0 >one solution */
extern int Cmode;                       /* trace mode 0,1,2 */
extern int refute_node_count;   /* refute counter using in c.t. */

extern int char_type[128];
extern char nbuf[];             /* name buffer */
extern int tokentype, reread;
extern char     genname[8];             /* generated function name */
extern char     logfile[32];            /* log file name */
extern char     Anonymous_VarName[4];
extern int GENSYM;

extern int    v_number, p_number;       /* temporary var number */
extern struct term *v_list, *pv_list;   /* temporary var list */
extern struct func *f_list;      /* new function list entry */
extern struct operator *o_list;
extern struct node *n_last;             /* node list */
extern struct itrace *newf_list;        /* new function definition  */
extern struct pst_item *psttable;

extern int  def_Modified;               /* def modified flag */

extern int Refcount;            /* maximum of refute counter */
extern int MODULARMAX;          /* maximum number of Variables in Trans */

/*  system predicates in cu-prolog  */
struct func *LIST,*CUNIFY;
struct term *NIL,  *FAIL, *END_OF_FILE;
struct term *Anonymous_var;
struct pair *Anonymous_env;
struct clause *MFAIL;
struct term *XF_P, *YF_P, *FX_P, *FY_P, *XFX_P, *XFY_P, *YFX_P;
struct term *S_GLOBAL_VAR, *S_VAR, *S_INTEGER, *S_FLOAT;
struct term *S_STRING, *S_FILE_POINTER, *S_PST, *S_CLAUSE;
struct term *S_LIST, *S_FUNCTOR, *S_ATOM, *S_PSTORJ;
struct term *S_EQ, *S_GREATER, *S_LESS;
extern struct node  *last_BT, *last_SKIP;

#include <setjmp.h>
extern jmp_buf reset;
extern jmp_buf unbreak_reset;           /* trace --  unbreak */
```

```
/*-----------------------------------------------------------------
*              cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 91
-----------------------------------------------------------------*/
/*-----------------------------------------------------------------
*          <<syspdef.h>>
*     initialize system predicate variable
-----------------------------------------------------------------*/

/* system predicate in cu-Prolog */
struct func *ABOMB_P;
struct func *ABOLISH_P;
struct func *ARG_P;
struct func *ASSERTA_P;
struct func *ASSERTZ_P;
struct func *ASSERT_P;
struct func *ATTACH_P;
struct func *CAT_P;
struct func *CLAUSE_P;
struct func *CLOSE_P;
struct func *CMP_P;
struct func *CNAME_P;
struct func *CONCAT2_P;
struct func *CONCAT_P;
struct func *COUNT_P;
struct func *CUT_P;
struct func *DEFAULT_P;
struct func *DIVSTR_P;

struct func *EQUAL_P;
struct func *EQ_P;
struct func *EXECUTE_P;
struct func *FAIL_P;
struct func *FUNCTOR_P;
struct func *GENSYM_P;
struct func *GEQ_P;
struct func *GREATER_P;
struct func *HALT_P;
struct func *ISOP_P;
struct func *INTEG_P;
struct func *LESS_P;
struct func *LEQ_P;
struct func *MAKELIST_P;
struct func *MEMB_P;
struct func *MODULAR_P;
struct func *MULTIPLY_P;
struct func *NAME_P;
struct func *NL_P;
struct func *OP_P;
struct func *OPEN_P;
struct func *OR_P;
struct func *PCONSTRAINT_P;
struct func *PCONSTRAINT2_P;
struct func *READ_P;

struct func *RETRACT_P;
struct func *SEE_P;
struct func *SEEN_P;
struct func *SUBSTR_P;
struct func *STAY_P;
struct func *STRCMP_P;
struct func *STRLEN_P;
struct func *SUB_P;
struct func *T_P;
struct func *TAB_P;
struct func *TELL_P;
struct func *TOLD_P;
struct func *TREE_P;
struct func *TRUE_P;
struct func *UNBREAK_P;
struct func *VAR_P;
struct func *WRITE_P;

struct func *PNAME_P;

struct func *DEF_P;
struct func *QUERY1_P;
struct func *QUERY2_P;
struct func *NOT_P;
struct func *EQSIGN_P;
struct func *MKLIST_P;
struct func *CONSTRAINT_P;
struct func *CONSTRAINT2_P;
struct func *GREATER2_P;
struct func *GEQ2_P;
struct func *LESS2_P;
struct func *LEQ2_P;
struct func *EQUAL2_P;
struct func *EQ2_P;

struct func *PNAMES_P;
struct func *PVALUE_P;
struct func *TYPE_P;

struct func *RESET_TIMER_P;
struct func *TIMER_P;
```

```
/*-------------------------------------------------------------------
*              cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                        1989 -91
-------------------------------------------------------------------- */
/*-------------------------------------------------------------------
*              << sysp.h >>
*       (system predicate external reference)
-------------------------------------------------------------------*/

/*  functions included in cu prolog  */
extern struct func *ABOLISH_P;
extern struct func *ARG_P;
extern struct func *ASSERT_P;           /* assert */
extern struct func *ASSERTA_P;          /* asserta */
extern struct func *ASSERTZ_P;          /* assertz */
extern struct func *ATTACH_P;

extern struct func *CAT_P;
extern struct func *CLAUSE_P;
extern struct func *CLOSE_P;
extern struct func *CMP_P;
extern struct func *CNAME_P;
extern struct func *CONCAT_P;
extern struct func *CONCAT2_P;
extern struct func *COUNT_P;            /* count */
extern struct func *CUT_P;

extern struct func *DEFAULT_P;
extern struct func *DIVSTR_P;
extern struct func *EOF_P;
extern struct func *EQUAL_P;            /* equal ( = )  */
extern struct func *EQ_P;               /* eq (==) */
extern struct func *EXECUTE_P;
extern struct func *FAIL_P;
extern struct func *FUNCTOR_P;
extern struct func *GENSYM_P;
extern struct func *GEQ_P;
extern struct func *GREATER_P;
extern struct func *HALT_P;
extern struct func *ISOP_P;
extern struct func *INTEG_P;            /* integrate */
extern struct func *LESS_P;
extern struct func *LEQ_P;
extern struct func *MULTIPLY_P;
extern struct func *MAKELIST_P;         /* =.. */
extern struct func *MODULAR_P;          /* modularize */
extern struct func *MULTIPLY_P;
extern struct func *NAME_P;             /* name */
extern struct func *NL_P;               /* nl   */
extern struct func *OP_P;
extern struct func *OPEN_P;
extern struct func *OR_P;

extern struct func *PCONSTRAINT_P; /* print constraint */
extern struct func *PCONSTRAINT2_P;
extern struct func *READ_P;             /* read */
extern struct func *RETRACT_P;          /* retract */
extern struct func *SEE_P;
extern struct func *SEEN_P;
extern struct func *SUBSTR_P;
extern struct func *STAY_P;
extern struct func *STRCMP_P;
extern struct func *STRLEN_P;
extern struct func *SUM_P;
extern struct func *T_P;
extern struct func *TAB_P;
extern struct func *TELL_P;
extern struct func *TOLD_P;
extern struct func *TREE_P;
extern struct func *TRUE_P;
extern struct func *UNBREAK_P;
extern struct func *VAR_P;
extern struct func *WRITE_P;            /* write */

/* operators */
extern struct func *DEF_P;
extern struct func *QUERY1_P;
extern struct func *QUERY2_P;
extern struct func *NOT_P;
extern struct func *EQSIGN_P;
extern struct func *MKLIST_P;
extern struct func *CONSTRAINT_P;
extern struct func *CONSTRAINT2_P;
extern struct func *GREATER2_P;
extern struct func *GEQ2_P;
extern struct func *LESS2_P;
extern struct func *LEQ2_P;
extern struct func *EQUAL2_P;
extern struct func *EQ2_P;
extern struct func *PNAME_P;

/*  functions included in cu prolog  */
extern struct func *LIST, *CUNIFY;
extern struct term *NIL, *FAIL;

extern struct func *PNAMES_P;
extern struct func *PVALUE_P;
extern struct func *TYPE_P;

extern struct func *RESET_TIMER_P;
extern struct func *TIMER_P;
```

```
/*-----------------------------------------------------------------------
*                 cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 91
*       Programmed by:  Hiroshi Tsuda (tsuda@icot.or.jp)
*                       Hidetosi Sirai (sirai%sscc.chukyo-u.ac.jp)
-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------
*                 << main.c >>
*                 88.11.21 Ver.2.00       OS command
*                 90.4.1 rewrite refute, sysgred (ver.3.0)
*                 91.12  cu Prolog III
-----------------------------------------------------------------------*/

#define VERSION "III"
#define COPYRIGHT "Institute for New Generation Computer Technology, \
 Japan 1989 91\nin Cooperation with SIRAI@sscc.chukyo-u.ac.jp"
#define  MAIN   1

#include "include.h"
#include <signal.h>

struct itrace *newflist_save;

void main(argc,argv)
int argc;
char *argv[];
{
  void on_interrupt();

        prepare();              /* set flag  etc. */

        if (argc == 1) fp = stdin;      /* standard input */
        else if ((fp = fopen(argv[1],"r")) == NULL)  {
                printf(">>>  %s : can't open \n",argv[1]);
                fp = stdin;
        }
        else {
                printf(">>> open %s \n",argv[1]);
                settimer();
        }

        utop = &ustack[0];

        signal(SIGINT, on_interrupt);
        setjmp(reset);
        setjmp(unbreak_reset);

        while(1){
                f_list = NULL;
                usp = utop;
                chp = &cheap[0]; /* save constraints heap pointer */
                hp = &heap[0]; /* save user heap pointer */
                ep = &cheap[0];  /* save user stack pointer */
                newflist_save = newf_list; /* save old c.t. trace */
                prolog_execution();
        }
}

void prolog_execution()
{
        if(tty && KEVIN) putcursor();           /* print cursor */
        advance;                /* read next one char into cbuf */

        switch (cbuf) {
          case '<': {   /* read file */
            advance;
            settimer(); /* set timer */
            readfile();
            break;
          }
          case EOF: {   /* file end */
            set_eof();
            printtime();        /* print execution time */
            break;
          }
          case '%': {           /* flag statement */
            next();
            systemcommand(cbuf);
            break;
          }
          case '$': {   /* os command interpreter */
            advance;
            oscommand();
            break;
          }
          case ':':
          case '?': {   /*  question clause */
            check_recursion();
            questionclause();
            break;
          }
          case '@':{            /* modularize clause */
            advance;
            check_recursion();
            trans_routine();
            break;
          }
          case '$':     /* define new predicate */
            advance;
            defnewfunc();
            break;
          case '.':
            skipline;
            break;
          default :
            defclause(); /* definition clause */
          }
}
```

```
void on_interrupt()
{
  error("\ninterrupt\n");
}

void error_detail(t,e,s)
struct term *t;
struct pair *e;
char *s;
{
        if ((wfp != stdout) && (wfp != stderr)) fclose(wfp);
        wfp = stderr;
        Pterm(t,e);
        error(s);
}

void error(s)
char *s;
{
        if ((wfp != stdout) && (wfp != stderr)) {
                if (wfp != NULL) fclose(wfp);   /* in %w command */
                wfp = stderr;
        }
        if (!KEYIN) {
          fprintf("%s\n",cbuf);
          while (cbuf != '\n') { next(); pute(cbuf,stderr); }
          fclose(fp);
          fp = stdin;
          fprintf("\n**** error in reading file ****\n");
        }
        fprintf("\n%s\n", s);
        skipline;
        if ((sbp - sheap) >= ((SHEAPTOP - sheap) * 0.99))
                garbagecollect();
        if (fp != stdin) {
          fclose(fp);
          fp = stdin;
        }
        if (utop != &ustack[0]) {
          utop = &ustack[0];
          undo(utop);
        }
        wfp = stdout;
        if (fp == stdin) printtime();   /* print execution time */
        newf_list = newflist_save;  /* c.t. trace */
        freeheap();
        longjmp(reset, 0); /* in main() */
}

void prepare()                  /*  system preparation */
{
        tty = isatty(0);

        init_status();
        wfp = stdout;                   /* with echo back */
        lfp = NULL;                     /* no log file  */
        ECHO_MODE = FALSE;
        pstable = snew(pst_item);

        /* push_status(); */    /* save f_list, etc. */
        open_title();                   /* opening title        */
}


void open_title()               /* opening title */
{
        printf("\n\n\*****  cu - Prolog  Ver. %s  *****\n",VERSION);
        printf("Copyright: %s\n",COPYRIGHT);
        printf("\t%s mode",(is_Mcelvable ? "M_solvable" : "All Modular"));
        printf("\t(help -> %%h)\n\n");
}

void init_status()              /* initialize global vars */
{
        int i;

        Modular_mode;           /* solution flag */
        No_trace_mode;          /* trace flag */
        refute_node_count = 1; /* refute node counter*/
        MODULARMAX = Modmax_def;
        Refcount = REFMAX;

        f_list = NULL;
        newf_list = NULL;
        o_list = NULL;
        sbp = &sheap[0];
        nbp = &nheap[0];
        for(i = 0; i < HASH_SIZE; hash_list[i++] = NULL)
                ;               /* initialize hash table */
        init_heap_max();        /* cf. new.c */
        init_syspred();                 /* initialize system predicates */
        def_Modified = 0;       /* user pred not modified */
        GENSYM = 0;
        Print_Depth = 32;
}

void print_constant()           /* print constant list */
{
        struct func *f;
        int i;

        for (i = 0; i < HASH_SIZE; i++)
                for (f = hash_list[i]; f != NULL; f = f->f_link)
                if (f->f_arity == 0)
                        fprint2("%s/%x  ",f->f_name,f);
        NL;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  systemcommand()
```

```
  process cu Prolog system (%) commands
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void systemcommand(c)   /*  % command */
int c;
{
  switch(c)
        {
        case 'C':               /* change cat() function */
                set_category();
                break;
        case 'D':       /* maximum of print depth */
                readword(nbuf);
                Print_Depth = atoi(nbuf);
                break;
        case 'G':               /* garbage collection */
                garbagecollect();
                return;
        case 'H':               /* for debug */
                print_hash_table();
                                break;
        case 'L':               /*  list trace definition */
                tprint0("\n +-- list new predicate : <vars, terms>: +\n");
                Shownewfunc();
                break;
        case 'M':       /* maximum of variables in transformation */
                readword(nbuf);
                MODULARMAX = atoi(nbuf);
                if (MODULARMAX < 0)
                  MODULARMAX = Modmax_def;
                break;
        case 'N':               /* for debug */
                show_heap_max();
                break;
        case 'P':               /* Preprocess Constraints */
                readword(nbuf);
                preprocess_constraints(nbuf);
                break;
        case 'Q':               /* QUIT cu-prolog */
                quit_prolog();
                return;
        case 'R': /* system reset */
                tprint0("System initialized\n");
                prepare();
                break;
        case 'X':               /* print constant (for debug) */
                tprint0("+++++ print constants +++++\n");
                print_constant();
                break;
        case 'Y':               /* edit predicates (for debug) */
                tprint0("+++++ edit predicates +++++\n");
                edit_predicate();
                break;
        case 'a':                       /*   all modular mode   */
                tprint0("\n   ___ all modular mode  ___\n");
                Modular_mode;
                break;
        case 'c':               /* maximum of refute counter    */
                readword(nbuf);
                Refcount = atoi(nbuf);
                if (Refcount <= 0) Refcount = REFMAX; /* default */
                break;
        case 'd':                       /*  list definition */
                readword(nbuf);
                showdef(nbuf);
                break;
        case 'f':                       /*  free space  */
                tprint0("show the status of memory allocation\n");
                freeheap();
                break;
        case 'h':                       /*  help menu */
                tprintf("** %% command menu  ver.%s ****",VERSION);
                tprint0(" (prompt _:normal, $:trace, >:step)\n");
                helpmenu();
                break;
        case 'l':                       /*  log file */
                readword(nbuf);
                loghandle(nbuf);
                break;
        case 'n':               /* change genfunc name */
                readword(genname);
                break;
        case 'o':                       /*   M:Solvable mode  */
                tprint0("\n   ___ M solvable mode  ___\n");
                Msolvable_mode;
                break;
        case 'p':                       /*  set/reset spy flag */
                readword(nbuf);
                spyswitch(nbuf);
                break;
        case 's':                       /*  step trace on/off */
                stepswitch();
                break;
        case 't':                       /*  trace on */
                traceswitch();
                break;
        case 'u': /* undefined predicate handling */
                Handle_Undefined = (Handle_Undefined == TRUE) ?
                  FALSE : TRUE;
                tprintf("Undefined Predicates causes %s\n",
                        ((Handle_Undefined == TRUE) ? "ERROR" : "FAIL"));
                break;
        case 'w':                       /*  write file */
                readword(nbuf);
                filewrite(nbuf); /* save program */
                break;
        default:                /* else */
                break;
        }
        skipline;               /* skip one line */
}
```

```
void garbagecollect()           /* garbage collection */
{
        if (fp != stdin) fclose(fp);
        if ((wfp != stdout) && (wfp != stderr)) fclose(wfp);
        tprint0("------ Garbage Collection ------\n");
        strcpy(nbuf, "TEMPF.###");              /* temporary file */
        delete_tmp();                   /* delete old temp file */
        tprint0("--->");
        filewrite(nbuf);        /* save program to nbuf */
        init_status();          /* initialize shp, f_list, etc  */
        tprint0("--->");
        set_inputfile(nbuf);
        /* wfp = NULL; no echo back */
}

void edit_predicate()           /* edit predicate */
{
      tprint0("######## Garbage Collection ########\n");
        strcpy(nbuf, "TEMPF.prd");              /* temporary file */
        system("rm -f TEMPF.prd");              /* delete old temp file */
        tprint0("######## Step 1: write file \n");
        filewrite(nbuf);        /* save program */
/*      pop_status();   */ /* initialize shp, f_list, etc. */
        system("$EDITOR TEMPF.prd"); /* edit */
        tprint0("######## Step 2: read file \n");
        set_inputfile(nbuf);
}

void trans_routine()    /* modular translation routine (@ C1,C2,...,Cn.) */
{
        register struct term *t;
        struct clause *c;

        if (Is_Steptrace && isspy(MODULAR_P)) CTstep;
        else if (Is_Normaltrace && isspy(MODULAR_P)) CTnormal;
        else CTnotrace;
        v_number = 0; v_list = NULL;
        p_number = 0; pv_list = NULL;
        reread = FALSE;
        psttable->p_link = (struct pst_item *)NULL;

        t = Rterm(1200,TEMPORAL);          /* read constraints */
        if (tokentype != FULLSTOP) error("Syntax error --- . missing");
        skipline;
        NL;
        settimer();     /* set timer */
        if (is_clause(t))
                c = (struct clause *)t;
        else
                c = Nclause(t, (struct clause *)NULL, TEMPORAL);
        modular(c,v_list,v_number+p_number);
        undo(utop);     /* pop user stack  ( u : static var ) */
}
```

```
void questionclause()   /*  ? g1,g2,...gm;c1,c2...cn.   */
{
  register struct term *q;
  struct eclause *co;
  struct pair *e;
  struct node *Last_Node, *Initial_Goal;
  struct term *initial_vlist;
  struct clause *c;
  int Status, refute();

  if (Is_Steptrace && isspy(MODULAR_P)) CTstep;
  else if (Is_Normaltrace && isspy(MODULAR_P)) CTnormal;
  else CTnotrace;
  v_number = 0; v_list = NULL;
  p_number = 0; pv_list = NULL;
  psttable->p_link = (struct pst_item *)NULL;

  reread = FALSE;
  q  = Rterm(1200,TEMPORAL);            /* read goal */
  if (tokentype!=FULLSTOP) error("Syntax error --- . expected");
  skipline;             /* skip CR */

  renum_pvars((struct pstvar *)pv_list,v_number);
  e = Newv(v_number+p_number);  /* initial environments */

  settimer();
  if ((Pred(q) == CONSTRAINT_P) || (Pred(q) == CONSTRAINT2_P)) {
    c = (is_clause(Arg2(q))) ? (struct clause *)Arg2(q) :
      Nclause(Arg2(q),(struct clause *)NULL,TEMPORAL);
    co = transform((struct eclause *)NULL,c,e);
    if (co == (struct eclause *)MFAIL)
    {
      tprint0("no (unsatisfied constraints)\n");
      if (fp == stdin) printtime();
      refute_node_count = -1;
      undo(utop);
      return;
    }
    q = Arg1(q);
  }
  else {
     co = (struct eclause *)NULL;
  }
  if ((Pred(q) != QUERY1_P) && (Pred(q) != QUERY2_P))
    error("Syntax error --  Query Predicate was expected");
  f_list = NULL;                /* temp. pred list */
  c = (is_clause(Arg1(q))) ? (struct clause *)Arg1(q) :
      Nclause(Arg1(q),(struct clause *)NULL,TEMPORAL);
  Initial_Goal = Last_Node
    = Newnode(c,co,e,(struct node *)NULL,(struct node *)NULL);
  Last_BT = NULL; Last_SKIP = NULL;
  Status = DOWN;
  initial_vlist = v_list;
```

```
  /* refutation */
  while (1) {
    if (refute(Initial_Goal,Last_Node,Status) == FALSE)
      [ tprint0("no.\n"); break; ]
    else if (initial_vlist == NULL) { tprint0("true.\n"); break; }
    else if ((fp != stdin) ||
             (Panswer(Initial_Goal,initial_vlist) == FALSE)) break;
    Status = BACKTRACK;
    Last_BT = Last_Node = backtrack_node(Last_BT);
  }
  if (fp == stdin) printtime();
/*  freeheap(); */
/*  newf_list=index_newflist(newf_list,newflist_save); register itrace list */
/*  index_funclist(f_list);      tmp. pred -> hash */
  refute_node_count = -1;       /* refute node counter */
  undo(utop);                   /* pop user stack ( u : static var ) */
}


void defclause()        /* definition clause read & set */
{
  register struct term *t;
  register struct clause *c, *cstr;

  v_number = 0; v_list = NULL;
  p_number = 0; pv_list = NULL;

  reread = FALSE;
  t = Rterm(1200,STINGY);
  if (tokentype != FULLSTOP) error("Syntax error --- . was expected");
  skipline;
  if (isvar(t)) error("Syntax error --- Variables cannot be asserted");
  rename_var_names((struct var *)v_list); /* STINGY -> ETERNAL */
  if ((Pred(t) == CONSTRAINT_P) || (Pred(t) == CONSTRAINT2_P)) {
    cstr = (is_clause(Arg2(t))) ? (struct clause *)Arg2(t) :
           Nclause(Arg2(t),(struct clause *)NULL,ETERNAL);
    t = Arg1(t);
  }
  else  cstr = NULL;
  if (Pred(t) == IMF_P) {
    if (isvar(Arg1(t))) {
      tprint0(">>>>> ");
      Pterm(t,(struct pair *)NULL);
      tprint0(" <<<<<");
      error("Syntax error --- Variables cannot be asserted");
    }
    if (is_clause(Arg2(t)))
      c = Nclause(Arg1(t),(struct clause *)Arg2(t),ETERNAL);
    else
      c = Nclause(Arg1(t),
                  Nclause(Arg2(t),(struct clause *)NULL,ETERNAL), ETERNAL);
  }
  else if (is_functor(t))
    c = Nclause(t,(struct clause *)NULL,ETERNAL);
  else error("Illegal definition");

  if (p_number != 0) {
    renum_pvars((struct pstvar *)pv_list,v_number);
  }
  index_set(c, cstr, 'z');
}

void rename_var_names(v)
struct var *v;
{
  while (v != (struct var *)NULL) {
    truncate_varname(v->v_name,nbuf);
    v->v_name = malloc(nbuf,ETERNAL);
    v = v->v_link;
  }
}

void truncate_varname(n,nbuf)
char n[], nbuf[];
{
  register int i = 0;
  while ((n[i] != '\0') && (i < 7)) {
    if (n[i] == '_') break;
    nbuf[i] = n[i];
    i++;
  }
  nbuf[i] = '\0';
}

void renum_pvars(pvs,vnum)
struct pstvar *pvs;
int vnum;
{
  while (pvs != (struct pstvar *)NULL) {
    pvs->v_number += vnum;
    pvs = (struct pstvar *)pvs->v_link;
  }
}

void defnewfunc()       /* definition clause read & set */
{
  register struct term *t;
  register struct itrace *it;
  struct clause *c;

  v_number = 0; v_list = NULL;
  p_number = 0; pv_list = NULL;

  reread = FALSE;
  t = Rterm(1200,STINGY);
  if (tokentype != FULLSTOP) error("Syntax error --- . was expected");
  skipline;
  if (Pred(t) != EQSIGN_P) error("Illegal itrace definition");
```

```
  rename_var_names((struct var *)v_list); /* STINGY  > ETERNAL */

  c =  (is_clause(Arg2(t))) ? (struct clause *)Arg2(t) :
    Nclause(Arg2(t),(struct clause *)NULL,ETERNAL);

  it = snew(itrace);
  it->it_clause = Nclause(Arg1(t),c,ETERNAL);
  it->it_anumber = v_number+p_number;
  it->it_cnumber = literalnumber(c);
  it->it_link = newf_list;
  newf_list = it;
  Pred(Arg1(t))->f_iuteq = it;
}
```

```
/*------------------------------------------------------------------------
*               cu Prolog ITI (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 -91
------------------------------------------------------------------------ */
/*------------------------------------------------------------------------
*             << mainsub.c >>
*               system command etc.
------------------------------------------------------------------------ */

#include "include.h"

void putcursor () {
    if (Is_Notrace) {           /* trace off */
        tpute ('_');
    }
    else
        if (Is_Normaltrace) {   /*  trace on */
            tpute ('$');
        }
        else
            tpute ('>');        /*  step trace on */
}


void traceswitch () {
    if (Is_Normaltrace) {
        Notrace_mode;
        tprint0 ("\n !!! normal trace off !!!\n");
    }
    else {
        Normaltrace_mode;
        tprint0 ("\n !!! normal trace on !!!\n");
    }
}

void stepswitch () {
    if (Is_Steptrace) {
        Notrace_mode;
        tprint0 ("\n !!! step trace off !!!\n");
    }
    else {
        Steptrace_mode;
        tprint0 ("\n !!! step trace on !!!\n");
    }
}

int decode_pname(fname)         /* 'member/2'  -> 'member', return 2 */
char *fname;
{
        for (; *fname != '\0'; fname++)
                if (*fname == '/') {
                        *fname = '\0';
                        return(atoi(fname + 1));
                }
        return( 1);
}

void spyswitch (fname)
char   *fname;
{
    struct func *f;
    int i,arity;

    if (strcmp (fname, "*")  == 0) {
        tprint0 (" -- set all spy flag  --");
        NL;
        allspy(1);              /* set all spy flag */
        return;
    }
    if (strcmp (fname, " ") == 0) {
        tprint0 ("    reset all spy flag  --");
        NL;
        allspy(0);              /* reset all spy flag */
        return;
    }
    if (strcmp (fname, "?") == 0) /* list spyed predicates */
        tprint0 ("!!! list spyed predicates !!!\n");
        for (i=0; i < HASH_SIZE; i++)
          for (f = hash_list[i]; f != NULL; f = f -> f_link) {
            if isspy(f) tprint2 ("%s(%d)  ", f -> f_name, f  > f_arity);
        }
        NL;
        return;
    }
    if (strcmp (fname, ">") == 0) /* spy fold/unfold */
    {
            tprint0(" !!! ");
            if  isspy(MODULAR_P) tprint0("no");
            tprint0("spy fold/unfold transformation \n");
            spychange(MODULAR_P);
            spychange(INTRG_P);
            return;
    }
    arity = decode_pname(fname);
    if (!exist_fname(fname))
    {
            tprint1(" '%s' does not exist.\n", fname);
            return;
    }
    if (arity ==  1)            /* spy switch fname/?? */
    {
            for(f = hash_list[hash(fname)]; f != NULL; f = f >f_link)
                    if (streq(fname,f->f_name)) {
                            tprint0 ("!!! ");
                            if isspy(f) tprint0 ("no");
                            tprint2("spy %s/%d\n", f->f_name, f->f_arity);
                            spychange(f);
                    }
    }
```

```
    else
    {
            f = funcsearch(fname, arity);
            if (f != NULL) {
                    tprint0 (" is");
                    if isspy(f) tprint0 ("no");
                    tprint2("spy %s/%d\n", f->f_name,f->f_arity);
                    spychange(f);                   /*  switch spy flag on/off */
            }
            else
                    tprint2(" '%s/%d' does not exist.\n", fname,arity);
    }
}

void allspy (n)  /* if n == 1: set all spy flag, else resetall flag */
int     n;
{
    struct func *f;
    int i;

    if (n == 1)
    {
            for (i = 0; i < HASH_SIZE; i++)
                    for (f = hash_list[i]; f != NULL; f = f->f_link)
                            spyfun(f);
    }
    else
    {
            for (i = 0; i < HASH_SIZE; i++)
                    for (f = hash_list[i]; f != NULL; f = f->f_link)
                            nospyfun(f);
    }
}

void show_predroles(f)          /* show component */
struct func *f;
{
  int i, arity;
  struct component *cm;
  register struct component *c;

  for (i = 0, arity = f->f_arity; i < arity ; i++) {
          cm = Component[f,i];
          if (cm == NULL) {
                  tprint0("_");
          }
          else
          {
                  for (c = cm; c != NULL; c = c->c_next)
                  {
                          if (c->c_label == NULL) {
                                  tprint0("+");
                          }
                          else {
                                  tprint1("%s",c->c_label->f_name);
                          }
                          if (c->c_next == NULL) break;
                          else
                          {
                                  tprint0(".");
                          }
                  }
          }
          if (i == (arity-1)) return;
          else
          {
                  tprint0("!");
          }
  }
}

/* the number of pred names printed in one line */
#define PRED_IN_LINE 5

void show_syspred_name()
{
        int i, j = 0;
        register struct func *f;

        tprint0("   +-      <system> ----------[ +:recursive, ~:functor ]---\n");
        for (i = 0; i < HASH_SIZE; i++) /* print system predicates */
        for (f = hash_list[i]; f != NULL; f = f->f_link)
            if issystem(f) {
              tprint2 ("%s/%u", f->f_name, f->f_arity);
              if isrecursive(f) tprint0 ("+");
              if (f->def.f_sysfunc == NULL) tprint0("~");
              tputc('\t');
              if (++j >= PRED_IN_LINE) {j = 0; NL;}
            },
        NL;
}

void show_userpred_name()
{
        int i, j = 0;
        register struct func *f;

        tprint0("   +-----  <user> ------");
        tprint0("-  [ *:spy,  -:reduced, +:recursive, #:new ]---\n");
        for (i = 0; i < HASH_SIZE; i++) /* print user predicates */
                for (f = hash_list[i]; f != NULL; f = f->f_link) {
/*                      if (f->f_arity < 0) continue; */
                        if (f->def.f_sysfunc == NULL)
                                continue;       /* cut constant. */
                        if issystem(f) continue;
                        tprint2 ("%s/%u", f->f_name, f->f_arity);
                        if isspy(f) tprint0 ("*");
                        if isreduced(f) tprint0 ("-");
                        if isrecursive(f) tprint0 ("+");
                        if isnewpred(f) tprint0("#");
```

```
                        tputc('\t');
                        if (++j >= PRED_IN_LINE) { j = 0; NL; }
                }
        NL;
}

void show_syspred_status(f)
struct func *f;
{
  tprint0("--<system>--[");
  show_pred_roles(f);
  tprint1("]--<%s>---+\n",
         ((is_funcsys(f)) ? "functional" : "multi-valued"));
}

void show_pred_def(f)           /* show def of each pred */
struct func *f;
{
  void show_pred_roles();

        if (f->def.f_set == NULL) return; /* constant */
        tprint2 (" +-- ---- ( %s/%u )----", f->f_name, f->f_arity);
        if issystem(f)
        {
                show_syspred_status(f);
                return;
        }
        if is_nofuncsys(f)
        {
                tprint0(" --<system> --<multi-valued>--+\n");
                return;
        }
        if isspy(f) tprint0 ("-<spy>-");
        if isreduced(f) tprint0 ("-<reduced>-");
        if isrecursive(f) tprint0("-<recursive>-");
        if isnewpred(f) tprint0("-<new>-");
        tprint0("|"); show_pred_roles(f);
        tprint2("|--%d/%d--+\n",f->f_unitcount,f->f_setcount);
        if (f -> f_integ != NULL) {
            tprint0 ("<def> ");
            P_dclause (f -> f_integ -> it_clause,(struct pair *)NULL);
            NL; NL;
        }
        Showfunc(f);
}

void showdef (fname)            /* list definition   (%d command)   */
char   *fname;
{
    register struct func *f;
    int i,arity;

    check_recursion();
    if (streq(fname, "/")) {
        tprint0 (" +-- List all predicates ---+ \n");
        for (i = 0; i < HASH_SIZE; i++)
                for (f = hash_list[i]; f != NULL; f = f -> f_link)
                        Showfunc (f);
        return;
    }
    if (streq(fname, "*")){
        tprint0 (" +-- List predicates ---+ \n")
        for ( i=0; i < HASH_SIZE; i++)
            for (f = hash_list[i]; f != NULL; f = f -> f_link) {
            if isnotreduced(f) Showfunc (f);
        }
        return;
    }
    if (streq(fname, "?")){
            show_syspred_name();
            show_userpred_name();
            return;
    }
    if (streq(fname, "-"))
    {
            show_userpred_name();
            return;
    }
    arity = decode_pname(fname);
    if (exist_fname(fname) == NULL) {
            tprint1 ("'%s' does not exist.\n", fname);
            return;
    }
    if (arity == -1)            /* show fname/?? */
    {
            for (f =  hash_list[hash(fname)]; f != NULL; f = f->f_link)
                    if (streq(f->f_name,fname))
                            show_pred_def(f);
    }
    else
    {
            f = funcsearch(fname,arity);
            if (f != NULL)
                    show_pred_def(f);
            else
                    tprint2("'%s/%d' does not exist.\n",fname,arity);
    }
}

void loghandle (fname)          /* log file     (%l command) */
char   *fname;
{
    if (strcmp (fname, "no") == 0) {
        tprint0 ("---  log stop  ---\n");
        if (lfp != NULL)
            fclose (lfp);
        lfp = NULL;
        strcpy (logfile, "no");
    }
    else
```

```
        if ((fopen (fname, "r")) != NULL) {
            fprintf ("'%s' : already exist \n", fname);
        }
        else
            if ((fopen (fname, "w")) == NULL) {
                fprintf (" '%s' : can't open \n", fname);
            }
            else {
                if (lfp != NULL) {
                    fclose (lfp);
                    fprintf (" == %s : close ==\n", logfile);
                }
                lfp = fopen (fname, "w");
                fprintf (" log file '%s' \n", fname);
                strcpy (logfile, fname);
            }
}


void helpmenu () {              /* on-line help */
  tprint0 ("\t%%h\t: help\t\t\t%%Q : quit \n");
  tprint0 ("\t! <OS command): OS command interpreter \n");
  tprint0 ("\t%%d <predicate name>: list definition\n");
        tprint0 ("\t\t%%d* %%d/: list all   %%d?: list names   %%d : user pred\n
  tprint2 ("\t%c <file name> %c: consult file (no echo)\n", '"', '"');
  tprint1 ("\t%c <file name> ?: consult file (with echo)\n", '"');
  tprint1 ("\t%l <file name>: set log file ['%s']\n", logfile);
  tprint0 ("\t%%w <file name>: save program\n");
  tprint0 ("\t%%p <predicate name>: spy switch\n");
        tprint0 ("\t\t%%p*:spy all\t%%p-:nospy all\t%%p?:list spyed preds.\n");
  tprint1 ("\t%%t\t: normal trace switch [%s]\n",
        ((Is_Normaltrace) ? "on" : "off"));
  tprint1 ("\t%%s\t: step trace switch [%s]\n",
        ((Is_Steptrace) ? "on" : "off"));
  tprint0 ("\t%%a\t: all modular mode ");
        if (Is_Modular) tprint0(" <-now");
  tprint0 ("\n\t%%o\t: M-Solvable mode ");
        if (Is_Msolvable) tprint0(" <-now\n")
        else NL;
  tprint1 ("\t%%c <number>: max number of refutation node [%u]\n",
            Refcount);
  tprint0 ("\t%%n <name>\t: new predicate name ");
    tprint1 ("['%s']\n", genname);
  tprint0 ("\t%%L\t: list new predicate definitions\n");
  tprint0 ("\t%%f\t: show the system heap size\n");
  tprint0 ("\t%%C [Feature,type,.... ]. : set cat() functor\n");
  tprint0 ("\t\t ==> ");show_category();NL;
  tprint0 ("\t%%G\t: Garbage Collection \n");
  tprint1 ("\t%%D <number> : Max Depth of Printing, now is %d\n", Print_Depth);
  tprint1 ("\t%%u\t: Undefined Predicate Handling Switch [%s]\n",
        ((Handle_Undefined == TRUE) ? "ERROR" : "FAIL"));
  tprint1("\t%%V <number> : Max number of Variables in Transformation[%u]\n",
        MODULARMAX);
  tprint0 ("\t%%P <predicate name>: Preprocess Constraints\n");
  tprint0 ("\t\t%%P*: preprocess all\t%%P?: predicates with nonmodular\n");
  tprint0 ("\t%%R\t: system Reset \n");
}

void freeheap () {              /* print ship status */
    tprint2 ("\npermanent data area:\n\tprogram : %d%%(%d K) ",
             (int)(((shp - sheap) * 100) /(float)(SHEAPTOP - sheap)),
             ((int)(shp shcap)/1000));
    tprint2 ("string : %d%%(%d K)\n",
             (int)(((nhp - nheap) * 100) / (NHEAPTOP - nheap)),
             (int)((nhp nheap)/1000));
        show_heap_max();
}

void show_heap_max()                    /* for debug */
{
        tprint0("temporal data area\n\t ");
    tprint2 ("pst/reast : %d%%(%d K) ",
         (int)(((Cheap_Max - cheap) * 100) / (CHEAPTOP - cheap)),
         (int)((Cheap_Max - cheap)/1000));
        tprint2("heap : %d%%(%d K) ",
                 (int)(((Heap_Max - heap) * 100) / (HEAPTOP - heap)),
                 (int)((Heap_Max - heap)/1000));
        tprint2 ("stack : %d%%(%d K) ",
                 (int)(((Stack_Max - ustack) * 100)/ (STACKTOP - ustack)),
                 (int)((Stack_Max - ustack)/1000));
        tprint2 ("env. : %d%%(%d K)\n",
                 (int)(((Esp_Max - eheap) * 100)/ (ESPTOP - eheap)),
                 (int)((Esp_Max - eheap)/1000));
}

void init_heap_max()
{
        Cheap_Max = cheap;
        Stack_Max = ustack;
        Heap_Max = heap;
}

void filewrite (n) /* write program to file */
char    *n;
{
    FILE * lfpsave;
    int i;
    register struct func *f;

    if ((wfp = fopen (n, "r")) != NULL) {
        fclose (wfp);
        wfp = stdout;
        tprint1 (" %s : already exist \n", n);
        return;
    }

    if ((wfp = fopen (n, "w")) == NULL) {
        wfp = stdout;
        tprint1 (" %s : can't open \n", n);
        return;
```

```
    }

    fprintf(wfp,"%%%%%% cu Prolog predicates  %%%%%%%% \n");
    lfpsave = lfp;
    lfp = NULL;
    writenewfunc(); NL;
    for (i = 0; i< HASH_SIZE; i++)
            for (f = hash_list[i]; f != NULL; f = f->f_link)
                    if (isuser(f) && isnoreduced(f))
                            Showfunc(f);
    fclose (wfp);
    wfp = stdout;
    lfp = lfpsave;
    tprintl ("--- write to: %s ---\n", n);
    return;
}


void disp_func_def (f_from, f_to)/* show defs of f_from->f_to  */
struct  func *f_from,
             *f_to;
{
    if (f_from == NULL)
        return;
    if (f_from == f_to)
        return;
    disp_func_def (f_from->f_link, f_to);
    if (isuser (f_from) && isnoreduced (f_from))
        Showfunc (f_from);
}

void set_inputfile (n)
char   *n;                          /* file name */
{
    fp = fopen (n, "r");
    if (fp == NULL) {               /* open error */
        fp = stdin;
        tprintl ("%s  ", n);
        error ("can't open !");
    }
    else {
      tprintl ("--- open '%s'\n", n);
    }
  }

void readfile () {                  /*  "file name" or "file name?"  */
    int     i;

/*    if (!KEYIN)
        error ("file already opened"); */
    for (i = 0; ((cbuf != '"') && (cbuf != '?')); advance)
        nbuf[i++] = cbuf;
    nbuf[i] = '\0';                 /* n|| <- file name */
    if (cbuf == '?')
        ECHO_BACK = TRUE;                   /* echo back on */
    skipline;
        upush(&fp);
        utop = usp;
    set_inputfile (nbuf);       /* set file pointer */
    if (ECHO_BACK == TRUE)    wfp = stdout;     /* echo back on */
}


void set_eof () {               /* file EOF */

        clearerr(fp);   /* clear eof */
        if (tty && KEYIN){              /* from keyboard */
                error(" ");     /* EOF ('D in UNIX)  */
        }
        fclose (fp);
        if (utop != &ustack[0]) {
                utop -= 1;
                undo(utop);
                }
        else fp = stdin;
        if (wfp == NULL)
                wfp = stdout;           /* echo back on */
        tprint0 ("\n ****** end of file ******* \n");
}

/*      ---    static program analyzer  -------  --  ---   */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*    check_recursion()  check recursive/finite predicates
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int REC_to_FINITE = 1;
void check_all_unit(),rec_to_finite(),check_unitpred();
int is_body_finite();

void check_recursion()          /* check recursive user predicates */
{
        int i;
        void reset_component(),calc_component();

        if (Def_Modified == 0) return;
        for (i = 0; i < HASH_SIZE; i++)
                check_all_unit(hash_list[i]);
        REC_to_FINITE = 1;
        while(REC_to_FINITE != 0)
        {
                REC_to_FINITE = 0; /* global flag */
                rec_to_finite(); /* traverse predicates */
        }
        Def_Modified = 0;       /* def modified flag off */
        reset_component();
        calc_component();
}

void check_unitpred(f)
struct  func *f;
{
```

```
        if (issystem(f)) return;
        if ((f->f_setcount == 0) || (f->f_setcount == f->f_unitcount))
        {
                finitefun(f);
                return;
        }
        recursivefun(f);
}

void check_all_unit(f1)
struct func *f1;
{
        register struct func *f;

        if (f1 == NULL) return;
        for (f = f1; f != NULL; f = f->f_link)
        {
                check_unitpred(f);
        }
}

void rec_to_finite()            /* recursive pred -> finite pred */
{
        register int i;
        register struct func *f;

        for (i = 0; i < HASH_SIZE; i++)
                for(f = hash_list[i]; f != NULL; f = f->f_link)
                {
                        if (issystem(f)) continue;
                        if (isfinite(f)) continue;
                        if (is_body_finite(f) != 0)
                        {
                                REC_to_FINITE = 1;
                                finitefun(f);
                        }
                }
}

int is_body_finite(f)           /* if all the body is finite  */
struct func *f;
{
        register struct set *s;
        register struct clause *c;

        for (s = f->def.f_set; s != NULL; s = s->s_link)
                for (c = s->s_clause->c_link; c != NULL; c = c->c_link)
                        if (isvar(c->c_form) ||
                            isrecursive(c->c_form->type.t_func))
                          return(FALSE);
        return(TRUE);
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 *    recalc_component()  component of each argument
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
void mark_component_checked_all(), set_all_head_component(),
        set_head_component(), set_all_body_component(),set_body_component(),
        add_component_pst(), add_label(),calc_all_var();

struct component *merge_component();

int COMPONENT_CHANGED;

void calc_component()
{
        register int i;
        register struct func *f;
        do
        {
                COMPONENT_CHANGED = 0;
                for (i = 0; i < HASH_SIZE; i++)
                        for (f = hash_list[i]; f != NULL; f = f->f_link)
                                if (isuser(f))
                                        set_head_component(f);
                for (i = 0; i < HASH_SIZE; i++)
                        for (f = hash_list[i]; f != NULL; f = f->f_link)
                                if (isuser(f))
                                        set_body_component(f);
        }
        while(COMPONENT_CHANGED != 0);
        for (i = 0; i < HASH_SIZE; i++)
                for (f = hash_list[i]; f != NULL; f = f->f_link)
                        if (isuser(f))
                        {
                                component_checked(f);
                                calc_all_var(f);
                        }
}

void recalc_component()         /* calc component for  */
{
        register int i;
        register struct func *f;

        do
        {
                COMPONENT_CHANGED = 0;
                for (i = 0; i < HASH_SIZE; i++)
                        for (f = hash_list[i]; f != NULL; f = f->f_link)
                          if (isuser(f) && is_component_not_checked(f))
                                        set_head_component(f);
                for (i = 0; i < HASH_SIZE; i++)
                        for (f = hash_list[i]; f != NULL; f = f->f_link)
                          if (isuser(f) && is_component_not_checked(f))
                                        set_body_component(f);
        }
        while(COMPONENT_CHANGED != 0);
        for (i = 0; i < HASH_SIZE; i++)
                for (f = hash_list[i]; f != NULL; f = f->f_link)
```

```
                        if (isuser(f) && is_component_not_checked(f))
                        {
                                component_checked(f);
                                calc_all_var(f);
                        }
}

void calc_all_var(f)
struct func *f;
{
        register struct set *s;
        for (s = f->def.f_set; s != NULL; s = s->s_link)
                recalc_occurrence(s->s_clause, s->s_vlist);
}

void reset_component()          /* reset all Component() */
{
        int i;
        register int j;
        struct func *f;
        register struct set *s;
        register struct term *v;

        for (i = 0; i < HASH_SIZE; i++)
                for(f = hash_list[i]; f != NULL; f = f->f_link)
                if (isuser(f))
                {
                        for(j = f->f_arity - 1; j >= 0; j--)
                                Component(f,j) = NULL;
                        for(s = f->def.f_set; s != NULL; s = s->s_link)
                                for (v = s->s_vlist; v != NULL; v = vlink(v))
                                        vcomponent(v) = NULL;
                }
}

void set_head_component(f)      /* check heads of f */
struct func *f;
{
        register int i;
        register struct set *s;
        register struct term *t,*arg;

        if (f->f_arity == 0) return;
        for(s = f->def.f_set; s != NULL; s = s->s_link)
        {
                t = s->s_clause->c_form; /* head */
                for (i = f->f_arity - 1; i >= 0; i--)
                {
                      arg = Arg(t,i);
                      if (isvar(arg))
                              Component(f,i) = merge_component(Component(f,i),
                                vcomponent(arg),ETERNAL);
                      else if(is_pst(arg))
                        add_component_pst(f,i,((struct pst *)arg)->p_lists);
                      else add_label(f,i,NULL,ETERNAL); /* normal term */
                }
        }
}

void set_body_component(ff)
struct func *ff;
{
        struct set *s;
        register struct clause *c;
        register struct term *t,*arg;
        register struct func *f;
        int i;

        if (ff->f_arity == 0) return;
        for(s=ff->def.f_set; s != NULL; s = s->s_link)
        for(c=s->s_clause->c_link; c !=NULL; c= c->c_link)
        {
                t = c->c_form;
                f = Pred(t);
                for (i = f->f_arity - 1; i >= 0; i--)
                {
                        arg = Arg(t,i);
                        if (isvar(arg))
                                vcomponent(arg)=
                                        merge_component(vcomponent(arg),
                                        Component(f,i),TEMPORAL);
                }
        }
}

void add_component_pst(f,a,ec)  /* add pst ec to f/a */
struct func *f;
int a;
struct eclause *ec;
{
        register struct eclause *e;
        register struct term *value;
        register struct func *label;

        if (ec == NULL) return;
        for (e = ec; e != NULL; e = e->e_link)
        {
                label = Pred(Arg1(e->e_form));
                value = Arg2(e->e_form);
                if (isvar(value) && vcomponent(value) == NULL) continue;
                else add_label(f,a,label,ETERNAL);
        }
}

int cmp_label(l1,l2)            /* 0:equal -1:l1<l2, 1:l1>l2 */
struct func *l1,*l2;
{
        if (l1 == l2) return(0);
        else if (l1 == NULL) return( 1);
        else if (l2 == NULL) return(-1);
```

```
        else if (l1->f_number == l2->f_number) return(0);
        else if (l1->f_number < l2->f_number) return(-1);
        else return(1);
}

void add_label(f,a,l,flag)              /* add label l to f/a */
struct func *f,*l;
int a,flag;                     /* ETERNAL or TEMPORAL */
{
        register struct component *c,*cprev,*nc;
        register int cmp;

        for (cprev=NULL,c=Component(f,a); c != NULL; cprev= c, c = c->c_next)
        {
                cmp = cmp_label(l,c->c_label);
                if (cmp == 0) return;
                else if (cmp < 0) break; /* l < c_label */
        }
        MEMORY_ALLOC(nc,component,flag);
        nc->c_next = c;
        nc->c_label = l;
        if (cprev == NULL) Component(f,a) = nc;
        else cprev->c_next = nc;
        COMPONENT_CHANGED++;
}

struct component *merge_component(ca,cb,flag) /* merge cb in ca */
struct component *ca,           /* ca will be changed */
        *cb;
int flag;                       /* ETERNAL or TEMPORAL */
{
        int a;
        struct component *nc;
        if (cb == NULL) return(ca);
        if (ca != NULL)
        {
                a = cmp_label(ca->c_label,cb->c_label);
                if (a == 0) {
                        ca->c_next =
                                merge_component(ca->c_next,cb->c_next,flag);
                        return(ca);
                }
                else if (a < 0)         /* ca > cb */
                {
                        ca->c_next = merge_component(ca->c_next,cb,flag);
                        return(ca);
                }
        }
        COMPONENT_CHANGED++; /* global var */
        MEMORY_ALLOC(nc,component,flag);
        nc->c_label = cb->c_label;
        nc->c_next = merge_component(ca,cb->c_next,flag);
        return(nc);
}
```

```
int has_common_label(ec,cm)     /* TRUE/FALSE */
struct eclause *ec;
struct component *cm;
{
        register int cmp;
        if (ec == NULL || cm == NULL) return(FALSE);
        if (cm->c_label == NULL) return(TRUE); /* cm is not vacuous */
        cmp = cmp_label(Pred(Arg(ec->c_form)), cm->c_label);
        if (cmp == 0) return(TRUE);
        else if (cmp < 0) return(has_common_label(ec->c_link,cm));
        else return(has_common_label(ec,cm->c_next));
}

void oscommand () {             /* os command interpreter */
    int     i;

    for (i = 0; (cbuf != '\n'); next ())
        nbuf[i++] = cbuf;
    nbuf[i] = '\0';
    if (system (nbuf) != 0)
        fprint0 ("** OS command error **\n");
}


void delete_tmp () {            /* delete temp file */
        FILE *fptmp;

        if ((fptmp = fopen("TEMPF.###","r")) != NULL)
        {
                fclose(fptmp);
#if MSDOS == 0                  /* for UNIX */
    system ("rm -f TEMPF.###");
#else                           /* for MS-DOS */
    system ("del TEMPF.###");
#endif
        }
}

void quit_prolog () {           /* system quit */
    fprint0("\n*** Quit cu-Prolog ? (y/n) --- ");
    skipline;
    if (keyread('y')) {
            if (lfp != NULL)
                    fclose (lfp);       /* close log file */
            delete_tmp ();              /* delete temp file */
            exit (1);                   /* end */
    }
    fprint0("\n.... Return to Prolog ....\n");/* cansel */
    return;
}

void preprocess_constraints(fn)
char *fn;
{
```

```
        struct func *f;
        int arity;
        void preprocess_constr_sub(), preprocess_unit();

        if (strcmp(fn,"*") == 0) {
                fprint0("--- preprocess all predicates ---\n");
                preprocess_constr_sub(TRUE);
                fprint0("done\n");
                return;
                }
        if (strcmp(fn,"?") == 0) {
          fprint0("--- showing predicates with nonmodular constraints ---\n");
          preprocess_constr_sub(FALSE);
          NL;
          return;
          }
        arity = decode_pname(fn);
        if (! exist_fname(fn)) {
                fprint1("'%s' does not exist\n",fn);
                return;
        }
        if (arity == -1) { /* any arity */
          for (f = hash_list[hash(fn)]; f != NULL; f = f->f_link)
            if (streq(f->f_name,fn))
              preprocess_unit(f,TRUE);
        }
        else {
          f = funcsearch(fn,arity);
          if (f == NULL) {
            fprint2("'%s/%d' does not exist.\n",fn,arity);
            return;
          }
          preprocess_unit(f,TRUE);
        }
        fprint0("done\n");
}

void preprocess_constr_sub(flag)
int flag;       /* preprocess control flag */
{
        register int i;
        void preprocess_all_unit();

        for (i = 0; i < HASH_SIZE; i++)
                preprocess_all_unit(hash_list[i],flag);
}

void preprocess_all_unit(fl,flag)
struct func *fl;
int flag;
{
        register struct func *f;
        void preprocess_unit();

        if (fl == NULL) return;
        for (f = fl; f != NULL; f = f->f_link)
                preprocess_unit(f,flag);
}

void preprocess_unit(f,flag)
struct func *f;
int flag;
{
        register struct set *s;
        struct clause *c, *sol, *reduce_cstr();
        struct pair *env;
        int check_modularity(); /* cf. is_modular_clause() */

        if (issystem(f)) return;
        for (s = f->def.1_set; s != (struct set *)NULL; s = s->s_link) {
                c = s->s_constraint;
                if (c == (struct clause *)NULL) continue;
                if (check_modularity(c) == TRUE) continue;
                if (flag == FALSE) {
                        Showhorn(s->s_clause,s->s_constraint,
                                (struct pair *)NULL);  NL;
                        continue;
                }
                fprint2("%s/%d\t", f->f_name,f->f_arity);
                env = Newv(s->s_anumber);
                sol = reduce_cstr(c,s->s_vlist,s->s_anumber,env);
                if (sol == FAIL) { /* failing transformation */
                        wfp = stderr;
                        fprint1("Warning: Failing transformation in %s\n",
                                f->f_name);
                        c->c_form = FAIL;
                        c->c_link = (struct clause *)NULL;
                        Showhorn(s->s_clause,s->s_constraint,
                                (struct pair *)NULL);NL;
                        wfp = stdout;
                        continue;
                }
                up_init();
                s->s_constraint = (struct clause *)termset(sol,env,ETERNAL);
                s->s_clause = (struct clause *)
                        termset(s->s_clause,env,ETERNAL);
                up_restore();
                if (p_number != 0) {
                        renum_pvars((struct pstvar *)pv_list,v_number);
                }
                s->s_anumber = (unsigned short int)(v_number*p_number);
                s->s_vlist = v_list;
        }
}

int check_modularity(cst)       /* cf. is_modular_clause */
struct clause *cst;
{
        register struct clause *c;
        register struct term *t;
```

```
        for (c= cst; c != NULL; c = c->c_link)
        {
                t = c->c_form;
                if (Pred(t) == EQ2_P &&
                    (! is_modular_literal(t))) return(FALSE);
        }
        return(TRUE);
}


#define NOREDUCED_CLAUSE (struct clause *)1

struct clause *reduce_cstr(cst,vlist,anum,env)
struct clause *cst;
struct term *vlist;
int anum;
struct pair *env;
{
        struct clause *nc,*reduce_substitute();
        int reduced = 0;

        nc = reduce_substitute(cst,env); /* reduce x=y */
        if (nc == MFAIL) return(MFAIL);
        else if (nc == NOREDUCED_CLAUSE) /* no reduction */
                return(startmodular(cst,vlist,anum));
        else if (nc == NULL) return(NULL);
        else
        {
                up_init();
                nc = (struct clause *)termset(nc,env,MEMTP);
                up_restore();
                if (p_number != 0) {
                        renum_pvars((struct pstvar *)pv_list,v_number);
                }
                return(startmodular(nc,v_list,v_number + p_number));
        }
}


struct clause *reduce_substitute(cst,e)/* preprocess constraint */
struct clause *cst;
struct pair *e;
{
        register struct clause *c;
        struct clause *compress_clause();
        int reduced = 0;
        struct term *t;

        for (c = cst; c != (struct clause *)NULL; c = c->c_link)
        {
                t = c->c_form;
                if (Pred(t) == EQ2_P) /* '=' */
                {
                        reduced = 1;
                        c->c_form = NULL;
                        if (unify(Arg(t,0),e,Arg(t,1),e,0) == FALSE)
                                return(MFAIL);
                }
        }
        if (reduced == 0) return(NOREDUCED_CLAUSE);
        else return(compress_clause(cst));
}

struct clause *compress_clause(cst) /* cut c (c->c_form == NULL) */
struct clause *cst;
{
        if (cst == (struct clause *)NULL) return(NULL);
        if (cst->c_form != NULL) return(compress_clause(cst->c_link));
        else
        {
                cst->c_link = compress_clause(cst->c_link);
                return(cst);
        }
}


/************* print CPU time ****************************/
#if SUN4 == 1 /* print CPU time */
#include <sys/time.h>
long    TIMESAVE;               /* system time saver (time_t = long) */

void printtime () {             /* print CPU time from the previous settime()*/
  long clock();
  tprint2 ("CPU time = %.3lf sec (Constraints Handling = %.3lf sec)\n",
           ((double)(clock() - TIMESAVE))/ 1000000.0,
           ((double)CONSTRAINT_HANDLING_TIME/1000000.0));
}

void settimer () {              /* set clock     */
  long clock();
  TIMESAVE = clock();
  CONSTRAINT_HANDLING_TIME = 0L;
}
#else
#if CPUTIME == 0                /* do not print CPU time */
void printtime () {
}
void settimer () {
}

#else  /* BSD */
#include <sys/types.h>
#include <sys/times.h>
time_t    TIMESAVE;             /* system time saver (time_t = long) */
struct tms TIMES;               /* cf. times() */

/*
* Structure returned by old times() interface
*     struct tms {
```

```
*           time_t     tms_utime;           user time
*           time_t     tms_stime;           system time
*           time_t     tms_cutime;          user time, children
*           time_t     tms_cstime;          system time, children
*       };
*/

void printtime () {                 /* print CPU time from the previous settime()*/
        time_t  ttemp;

        times(&TIMES);              /* get time */
        ttemp = TIMES.tms_stime + TIMES.tms_utime;
        fprintf ("CPU time = %.3lf sec (Constraints Handling = %.3lf sec)\n",
             (ttemp   TIMESAVE) / CPUTIME.0,
             (CONSTRAINT_HANDLING_TIME / CPUTIME.0));
}

void settimer() {                   /* set clock      */
        times(&TIMES);
        TIMESAVE = TIMES.tms_stime + TIMES.tms_utime;
        CONSTRAINT_HANDLING_TIME = 0L;
}

#endif
#endif
```

```
/*-----------------------------------------------------------------------
*               cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                        1989 - 91
-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------
*               <<<< new.c >>>>
*               memory management
-----------------------------------------------------------------------*/

#define DEBUG  0                /* if debug 1 else 0 */
#define NEW    1

#include "include.h"

/* struct allocation macro   int a:arity  */
int TERM_SIZE = (sizeof(struct term) / sizeof(int));
int FUNC_SIZE = (sizeof(struct func) / sizeof(int));
int POINTER_SIZE = (sizeof(struct term *) / sizeof(int));

#if SUN4 == 1
#define Termalloc(a)    (struct term *)salloc(TERM_SIZE + a * POINTER_SIZE)
#define tempterm(a)     (struct term *)alloc(TERM_SIZE + a * POINTER_SIZE)
#define mediterm(a)     (struct term *)challoc(TERM_SIZE + a * POINTER_SIZE)
#define funcalloc(a)    (struct func *)salloc(FUNC_SIZE + a * POINTER_SIZE)
#else
#define Termalloc(a)  (struct term *)salloc(TERM_SIZE + (a-1) * POINTER_SIZE)
#define tempterm(a)   (struct term *)alloc(TERM_SIZE + (a-1) * POINTER_SIZE)
#define mediterm(a)   (struct term *)challoc(TERM_SIZE + (a-1) * POINTER_SIZE)
#define funcalloc(a)  (struct func *)salloc(FUNC_SIZE + (a-1) * POINTER_SIZE)
#endif

void print_hash_table()         /* for debug */
{
        register int i;
        register struct func *f;

        for (i = 0; i < HASH_SIZE; i++)
        {
                tprint1("[%d]",i);
                for (f = hash_list[i]; f != NULL; f = f->f_link)
                        tprint2("%s/%d  ",f->f_name,f->f_arity)
                NL;
        }
}

int hash(fname)
char *fname;
{
        register int h = 0, factor;

        for (factor = strlen(fname) + 1; *fname != '\0'; fname++, factor--)
                h += ((*fname) * factor);
        if (h < 0) return(0);
        return(h % HASH_SIZE);
}

int *salloc(n)          /* system heap allocation */
register int n;
{
        register int *p;
#if DEBUG == 1
        if (shp < SHEAPBOTTOM)
                error("system heap underflow");
#endif
        p = shp;
        shp += n;
        if (shp < SHEAPTOP)
                return(p);
        else
                error("system heap overflow");
}

int *alloc(n)   /* user heap allocation */
register int n;
{
        register int *p;
        /*      hp */

        p = hp;
        hp += n;
#if DEBUG == 1
        if (hp < HEAPBOTTOM){
                sprintf(nbuf,"hp = %d  : user heap underflow",hp);
                error(nbuf);
        }
#endif
        if (hp > Heap_Max) Heap_Max = hp;
        if (hp < HEAPTOP)
                return(p);
        else
                error("user heap overflow");
}

int *challoc(n) /* constraints/pst heap allocation */
register int n;
{
        register int *p;

        p = chp;
        chp += n;
#if DEBUG == 1
        if (chp < CHEAPBOTTOM){
                sprintf(nbuf,"chp = %d  : constraints heap underflow",chp);
                error(nbuf);
        }
#endif
        if (chp > Cheap_Max) Cheap_Max = chp;
        if (chp < CHEAPTOP)
                return(p);
```

```
        else
                error("constraints heap overflow");
}

struct pair *palloc(n)  /* environment stack allocation */
register int n;
{
        register struct pair *p;

        p = ep;
        ep += n;
#if DEBUG == 1
        if (ep < eheap){
                sprintf(nbuf,"ep = %d  : environment stack underflow",ep);
                error(nbuf);
        }
#endif
        if (ep > Esp_Max) Esp_Max = ep;
        if (ep < ESPTOP)
                return(p);
        else
                error("environment stack overflow");
}


char *nalloc(n,flag)    /* name string heap allocation */
register char *n;
int flag;
{
        register char *p;
        register int q;
        register struct func *f;

        if ((&nheap[0] <= n) && (n <= nhp)) return(n);
        if ((f = exist_fname(n)) != NULL)  return(f->f_name);

        /* - nhp */
        switch (flag) {
          case ETERNAL:
          case MEDIUM:
                q = strlen(n)+1;
                p = nhp;
                nhp += q;
                if(nhp > NHEAPTOP) error("name heap overflow");
                break;
          default : /* TEMPORAL or STINGY */
                q = strlen(n)+4;
                p = (char *)alloc(q / sizeof(int));
              }
        strcpy(p,n);
        return(p);
}


struct term *Num(nbuf,flag) /* make number */
char *nbuf;
int flag;
{
        register struct term *n;
        float x;
        double atof();

        MEMORY_ALLOC(n,term,flag);
        n->type.ident = ATOMIC_TYPE;
        sscanf(nbuf,"%f",&x);
        n->tag.n_value = x;
        if (x == ((float)((int)x))) n->t_arity = INT_NUM;
        else n->t_arity = FLOAT_NUM;
        return(n);
}

struct term *Num_val(x,flag)    /* make a term representing x */
register float x;
int flag;
{
        register struct term *n;

        MEMORY_ALLOC(n,term,flag);
        n->type.ident = ATOMIC_TYPE;
        if (x == ((float)((int)x))) n->t_arity = INT_NUM;
        else n->t_arity = FLOAT_NUM;
        n->tag.n_value = x;
        return(n);
}

struct term *Nstr(x, flag)      /* make a term representing x */
char *x;
int flag;
{
  register struct term *s;

  MEMORY_ALLOC(s,term,flag);
  s->type.ident = ATOMIC_TYPE;
  s->t_arity = STRING;
  if (flag==STINGY) flag=ETERNAL;
  s->tag.s_value = nalloc(x,flag);
  return(s);
}

struct pst *Npst(flag)
int flag;
{
  register struct pst *p;
  struct pstvar *pv;

  MEMORY_ALLOC(p,pst,flag);
  p->type = PST_TYPE;

  MEMORY_ALLOC(pv,pstvar,flag);
  pv->v_type = VAR_PST_TYPE;
```

```
  pv->v_name = vname(Anonymous_var);
  pv->v_number = p_number++;
  pv->v_link = pv_list;
  pv->old_var = NULL;
  p->p_var = pv_list = (struct term *)pv;

  p->p_lists = (struct eclause *)NULL;
  return(p);
}

struct eclause *Neclause(val,env,tail,flag)
struct term *val;
struct pair *env;
struct eclause *tail;
int flag;
{
  struct eclause *obj;

  MEMORY_ALLOC(obj,eclause,flag);
  obj->e_type = ECLAUSE_TYPE;
  obj->e_env = env;
  obj->e_form = val;
  obj->e_link = tail;
  return(obj);
}

struct term *Npst_item(p,pobj,next)
struct pair *p;
struct eclause *pobj;
struct pst_item *next;
{
  struct pst_item *t;
  t = cnew(pst_item);
  t->p_var = p;
  t->p_lists = pobj;
  t->p_link = next;
  return((struct term *)t);
}

struct pst_item *find_pstitem(t,e)
struct term *t;
struct pair *e;
{
  register struct pair *p;
  register struct pst_item *table = psttable->p_link;

  if (e==(struct pair *)NULL)
    return((struct pst_item *)NULL);

  t = ((struct pst *)t)->p_var;
  down(p,t,e);
  while (table != (struct pst_item *)NULL) {
    if (table->p_var == p) return(table);
    table = table->p_link;
  }
  return(table);
}

struct term *Nfile(x)
FILE *x;
{
register struct term *t;

  t = cnew(term);
  t->type.ident = ATOMIC_TYPE;
  t->t_arity = FILE_POINTER;
  t->tag.f_value = x;
  return(t);
}


struct term *Nvar(nbuf,flag)    /* make new var */
char *nbuf;
int flag;
{
        register struct var *v;
        /* i     nbuf                    */
        /*       v_number, v_list, obp   */

       MEMORY_ALLOC(v,var,flag);
        v->v_type = VAR_GLOBAL_TYPE;
        v->v_number = v_number++;
        v->v_name = (nbuf--Anonymous_VarName) ? Anonymous_VarName :
                    salloc(nbuf,flag);
        v->v_link = (struct var *)v_list;
        v_list = (struct term *)v;
        v->v_constraint = NULL; /* for CMIC 89.6.16 */
        v->v_occurrence = 1;    /* var occurrence */
        return(v_list);
}

struct term *varsearch(varname) /* search varname in v_list */
char *varname;
{
  register struct term *v;
  for (v = v_list; v != NULL; v = vlink(v))
    if (streq(varname, vname(v))) {
            ((struct var *)v)->v_occurrence++;
            return(v);
    }
  return(NULL);
}

void reset_voccurrence(v)               /* all v_occurrence = 0 */
register struct term *v;
{
  while (v != NULL) {
    ((struct var *)v)->v_head_occur = ((struct var *)v)->v_occurrence;
    ((struct var *)v)->v_occurrence = 0;
    v = vlink(v);
```

```
  }
}

void recale_voccur_sub(t,v)
struct term *t,*v;
{
  if (t == NULL) return;
  switch (t->type.ident) {
    case VAR_VOID_TYPE:  /* var */
    case VAR_GLOBAL_TYPE:
        ((struct var *)t)->v_occurrence++;
    case VAR_PST_TYPE:
    case ATOMIC_TYPE:
        return;
    case PST_TYPE:
        recale_voccur_sub((struct term *)((struct pst *)t)->p_lists,v);
        return;
    case ECLAUSE_TYPE: /* for pst objects */
        recale_voccur_sub(Arg2(((struct eclause *)t)->e_form),v);
        recale_voccur_sub((struct term *)((struct eclause *)t)->e_link,v);
        return;
    case CLAUSE_TYPE:
    case LIST_TYPE:
        recale_voccur_sub(head_of_list(t),v);
        recale_voccur_sub(tail_of_list(t),v);
    case CONST_LIST_TYPE:
        return;
    default:
          {
            register int i, j=Pred(t)->f_arity;
            for (i = 0; i < j; i++)
              recale_voccur_sub(Arg(t,i),v);
          }
        }
}

void decrement_vacuous(t)       /* decrement voccurrence of vacuous position */
struct term *t;
{
        register struct func *f;
        register int i;
        register struct term *arg;

        for (f = Pred(t),i = f->f_arity - 1; i >= 0; i-- )
        {
                arg = Arg(t,i);
                if (isvar(arg) && Component(f,i) == NULL)
                        vdecrement(arg);
        }
}

void recale_voccurrence(cl,v)   /* cl ==  H :- C. */
struct eclause *cl;
struct term *v;
{
        register struct eclause *c;

        if (cl == NULL) return;
        if (v == NULL) return;
        reset_voccurrence(v);   /* all voccurrence=0 */
        recale_voccur_sub(cl->e_form,v); /* check head */
        reset_voccurrence(v);   /* body var -> head var */
        for (c = cl->e_link; c != NULL; c = c->e_link) /* check body */
                recale_voccur_sub(c->e_form,v);
        for (c = cl->e_link; c != NULL; c = c->e_link) /* vacuous vars */
                decrement_vacuous(c->e_form);
}


struct func *exist_fname(fname) /* search predicate name */
char *fname;
{
        register struct func *f;

        for (f = hash_list[hash(fname)]; f != NULL; f = f->f_link)
                if (streq(fname,f->f_name)) return(f);
        return(NULL);
}

struct func *Predicate(fname, arity) /* search fname/arity */
char *fname;                         /* if not exist, make Nfunc */
int arity;
{
        register struct func *f;

        f = funcsearch(fname,arity);
        if (f == NULL) return(Nfunc(USERFUN,fname,arity));
        else return(f);
}

struct func *funcsearch(fname, arity) /* search fname/arity */
char *fname;
int arity;
{
        register struct func *f;
        register int compare;

        for (f = hash_list[hash(fname)]; f != NULL; f = f->f_link)
        {
                if ((compare = strcmp(fname,f->f_name)) > 0)
                        return(NULL);
                if ((compare == 0) && (f->f_arity == arity))
                        return(f);
        }
        return(NULL);
}

int pred_compare(f1,f2) /* pred compare  1 <, 0: =, 1 > */
struct func *f1,*f2;
{
```

```
        register int cmp;

        cmp = strcmp(f1->f_name,f2->f_name);
        if (cmp != 0) return(cmp);
        return(f2->f_arity - f1->f_arity);
}

void index_func(fnew)   /* store predicate fnew into hash table */
struct func *fnew;
{
        struct func *flist;
        register struct func *f, *flast;
        int i = hash(fnew->f_name);

        flist = hash_list[i];
        if ((flist == NULL) || (pred_compare(fnew,flist) > 0))
        {
                hash_list[i] = fnew;
                fnew->f_link = flist;
                return;
        }
        for (flast=flist, f=flist->f_link; f != NULL; flast = f, f = f->f_link)
        {
                i = pred_compare(fnew,f);
                if (i > 0) break;
                if (i==0) {
/*        sprintf(nbuf,"function `%s' is already used",fnew->f_name);
          error(nbuf); */
                  return;
                }
        }
        flast->f_link = fnew;
        fnew->f_link = f;
        return;
}

void index_funclist(flist)  /* register TEMPORAL funcs */
struct func *flist;
{
  register struct func *f,*fnext;

  for (f = flist; f != NULL;)
    {
      fnext = f->f_link;
      if (isuser(f)) index_func(f);
      f = fnext;
    }
}

struct itrace *index_newflist(fl,it)
struct itrace *fl,*it;
{
  register struct itrace *t, *top, *s, *temp;
  if (fl==it) return(fl);
  top = temp = new(itrace);
  for (t=fl; t != it; t=t->it_link) {
    if (in_sheap(t)) {
      temp->it_link = t;
      temp = t;
    }
    else {
      temp->it_link = s = new(itrace);
      s->it_anumber = t->it_anumber;
      s->it_cnumber = t->it_cnumber;
      temp = s;
    }
    temp->it_clause = up_itrace_clause(t->it_clause,t->it_anumber);
  }
  temp->it_link=it;
  return(top->it_link);
}

struct operator *op_search(fname,otype)
char *fname;
register int otype;
{
  register struct operator *o;
  register struct func *f;

  f = (otype != INFIX) ? funcsearch(fname,1) : funcsearch(fname,2);
  if (f == NULL) return(NULL);
  for (o=o_list; o != NULL; o=o->o_link)
    if ((f == o->o_func) && (otype == (o->o_type & INFIX)))
      return(o);
  return(NULL);
}

int FNUMBER = 0; /* function number  (used in term compare ) */

struct func *Nfunc(ftype, n, a) /* make new function */
int ftype;                      /* predicate type in include.h */
char *n;                        /* functor name */
int a;                          /* arity */
{
        register struct func *f, *ff;
        int i;

        /*      FNUMBER, const_list,f_list, shp  */
        f = funcalloc(a);
        f->f_arity = a;
        f->f_name = palloc(n,EXTERNAL);
        f->f_setcount = 0;      /* number of def clauses */
        f->f_unitcount = 0;     /* number of unit clauses */
        f->def.f_set = NULL;
        f->f_number = FNUMBER++;
        f->f_integ = NULL;
        if (ftype != TEMPFUN)
        { f->f_mark = (a > 0) ? (ftype | VACUITY_NOCHECK) : ftype;
          index_func(f);
```

```
        }
        else
          { f->f_mark = (a > 0) ? (USERFUN | VARUTIL VARCHECK) :
              USERFUN;
            f->f_list;
            f->f_list = f;
            f->f_link = ff;
          }
        for (i = 0; i < a; i++) Component(f,i) NULL;
        return(f);
}


struct term *Nterm(n,flag)
int n,          /* arity */
int flag;
{
        struct term *t; /* alloc term in cheap */

/*      if (n > VMAX) error("too many arguments"); */
        switch (flag) {
          case TEMPORAL:
            t = temptem(n); break;
          case PERMANL:
          case STICKY:
            t = Termalloc(n); break;
          default: /* MEDIUM */
            t = mediterm(n);
          }
        t->t_arity = n;
        return(t);
}

struct pair *Newv(n)    /*  new environment for n vars    */
register int n;
{
        register struct pair *p;
        register int i;

        p = calloc(n);

        for(i = 0;  i < n;  i++)
        {
                p[i].p_body = NULL;
                p[i].p_env = NULL;
        }
        return(p);
}

struct clause *Nlist(head,body,flag)
struct term *head;
struct clause *body;
int flag;
{
  register struct clause *c;

  MEMORY_ALLOC(c,clause,flag);
  c->c_type =  (nevar(head) &&
                ((body == (struct clause *)NIL) ||
                 (body->c_type == CONST_LIST_TYPE))) ?
    CONST_LIST_TYPE : LIST_TYPE;
  c->c_form = head;
  c->c_link = body;
  return(c);
}

struct clause *Nclause(head,body,flag)
struct term *head;
struct clause *body;
int flag;
{
  register struct clause *c;

  MEMORY_ALLOC(c,clause,flag);
  c->c_type = CLAUSE_TYPE;
  c->c_form = head;
  c->c_link = body;
  return(c);
}

struct set *setconcat(slist, s)         /*  add s to the end of slist */
struct set *slist,*s;
{
        register struct set *ss;

        if (slist == NULL) return(s);

        for(ss = slist; ss->s_link != NULL; ss = ss->s_link) ;
        ss->s_link = s;
        return(slist);
}

int literalnumber(c)            /* number of literals in c */
register struct clause *c;
{
        register int i;

        for (i = 0; c != NULL; c = c->c_link, i++);
        return(i);
}

void index_set(chead,con,flag)
struct clause *chead, *con;
char flag;
{
  struct set *s;

  if (issystem(Pred(chead->c_form))) {
    sprintf(nbuf,"Caution!! : %s is a system predicate.\n",
    Pred(chead->c_form)->f_name);
```

```
    error(nbuf);
  }

  s = snew(set);
  s->s_clause = chead;

  recalc_occurrence(chead, v_list);
  s->s_vlist = v_list;
  s->s_anumber = v_number+p_number;
  s->s_constraint = con;
  s->s_link = NULL;

  add_set(s,flag);
}


void add_set(s,flag)            /* add definition s to the end */
struct set *s;
char flag;                      /* 'a' or 'z' */
{
        register struct func *f = s->s_clause->c_form->type.t_func;
        struct set *setconcat();

        /* check set_bodynumber */
        s->s_bodynumber = literalnumber(s->s_clause->c_link);

        if (flag == 'z') f->def.f_set = setconcat(f->def.f_set, s);
        else
        {
                s->s_link = f->def.f_set;
                f->def.f_set = s;
        }
        f->f_setcount++;
        if is_unitclause(s) f->f_unitcount++;
/*      add_f_chind(s->s_clause->c_form); *//* calc f_chind[] */
        Def_Modified = 1;       /* def modified flag (global v.) */
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  user stack operations:
  upush(), undo()
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void upush(p)
register int *p;
{
        /* -     usp     */

        if (p == NULL) return;
        usp->u_addr = p;
        (usp++)->u_val = *p;

/* for MS-DOS large model *//*
#if MSDOS == 2
        usp->u_addr = p + 1;
        (usp++)->u_val = *(p + 1);
#endif
*/
#if DEBUG == 1
        if (p < HEAPBOTTOM || p > HEAPTOP)
                error("out of range in upush");
        if (usp < STACKBOTTOM)
                error("user stack underflow");
#endif
        if (usp > Stack_Max) Stack_Max = usp;
        if (usp > STACKTOP)
                error("user stack overflow");
}

void undo(u)
register struct ustack *u;
{
        /* -     usp     */
#if DEBUG == 1
        if (u < STACKBOTTOM)
                error("user stack underpop");
#endif
        if (u > usp)
                error("user stack overpop");
        while(usp > u) {
                --usp;
#if DEBUG == 1
                if (usp->u_addr < HEAPBOTTOM || usp->u_addr > HEAPTOP)
                  fprintf(stderr, " over heap (undo)%x/%x\n",usp,STACKBOTTOM);
#endif
                if (usp->u_addr == NULL) return;
                *(usp->u_addr) = usp->u_val;
        }
}
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*               cu Prolog III (Constraint Unification Prolog)
*   Copyright : Institute for New Generation Computer Technology,Japan
*                           1989 '91
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*               <<<< read.c >>>>
*               input routines
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

#include "include.h"

/* Classification of Characters: */
#define BL  001  /* blank */
#define UC  002  /* Upper Character */
#define LC  003  /* Lower Character */
#define UL  004  /* Underline */
#define N   005  /* Numeric */
#define SG  006  /* sign, ! */
#define SP  007  /* special character */
#define Q   010  /* single/double quote */
#define CT  011  /* Cut */
#define CM  012  /* comment character */
#define BR  013  /* Brackets, Comma */
#define CO  014  /* Constraint Marker */

#define kanzi(CH)       (CH > 128)  /* for EUC */
#define alpha   (kanzi(cbuf) || ((UC <= char_type[cbuf]) && \
                                 (char_type[cbuf] <= N)))

/* special characters are #$%&*! /:<->?@\^|~ */
#define specialchar(C)  ((! kanzi(C)) && ((char_type[C] == SG) || \
                                          (char_type[C] == SP)))
#define delimitchar(C)  ((! kanzi(C)) && ((char_type[C] >= BR) || (C == '.')))
#define bracket(C)      ((! kanzi(C)) && (char_type[C] == BR))

#define quotesign       ((! kanzi(cbuf)) && (char_type[cbuf] == Q))
#define white           ((! kanzi(cbuf)) && (char_type[cbuf] == BL))

#define numeric(X)   ((X == '-') || ((! kanzi(X)) && (char_type[X]==N)))
#define isdigit(X)   ((! kanzi(X)) && (char_type[X]==N))
#define isxdigit(X)  ((! kanzi(X)) && ((char_type[X]==N) || \
                 ((char_type[X] == LC) && ('a' <= X) && (X <= 'f')) || \
                 ((char_type[X] == UC) && ('A' <= X) && (X <= 'F'))))
#define is_varname(X)  ((X == '_') || ((! kanzi(X)) && (char_type[X]==UC)))

#define notconst_list(L) (((struct clause *)L)->c_type != CONST_LIST_TYPE)
#define isconst_list(L) (((struct clause *)L)->c_type == CONST_LIST_TYPE)

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  RTerm(num,flag)
        general read routine entry
    num:  operator strength
    flag: heap mode (TERMSEGMENT,STRINGSEGMENT,...)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct term *RTerm(n,flag)
int n,flag;
{
  int m = 0;
  struct term *RTerm_half(),*RTerm_leftover();
  struct term *t = RTerm_half(n,flag,&m);

  t = RTerm_leftover(n,m,flag,t);
  if ((is_clause(t)) && (((struct clause *)t)->c_link == NULL))
    return(((struct clause *)t)->c_term);
  return(t);
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  next()
    get next char into cbuf (global char register)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void next()
{
        if (feof(fp))
        {
                clearerr(fp);
                cbuf = EOF;
                return;
        }
        if (ferror(fp))
        {
                clearerr(fp);
                error("input error !");
        }
        cbuf = getc(fp);
        if (lfp) putc(cbuf,lfp);        /* log file */
        if (ECHO_BACK) putc(cbuf, wfp);
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  int check(c)
  check whether next input is c
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int check(c)            /*  check if c is cbuf  */
char c;
{
        if(cbuf == c) {
                advance;
                return(TRUE);
        } else
                return(FALSE);
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  int keyread(a)
    check whether user's input is 'a....\n'
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int keyread(a)
char a;
```

```
{
        int c,c1;

        c = c1 = getchar();
        while (c1 != '\n') c1 = getchar();
        if (lfp) fprintf(lfp,"\n");
        if (c == a) return(TRUE);
        else return(FALSE);
}

void adv()              /*  skip white and set the next char into cbuf  */
{
        while(white)
                next();
}

int skip(c)             /*      skip c  */
char c;
{
        if(cbuf != c){
          wfp = stderr;
          fprint2("\n '%c' <-> '%c'", cbuf, c);
          error(" illegal character ");
        }
        advance;
        return(cbuf);
}

int alldigit(c)
register int *c;
{
      .   if (! numeric(*c)) return(FALSE);
          for( c++ ; *c!='\0' ; c++ )
            if (! isdigit(*c)) return(FALSE);
          return(TRUE);
}

void read_hexa(i)
register int i;
{
  for(i=0; isxdigit(cbuf); next())
    if (i < NAME_MAX) nbuf[i++]=cbuf;
  nbuf[i]='\0';
}


void read_digits(i)
register int i;
{
  for( ; isdigit(cbuf); next())
    if (i < NAME_MAX) nbuf[i++] = cbuf;
  if (cbuf=='.')
    do
      { nbuf[i++] = cbuf;
        next();
      } while((isdigit(cbuf)) && (i < NAME_MAX));
  if ((cbuf == 'e') || (cbuf == 'E')) {
    nbuf[i++]   'E';
    next();
    if ((cbuf == '-') || isdigit(cbuf))
      do
        { nbuf[i++]   cbuf;
          next();
        } while((isdigit(cbuf)) && (i < NAME_MAX));
  }
  nbuf[i]='\0';
}

void read_comments()
{
  while (1) {
    next();
    if (cbuf == '*') {
      next();
      if (cbuf == '/') {advance; return;}
    }
  }
}

void read_specharc(i)
register int i;
{
  while (specialchar(cbuf) && (i < NAME_MAX)) {
    nbuf[i++] = cbuf;
    next();
  }
  nbuf[i]='\0';
}

int Rtoken()    /* read name into nbuf[]  */
{
  if (reread) {
    reread = FALSE;
    return(tokentype);
  }

  adv();
  if (cbuf==EOF) {
    set_eof();
    error("unexpected End of File");
  }
  if (bracket(cbuf))   /* ()[]{}|, */
    {
      if (cbuf==',') return(tokentype = COMMA);
      nbuf[0] = cbuf;
      nbuf[1] = '\0';
      return(tokentype = BRACKET);
    }
  if (! kanzi(cbuf)) {
   /* if (char_type[cbuf] == CO) return(tokentype = CONST_MARK); */
```

```
      if (char_type[cbuf] == CM) {
        skipline;
        return(Rtoken());
      }
  }

  tokentype = NAME;
  if (isdigit(cbuf))
    {
      read_digits(0);
      return(tokentype = NUMBER);
    }

  if (alpha) {
    register int i = 0;
    if (is_varname(cbuf)) tokentype = VARNAME;
    while(alpha && (i < NAME_MAX)){
      if (kanzi(cbuf)) {
        nbuf[i++] = cbuf;
        next();
      }
      nbuf[i++] = cbuf;
      next();
    }
    nbuf[i]='\0';
    return(tokentype);
  }

  if (quotesign) {
    register char temp = cbuf;
    register int i;
    if (temp == '\"') tokentype = STRING;
    next();
    for (i = 0; i < NAME_MAX; next()) {
      if (cbuf != temp) nbuf[i++]=cbuf;
      else {
        next();
        if (cbuf == temp) nbuf[i++]=cbuf;
        else break;
      }
    }
    if (i >= NAME_MAX) {
      nbuf[i]='\0';
      wfp = stderr;
      sprintf(nbuf,">>> %s <<<",nbuf);
      error("too long string/name");
    }
    nbuf[i]='\0';
    return(tokentype);
  }

  if (cbuf == '!') {
    nbuf[0] = cbuf;
    nbuf[1] = '\0';
    next();
    return(tokentype);
  }

  if (char_type[cbuf] == OP) {
    nbuf[0] = cbuf;
    nbuf[1] = '\0';
    next();
    return(tokentype);
  }

  if (specialchar(cbuf)) {
    register int i = 0;

    nbuf[i++] = cbuf;
    next();

    switch (nbuf[0]) {
    case '-':
      if (isdigit(cbuf)) {
        tokentype = NUMBER;
        read_digits(i);
      }
      else read_spechar(i);
      break;
    case '/':
      if (cbuf == '*') {
        read_comments();
        return(Rtoken());
      }
      else read_spechar(i);
      break;
    case '#':
      if (isxdigit(cbuf)) {
        tokentype = FILE_TYPE;
        read_hexa(i);
      }
      else read_spechar(i);
      break;
    case '.':
      if (white) tokentype = FULLSTOP;
      else read_spechar(i);
      break;
    default:
      read_spechar(i);
    }
  }
  else {
    sprintf(nbuf,"Illegal Character:>>> %c <<<",cbuf);
    error(nbuf);
  }

  return(tokentype);
}

struct Term *Rlist(flag)        /* read list */
```

```
int flag;
{
        register struct term *v;
        register struct clause *c;
        int ac;

        advance;        /* skip | */
        if (cbuf == '|') {      /*  ||  */
                return(NIL);
              }
        /* read the first argument */
        c = Nlist(Rterm(999,flag),(struct clause *)NIL,flag);
        ac = c->c_type;
        switch (Rtoken()) {
          case COMMA:
                c->c_link = (struct clause *)Rlist(flag);
                if (notconst_list(c->c_link)) c->c_type = LIST_TYPE;
                return((struct term *)c);
          case BRACKET:
               if (nbuf[0]==']') {
                 advance;
                 v = Rterm(999,flag);
                 c->c_link = (struct clause *)v;
                 if (isvar(v) || notconst_list(v))
                   c->c_type = LIST_TYPE;
                 return((struct term *)c);
               }
               else {
                 reread = TRUE;
                 return((struct term *)c);
               }
          default: error("illegal list : ? ");
        }
}

struct term *Rpst(flag)         /* pst */
int flag;
{
   struct term *p = (struct term *)Npst(flag);
   struct term *t;
   struct eclause *pobj = (struct eclause *)NULL;

   while (1) {
     advance;
     if (cbuf == ')') return(p);
     t = Rterm(999, flag);
     if (Pred(t) != PNAME_P) {
       error_detail(1,(struct pair *)NULL,
               "illegal PST: delimiter of PST should be '/'");
     }
     if ((! is_functor(Arg1(t)) || (Arg1(t)->t_arity != 0))) {
       error_detail(1,(struct pair *)NULL,
               "illegal PST: PNAME of PST should be ATOM");
     }
      pobj = insert_pstobj(t,pobj,flag);
     switch (Rtoken()) {
      case COMMA:
          break;
      case BRACKET:
          reread = TRUE;
          ((struct pst *)p)->p_list = pobj;
          return(p);
      default:
          sprintf(nbuf,"illegal pst : >>> %c <<<",cbuf);
          error(nbuf);
        }
   }
}

struct eclause *insert_pstobj(val,tail,flag)
struct term *val;
struct eclause *tail;
int flag;
{
  struct eclause *pre, *ptop = tail;
  register int f,i;
  if (! is_functor(Arg1(val))) {
    error_detail(Arg1(val),(struct pair *)NULL,
                 "illegal Property name for PST");
  }
  if (tail == (struct eclause *)NULL)
    return(Npstobj(val,(struct pair *)NULL,tail,flag));

  pre = tail;
  f = Pred(Arg1(val))->f_number;

  while (tail != (struct eclause *)NULL) {
    i = Pred(Arg1(tail->e_form))->f_number - f;
    if (i > 0) {
      if (pre == tail) return(Npstobj(val,(struct pair *)NULL,tail,flag));
      else break;
    }
    if (i==0) {
      wfp = stderr;
      Pterm(Arg1(tail->e_form),(struct pair *)NULL); NL;
      Pterm(Arg1(val),(struct pair *)NULL);
      error("ERROR: there are same PNAMES in PST");
    }
    pre = tail;
    tail = tail->e_link;
  }
  pre->e_link = Npstobj(val,(struct pair *)NULL,tail,flag);
  return(ptop);
}

int is_term_end(c)
register char c;
{
  switch(c) {
    case ',':
```

```
    case ')':
    case ']' :
    case ',':
    case ';':
    case '}': return(TRUE);
    default:  return(FALSE);
    }
}

int prefix_is_atom(m)
int m;
{
  switch(tokentype) {
    case FULLSTOP: return(TRUE);
    case COMMA:
    case BRACKET:
      return(is_term_end(nbuf[0]));
    case NAME: {
      register struct operator *o;
      if ((o = op_search(nbuf,INFIX)) != NULL)
        if ((o->o_prec   [(o->o_type & 0010) != 0)] >= m) return(TRUE);
      if ((o = op_search(nbuf,POSTFIX)) != NULL)
        if ((o->o_prec   [(o->o_type & 0010) != 0)] >= m) return(TRUE);
      }
    default:  return(FALSE);
    }
}

struct term *Rterm_half(n,flag,m)
int n, flag, *m;
{
  register struct term *t;
  char tempname[NAME_MAX];
  int ac = CONSTANT_TERM;

  switch (Rtoken()) {
    case BRACKET:
      switch (nbuf[0]) {
        case '[':
          t = ((struct term *)Rlist(flag));
          if ((Rtoken() != BRACKET) || nbuf[0] != ']') {
            error_detail(t,(struct pair *)NULL,"Syntax error     ] missing");
          }
          advance;
          return(t);
        case '(':
          advance;
          t = Rterm(1200,flag);
          if ((Rtoken() != BRACKET) || nbuf[0] != ')') {
            error_detail(t,(struct pair *)NULL,"Syntax error - ) missing");
          }
          advance;
          if (is_clause(t)) {
            struct clause *c = (struct clause *)t;
            return((c->c_link == NULL) ? c->c_form : t);
            }
          else return(t);
        case '[':
          t = (struct term *)Rpst(flag);
          if ((Rtoken() != BRACKET) || nbuf[0] != ']') {
            error_detail(t,(struct pair *)NULL,"Syntax error --- ] missing");
          }
          advance;
          return(t);
        default :
          sprintf(nbuf,"Syntax Error: unexpected >>> %c <<<",nbuf[0]);
          error(nbuf);
        }
    case VARNAME: /* variable */
      if (streq(nbuf, "_")) return(Anonymous_var);
      if((t = varsearch(nbuf))==NULL)
          return(Nvar(nbuf,flag));
      else return(t);
    case NUMBER:  /* number */
      return(Nnum(nbuf,flag));
    case STRING:  /* string */
      return(Nstr(nbuf,flag));
    case FILE_TYPE: { /* file pointer */
      int pt;
      sscanf(nbuf,"%x",&pt);
      return(Nfile((FILE *)pt));
      }
    case NAME: /* name */
      strcpy(tempname,nbuf); /* tempname <- nbuf */

      /* constant or functor */
      if (check('(')) {
        register int i,arity = 0;
        struct clause *temp, *argstack;
        argstack = temp
          = Nclause((struct term *)NULL,(struct clause *)NULL,TEMPORAL);

        while(1) {
          reread = FALSE; /* read term */
          t = Rterm(999,flag);
          if (notconst(t)) ac = NOT_CONSTANT_TERM;
          temp->c_link = Nclause(t,(struct clause *)NULL,TEMPORAL);
          temp = temp->c_link;
          arity++;
          if (tokentype != COMMA) break;
          advance;
        }
        skip(')');
        t = Nterm(arity,flag);
        if (ac==CONSTANT_TERM) t->t_arity = -arity;
        t->type.t_func = Predicate(tempname,arity);
        temp = argstack;
        for(i=0; i < arity; i++) {
          temp = temp->c_link;
          t->tag.t_arg[i] = temp->c_term;
```

```
        }
        return(t);
      }

      {
        struct operator *o;

        if ((o = op_search(tempname, PREFIX)) == NULL) {
          t = Nterm(0,flag);
          t->type.t_func = Predicate(tempname,0);
          return(t);
        }

        if (o->o_prec > n) {
          sprintf(nbuf,"Syntax Error:>>> %s <<<",tempname);
          error(nbuf);
        }
        Rtoken();
        reread=TRUE;
        if (prefix_is_atom(o->o_prec)) {
          if (*m > n) error("Syntax error");
          t = Nterm(0,flag);
          t->type.t_func = Predicate(tempname,0);
          return(t);
        }

        (t = Nterm(1,flag))->type.t_func = o->o_func;
        Arg1(t) = Rterm((o->o_prec - o->o_type & PREFIX),flag);
        *m = o->o_prec;
        if (isconst(Arg1(t))) t->t_arity = -t->t_arity;
        return(t);
      }
    default: /* else */
      sprintf(nbuf,"Syntax error      unexpected %c",cbuf);
      error(nbuf);
    }
}

struct term *Rterm_leftover(n,m,flag,t)
register int n,flag,m;
struct term *t;
{
  struct operator *o;
  int ac = (noteconst(t)) ? NOT_CONSTANT_TERM : CONSTANT_TERM;

  switch(Rtoken()) {
    case NAME:
      reread=FALSE;
      if ((o = op_search(nbuf,INFIX)) != NULL) {
        if ((o->o_prec <= n) &&
            ((o->o_prec - ((o->o_type & 0010) ? 1 : 0)) >= m)) {
          struct term *tt = Nterm(2,flag);
          tt->type.t_func = o->o_func;
          Arg1(tt) = t;
          Arg2(tt) = Rterm(o->o_prec - (o->o_type & 0001), flag);
          if ((ac != CONSTANT_TERM) && (isconst(Arg2(tt))))
            tt->t_arity = -tt->t_arity;
          return(Rterm_leftover(n,o->o_prec,flag,tt));
        }
      }
      if ((o = op_search(nbuf,POSTFIX)) != NULL) {
        if ((o->o_prec <= n) &&
            (o->o_prec >= (m + ((o->o_type & 0010) ? 1 : 0)))) {
          struct term *tt = Nterm(1,flag);
          tt->type.t_func = o->o_func;
          Arg1(tt) = t;
          if (ac == CONSTANT_TERM) tt->t_arity = -tt->t_arity;
          return(Rterm_leftover(n,o->o_prec,flag,tt));
        }
      }
    case FULLSTOP:
      reread = TRUE;
      return(t);
    case BRACKET:
      switch(nbuf[0]) {
        case '(':
        case '[':
          sprintf(nbuf,"Syntax Error:>>> %c <<<",nbuf[0]);
          error(nbuf);
      }
      return(t);
    case COMMA:
      if ((n >= 1000) && (m < 1000)) {
        struct term *tt;

        advance;
        tt = Rterm(1000,flag);
        reread = TRUE;
        if (! is_clause(tt))
          tt=(struct term *)Nclause(tt,(struct clause *)NULL,flag);
        t = (struct term *)Nclause(t,(struct clause *)tt,flag);
        if (n > 1000)
          return(Rterm_leftover(n,1000,flag,t));
      }
    default:
      return(t);
    }
}
```

```
/*-------------------------------------------------------------------
*               cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                        1989  91
*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------
*           <<<< print.c >>>>>>
*           print out routine
-------------------------------------------------------------------*/

#define DEBUG 0                 /* when debug, 1 */
#include "include.h"

void Pterm_core(),Peclause_core(),Pclause_core(),Pcahc_core();
void init_pp(),scanpst_term(),scanpst_clause(),scanpst_eclause(),print_pp();
int pp_number();

/* global vars */
int PST_PRINT_NUM;              /* # of different pats */

/* Classification of Characters */
#define BL  001  /* blank */
#define UC  002  /* Upper Character */
#define LC  003  /* Lower Character */
#define UL  004  /* UnderLine */
#define N   005  /* Numeric */
#define SG  006  /* sign, + - */
#define SP  007  /* special character */
#define Q   010  /* single/double quote */
#define CT  011  /* Cut */
#define CM  012  /* comment character */
#define BR  013  /* Brackets, Comma */
#define CO  014  /* Constraint Marker */

#define kanzi(CH)       (CH < 0) /* for EUC */
#define alphabet(CH)    ((char_type[CH] <= N) || (char_type[CH] >= UC))
#define is_lower(CH)    ((kanzi(CH)) || (char_type[CH] == LC))

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*    print basic structures:
* Pterm(t,e)
* Peclause(ec) : print eclause
* Pclause(c,e): print clause with delimiter ','
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void Pterm(t,e)                 /* print term */
struct term *t;
struct pair *e;
{
        init_pp();
        scanpst_term(t,e);
        Pterm_core(t,e,Print_Depth);
        if(PST_PRINT_NUM > 1)
        {
                tputc(';');
                print_pp(Print_Depth); /* $1=[...],$2=[...],... */
        }
}

void Peclause(ec)               /* print eclause */
struct eclause *ec;
{
        init_pp();
        scanpst_eclause(ec);
        Peclause_core(ec,Print_Depth);
        if(PST_PRINT_NUM > 1)
        {
                tputc(';');
                print_pp(Print_Depth);
        }
}

void Pclause(c,e)               /* print clause */
struct clause *c;
struct pair *e;
{
        init_pp();
        scanpst_clause(c,e);
        Pclause_core(c,e);
        if(PST_PRINT_NUM > 1)
        {
                tputc(';');
                print_pp(Print_Depth);
        }
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* showhorn(body,constraint,env): print CNRC
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void Showhorn(c,cst,e)  /* Show horn clause */
register struct clause *c,*cst;
register struct pair *e;
{
        void P_hclause();

        if (cst == (struct clause *)NULL) P_hclause(c,e);
        else
        {
                init_pp();
                scanpst_clause(c,e);
                scanpst_clause(cst,e);
                Pcahc_core(c,cst,e);    /* H:-Body;Constraint */
                if (PST_PRINT_NUM > 1)
                {
                        tputc(',');
                        print_pp(Print_Depth);
                }
                tputc('.');
        }
}
```

```
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* Pgoal(n) : print goal in refutation
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void Pgoal(n)           /* print goal in refute() */
struct node *n;
{
        init_pp();
        scanpst_clause(n->n_clause, n->n_env);
        scanpst_eclause(n->n_constraint);
        Psequence(n->n_clause,n->n_env,Print_Depth);
        if (n->n_constraint != (struct eclause *)NULL) {
                tpute(';');
                Peclause_core(n->n_constraint,Print_Depth);
                if(PST_PRINT_NUM > 1)
                {
                        tpute(',');
                        print_pp(Print_Depth);
                }
        }
        else if(PST_PRINT_NUM > 1)
        {
                tpute(';');
                print_pp(Print_Depth);
        }
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* Showfunc(func): print definition of a predicate
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void Showfunc(f)    /*  Show definitions of function *f  */
register struct func *f;
{
        register struct set *ts;

        if (isuser(f))
        {
                for (ts = f->def.f_set; ts != NULL; ts = ts->s_link) {
                        Showhorn(ts->s_clause, ts->s_constraint,
                                (struct pair *)NULL);
#if DEBUG == 1
                        printf("(an=%d bn=%d)",ts->s_anumber,ts->s_bodynumber);
#endif
                        NL;
                }
        }
}


/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* P_hclause(cl,e): print Horn clause
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void P_hclause_sub(cl,e)        /* H:-C1,C2,...Cn */
struct clause *cl;
struct pair *e;
{
        register struct clause *c;

        Pterm_core(cl->c_form,e,Print_Depth);
        c = cl->c_link;
        if (c != NULL)
        {
                tprint0(" :- ");
                Pclause_core(c,e);
        }
}

void P_hclause(cl,e)
struct clause *cl;
struct pair *e;
{
        register struct clause *c;

        init_pp();
        scanpst_clause(cl,e);
        P_hclause_sub(cl,e,Print_Depth);
        if (PST_PRINT_NUM > 1)
        {
                tpute(';');
                print_pp(Print_Depth);
        }
        tpute('.');
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* P_delause(cl,e): print derivation clause of unfold/fold trans.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void P_delause(cl,e)
struct clause *cl;
struct pair *e;
{
        register struct clause *c;

        init_pp();
        scanpst_clause(cl,e);
        Pterm_core(cl->c_form,e,Print_Depth);
        c = cl->c_link;
        if (c != NULL)
        {
                tprint0(" <- ");
                Pclause_core(c,e);
        }
        if (PST_PRINT_NUM > 1)
        {
                tpute(';');
                print_pp(Print_Depth);
        }
        tpute('.');
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
* Shownewfunc(): print itrace
*****************************************************************/
void Shownewfunc()   /*  Show def of new functions constructed in integrate  */
{
        register struct itrace *it;

        for (it = newf_list; it != NULL; it = it->it_link){
                tprint2("<%d.%d> ",it->it_anumber,it->it_cnumber);
                P_delause(it->it_clause,NULL);
                NL;
        }
}

/* *****************************************************************
* writenewfunc(): print itrace to file
*****************************************************************/
void writenewfunc()
{
        register struct itrace *it;
        register struct func *f;

        if (newf_list == NULL) return;

        for (it = newf_list; it != NULL; it = it->it_link){
                f = it->it_clause->c_form->type.t_func;
                if (isnoreduced(f) && (f->def.f_set != NULL))
                {
                        tprint0("$ ");
                        P_delause(it->it_clause,NULL);
                        NL;
                }
        }
}

/* ---------- local functions ---------- */
int quote_needed(f)             /* need quote? */
struct func *f;
{
        register char *n = f->f_name;

        if (f == CUT_P) return(FALSE);

        if (! is_lower(*n)) return(TRUE);
        for ( ; *n != '\0'; n++) {        if (kanzi(*n)) n++;
          else if (! alphabet(*n)) return(TRUE);
        }
        return(FALSE);
}

void Pvar(t, n) /* print var with env. as "t_n" */
register struct term *t;
int n;
{
     if (((struct var *)t)->v_type==VAR_VOID_TYPE) tpute('_')
     else if (streq(vname(t)," ")) tprint1(" %u",n)
     else {
             tprint2("%s_%u",vname(t),n);
#if DEBUG == 1
             tprint2("<%d%d%d>",vhowbeoccurrence(t),voccurrence(t));
#endif
     }
}


void Pclause_core(c,e)          /* print clause main */
struct clause *c;
struct pair *e;
{
        if (c == NULL) return;
        for (;;)
        {
                Pterm_core(c->c_form,e,Print_Depth);
                c = c->c_link;
                if (c == NULL) return;
                tprint0(", ");
        }
}

void Pcahe_core(c,cst,e)        /* print CAHC main */
register struct clause *c,*cst;
register struct pair *e;
{
        Pterm_core(c->c_form,e,Print_Depth);    /* print head */
        if (c->c_link != NULL) {        /* print body */
                tprint0(":-");
                Psequence(c->c_link,e,0); /* 0 means infinity */
        }
        if (cst != NULL){       /* print constraint */
                tprint0("; ");
                Psequence(cst,e,0);
        }
        /* tpute('.'); */
}

void Pterm_core(t,e,d)  /* print term main */
register struct term *t;
register struct pair *e;
int d;
{
  register struct pair *p;

  if (t == NULL) {
    if (e == NULL) {
      tprint0("nil");
      return;
    }
    else return;        /* print nothing */
  }

  if (isvar(t)) {
```

```
    if (e == NULL) {
      Pvar(t, vnumber(t));
      return;
    }
    down(p,t,e);
    if(p != NULL) {     /* if t is not var */
      Pvar(t, (int)(p - cheap)); /* print its name with env */
      return;
    }
  }

  /* print literal */
  if (t == NIL) {               /* if t is NIL list ([]) */
    tprint0("[]");
    return;
  }

  if (!(--d)) {
    tprint0("???");
    return;
  }

#if DEBUG == 1
  if (isatom(t)) printf("~");
  else
  {
          if(t->type.ident < 100)
                  printf("<t=%x e=%x y=%d>",t,e,t->type.ident);
          else printf("<t=%x e=%x>",t,e);
  }
#endif

  switch (t->type.ident) {
    /* case VAR_VOID_TYPE:
       case VAR_PST_TYPE:
       case VAR_GLOBAL_TYPE: */ /* already checked in the above */
    case ATOMIC_TYPE: /* atomic */
      switch (t->t_arity) {
        case FLOAT_NUM : tprintl("%f",num_value(t)); /* float */
          return;
        case INT_NUM   : tprintl("%d",(int)(num_value(t))); /* int */
          return;
        case STRING : tprintl("\"%s\"", str_value(t)); /* string */
          return;
        default : tprintl("#%x", str_value(t)); /* file */
          return;
        }
    case PST_TYPE: /* pst */
      /* tputc('['); */
      Ppst(t,e,d);
      /* tputc(']'); */
      return;
    case CLAUSE_TYPE: /* clause */
      tputc('(');
      Psequence((struct clause *)t,e,d);
      tputc(')');
      return;
    case BCLAUSE_TYPE: /* bclause */
      Pbclause_core(t,d);
      return;
    case LIST_TYPE:
    case CONST_LIST_TYPE: /* list */
      tputc('[');
      Psequence((struct clause *)t,e,d);
      tputc(']');
      return;
    default:                        /* complex term */
      Pfunctor(t,e,d);
      return;
    }
}

void Pfunctor(t,e,d)  /* print complex term */
register struct term *t;
register struct pair *e;
int d;
{
  register struct func *f = t->type.t_func; /* f is functor of t */
  register int i, arity = f->f_arity;
  struct operator *o;

  if ((arity == 2) || (arity == 1)) {
    for(o = o_list; o != NULL; o = o->o_link)
      if (o->o_func == f) {
        switch (o->o_type & INFIX) {
        case INFIX:
          Pterm_core(Arg(t,0),e,d);
          tprintl("%s",f->f_name);
          Pterm_core(Arg(t,1),e,d);
          return;
        case PREFIX:
          tprintl("%s ",f->f_name);
          Pterm_core(Arg(t,0),e,d);
          return;
        case POSTFIX:
          Pterm_core(Arg(t,0),e,d);
          tprintl(" %s",f->f_name);
          return;
        }
      }
  }

  /* print functor name */
  if (quote_needed(f))
    tprintl("\'%s\'", f->f_name)
  else
    tprintl("%s", f->f_name);
  if(t->t_arity==0) return;     /* if t is const */

  tputc('(');           /* print args */
```

```
  i=0;
  while (1) {
    Pterm_core(Arg(t,i), e,d);  /* print one arg */
    if(++i >= arity) {
      fputc(')');
      break;
    }
    fprint0(", ");
  }
}

void Ppst_content(ptt,d)        /* [11/v1,12/v2,...] temporal PST*/
struct eclause *ptt;
{
        fputc('[');
        while (ptt->c_link != (struct eclause *)NULL) {
                Pterm_core(ptt->c_form, ptt->c_env,d);
                fprint0(", ");
                ptt = ptt->c_link;
        }
        Pterm_core(ptt->c_form, ptt->c_env,d);
        fputc(']');
}

                                /* patch 92/1/22 by H.Tsuda */
void Ppst_content2(ptt,env,d)   /* [11/v1,12/v2,...] static PST with env */
struct eclause *ptt;
struct pair *env;
{
        fputc('[');
        while (ptt->c_link != (struct eclause *)NULL) {
                Pterm_core(ptt->c_form, env,d);
                fprint0(", ");
                ptt = ptt->c_link;
        }
        Pterm_core(ptt->c_form,env,d);
        fputc(']');
}

void Ppst(t,e,d)                /* print pst (in Pterm_core) */
struct term *t;                 /* actually, (struct pst *) */
struct pair *e;
int d;
{
  register struct eclause *ptt = ((struct pst *)t)->p_lists;
  struct pst_item *target;
  int n;

  target = find_pstitem(t,e);
  if (target != (struct pst_item *)NULL) { /* print temporal PST */
    ptt = target->p_lists;
    if (ptt == (struct eclause *)NULL) {
            fprint0("[]");
            return;
    }
```

```
#if DEBUG == 1
    printf("<pl=%x>",ptt);
#endif
    n = pp_number(ptt);
    if (n > 0) {                /* called more than once!! */
            fprintl(" p%d",n);
            return;
    }
    Ppst_content(ptt,d);        /* print temporal PST */
  }
  else {                        /* print (static) PST in program */
    if (ptt == (struct eclause *)NULL)
    {
            fprint0("[]");
            return;
    }
#if DEBUG == 1
    printf("<pl=%x>",ptt);
#endif
    n = pp_number(ptt);
    if (n > 0) {                /* called more than once!! */
            fprintl("_p%d",n);
            return;
    }
    Ppst_content2(ptt,e,d);     /* print static PST with env */
  }
}

void Peclause_core(ec,d)        /* print eclause main */
struct eclause *ec;
int d;
{
        if (ec == NULL) return;
        while (1) {
                Pterm_core(ec->c_form, ec->c_env,d);
                ec = ec->c_link;
                if (ec == NULL) return;
                fputc(',');
        }
}

void Psequence(t,e,d)   /* print content of list t */
struct eclause *t;
register struct pair *e;
int d;
{
  register struct pair *p;
  register struct term *tt = (struct term *)t;

  if ((tt == NULL) || (tt == NIL)) return;

  while (1) {
    Pterm_core(t->c_form,e,d);  /* print the first argument */
    t = t->c_link;
    tt = (struct term *)t;
```

```
    if (tt == NULL) return;
    if (isvar(tt)) {
      if (e == NULL) {
        tprint0(" | ");
        Pvar(tt, vnumber(tt));
        return;
      }
      down(p, tt, e);
      if (p != NULL) { /* if tt is variable */
        tprint0(" | ");
        Pvar(tt, (int)(p - cheap));
        return;
      }
    }
    if (! (is_list(tt) || is_clause(tt))) {
      if (tt == NIL) return;
      tprint0(" | ");
      Pterm_core(tt,e,d);
      return;
    }
    tpute(',');
    t = (struct clause *)tt;
      }
}

/* ---------- functions for PST pretty print ---------- */
struct pstprint
{
        struct eclause *pp_ec;
        int pp_num;
        struct pstprint *pp_link;
};

struct pstprint *PST_PRINT_LIST; /* pst save entry */

void init_pp()
{
        PST_PRINT_LIST = NULL;
        PST_PRINT_NUM = 1;
}

void print_pp(d)                /* $1-[...],$2-[...],... */
int d;                          /* printing depth */
{
        register struct pstprint *pp;
        int printed = 0;

        for(pp = PST_PRINT_LIST; pp !=NULL; pp=pp->pp_link)
        {
                if (printed != 0) tpute(',');
                if (pp->pp_num != 0){
                        tprint1("_p%d=",pp->pp_num);
                        Ppst_content(pp->pp_ec,d);
                        printed=1;
                }
        }
}

int pp_number(ec)               /* print PST number */
struct eclause *ec;
{
        register struct pstprint *pp,*ppn;

        for (pp = PST_PRINT_LIST; pp != NULL; pp = pp->pp_link)
                if (pp->pp_ec == ec) return(pp->pp_num);
        return(0);
}

void scanpst_term(t,e)  /* scan PST in a term */
register struct term *t;
register struct pair *e;
{
  register struct pair *p;
  void addpst(),scanpst_functor();

  if (t == NULL) return;
  if (isvar(t)) {
          if (e == NULL) return;
          down(p,t,e);
          if(p != NULL) return;
  }
  if (t == NIL) return; /* if t is NIL list (||) */
  switch (t->type_ident) {
    case ATOMIC_TYPE: /* atomic */
          return;
    case PST_TYPE: /* pst */
          addpst(t,e);
          return;
    case CLAUSE_TYPE: /* clause */
          scanpst_clause((struct clause *)t,e);
          return;
    case ECLAUSE_TYPE: /* eclause */
          scanpst_eclause((struct eclause *)t);
    case LIST_TYPE:
    case CONST_LIST_TYPE: /* list */
          scanpst_clause((struct clause *)t,e);
          return;
    default:                    /* complex term */
          scanpst_functor(t,e);
          return;
  }
}

void scanpst_clause(t,e)        /* modify Psequence() */
struct clause *t;
struct pair *e;
{
  register struct pair *p,
  register struct term *it = (struct term *)t;
```

```
  if ((tt == NULL) || (tt == NIL)) return;

  while (1) {
          scanpst_term(t->c_form,e);    /* scan the first argument */
          t = t->c_link;
          tt = (struct term *)t;

          if (tt == NULL) return;
          if (isvar(tt)) {
                  if (e == NULL)  return;
                  down(p, tt, e);
                  if (p != NULL) return;
          }
          if (! (is_list(tt)  || is_clause(tt))) {
                  if (tt == NIL) return;
                  scanpst_term(tt,e);
                  return;
          }
          t = (struct clause *)tt;
  }
}

void scanpst_eclause(ec)
struct eclause *ec;
{
        if (ec == (struct eclause *)NULL) return;
        scanpst_term(ec->e_form,ec->e_env);
        scanpst_eclause(ec->e_link);
}

void scanpst_functor(t,e)
struct term *t;
struct pair *e;
{
        int i,arity;
        arity = t->t_arity;
        for (i = 0; i < arity; i++)
                scanpst_term(Arg(t,i),e);
}

void addpst(t,e)
struct term *t;
struct pair *e;
{
  register struct eclause *ptt = ((struct pst *)t)->p_lists;
  struct pst_item *target;
  struct pstprint *pp,*ppnew;

  target = find_pstitem(t,e);
  if (target != (struct pst_item *)NULL)  ptt = target->p_lists;
  if (ptt == (struct eclause *)NULL) return;
  for (pp = PST_PRINT_LIST; pp != NULL; pp = pp->pp_link)
          if (pp->pp_ec == ptt) {
                  if (pp->pp_num == 0)
                          pp->pp_num = PST_PRINT_NUM++;
                  return;
          }
  MEMORY_MALOC(ppnew,pstprint,TEMPORAL);
  ppnew->pp_ec = ptt;
  ppnew->pp_num = 0;
  ppnew->pp_link = PST_PRINT_LIST;
  PST_PRINT_LIST = ppnew;
}

/*      ---------- functions for debug ---- ---------- ---- - - */
void P_var(vlist)               /* for debug */
struct term *vlist;
{
        register struct term *v;

        for (v = vlist; v != NULL; v = vlink(v))
        {
                printf("%s (%d) ",vname(v),v->type.ident);

                Pclause_core(vconstraint(v),NULL);
                NL;
        }
}

void showvar(v)                 /* show variable (for debug) */
struct term *v;
{
        putchar('(');
        while (v != NULL)
        {
                printf("%s ",vname(v));
                v = vlink(v);
        }
        putchar(')');
}
```

```
/*----------------------------------------------------------------------
*               cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 9)
----------------------------------------------------------------------*/
/*----------------------------------------------------------------------
*               <<<< refute.c >>>>
*               Prolog refutation
----------------------------------------------------------------------*/

#include "include.h"
#include <signal.h>

#define is_dead(n) (n->n_set == NULL)
#define is_tip(n)  (n->n_clause == NULL)
#define is_root(n) (n->n_link == NULL)

struct node *Production(), *extend(), *proceed_node();
struct node *Newnode(),*backtrack_node();
struct set *init_set();
void Trace_True(),Trace_False(),Trace_Unification(),Trace_Answer();
void Trace_True2(), Trace_False2();
int Trace_Goal();
void Pbinding();
/*
[refute]----------------------------
.   backtrack_node
.   .   Trace_False2
.   Trace_False
.   Trace_Goal
.   .   print_ancestorst
.   Trace_True
.   is_dead.
.   extend
.   .   [system_funct:on]
.   .   init_set
.   .   resolve1
.   .   .   [unify]
.   .   .   Trace_Unification
.   is_tip.
.   proceed_node
.   .   Trace_True2.
.   .   next_goal1
.   .   have_nextgoal1
*/

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  Prolog refutation entry
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int refute(Root,n,Status)
struct node *Root, *n;
int Status;                     /* UP,DOWN,BACKTRACK */
{
  struct node *m;

  while(1)
    {
      if (Trace_Goal(n)==FALSE) return(FALSE);
      m = extend(n, Status);
      if (!is_dead(n)) Last_BT = n;
      if (m == NULL)    /* fail */
        {
          Trace_False(n);
          if (n == Root) return(FALSE);
          Status = BACKTRACK;
          Last_BT = n = backtrack_node(n->n_last);
        }
      else if (is_tip(m)) /* nil clause */
        {
          Trace_True(n);
          Status = UP;
          n = proceed_node(m,Last_BT);
          if (n == NULL) return(TRUE);
        }
      else {
        Status = DOWN;
        n = m;
       }
    }
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  head unification
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct node *extend(n, status) /* extend goal */
struct node *n;
int status;
{
        register struct term *sliteral; /* selected literal */
        struct node *m;
        register struct pair *p,*env;

        if (n->n_count > Refcount) {
                fprintf(" <<%d] fail! (over refute counter)\n",n->n_count);
                return(NULL); /* counter over = fail */
        }
        if (is_tip(n)) return(n); /* no goal */
        sliteral = n->n_clause->c_form;
        env = n->n_env;
        down(p,sliteral,env);
        if (p != NULL) return(NULL); /* goal is real var */
        m = Newnode((struct clause *)NULL,n->n_constraint,
                    (struct pair *)NULL,n,n);
        if (is_funcsys(sliteral->type.t_func)) /* functional syspred */
        {
                if (system_function(sliteral, env, n) == SYSFAIL)
                        return(NULL);
                else {
                  n->n_set = NULL;
                  return(m);
```

```
                }
        }
        if (is_nofunesys(sliteral->type.t_func)) /* sys pred.  */
        {
                if (system_pred(sliteral, env, n, m, status) == SYSFAIL)
                        return(NULL);
                else {
                  return(m);
                }
                /* n->n_set may be EMPTY_DEF */
        }
        if (is_dead(n)) return(NULL);
        if (resolve(n, m, sliteral, env) == FALSE)
                return(NULL); /* user pred.: resolution */
        m->n_usp = usp;
        m->n_hp = hp;
        m->n_ep = ep;
        m->n_set = init_set(m);
        return(m);
}


int resolve(n0, n, sliteral, env) /* resolution: called by extend() */
struct node *n0,*n;
struct term *sliteral;
struct pair *env;
{
  struct ustack *usave;
  int *hsave;
  struct pair *esave;
  register struct set *s;
  register struct eclause *ec;

  usave = usp;hsave = hp;esave = ep;
  for (s = n0->n_set; s != NULL; s = s->s_link)
    {
      n->n_env = Nenv((int)s->s_anumber);
      if (unify(sliteral, env, s->s_clause->c_form, n->n_env,0) == FALSE){
        undo(usave);hp = hsave; ep = esave;
        continue;
      }
      ec = transform(n0->n_constraint, s->s_constraint, n->n_env);
      if (ec == (struct eclause *)MFAIL) {
        /* constraint transformation failure */
        undo(usave);hp = hsave; ep = esave;
        continue;
      }
      n->n_constraint = ec;
      Trace_Unification(n0,s);
      n0->n_set = s->s_link;
      n->n_clause = s->s_clause->c_link;
      return(TRUE);
    }
  return(FALSE);
}
```

```
/*************************************************************
    make and process node
*************************************************************/
struct node *Newnode(goal, icons, env, nlink, nlast)
struct clause *goal;
struct eclause *icons;
struct pair *env;
struct node *nlink, *nlast;
{
        struct node *n;

        n = new(node);
        n->n_last = nlast;      /* backtrack node */
        n->n_link = nlink;      /* mother node */
        n->n_clause = goal;     /* goal */
        n->n_env = env;
        n->n_hp = hp;
        n->n_ep = ep;
        n->n_usp = usp;
        if (nlink == NULL) n->n_count = 0;
        else n->n_count = nlink->n_count + 1;
        n->n_spy = 0;
        n->n_tmp = 0;
        n->n_scount = 0;
        n->n_constraint = icons;
        if (goal != NULL) n->n_set = init_set(n);
        else n->n_set = NULL;
        return(n);
}

struct set *init_set(n)
register struct node *n;
{
        register struct term *t;
        register struct pair *e,*p;
        struct set *s;

        if (n->n_clause == NULL) return(NULL);
        t = n->n_clause->c_form;
        e = n->n_env;
        deref(p,t,e);
        if (p != NULL) return(NULL); /* goal is var */
        n->n_spy = is_trace && isspy(t->type.t_func);
        if (isuser(t->type.t_func)) {
          if (((s = t->type.t_func->def.f_set) == (struct set *)NULL)
              && (Handle_Undefined == TRUE)) {
            sprintf(mbuf,">>> %s <<< is UNDEFINED!",t->type.t_func->f_name);
            error(mbuf);
          }
          else return(s);
        }
        else return(NULL);
}
```

```
struct node *backtrack_node(n)  /* restore stack, heap */
struct node *n;
{
  while (n != NULL)
    {
      Trace_False2(n);
      if(! is_dead(n)) {
        undo(n->n_usp);
        hp = n->n_hp;
        ep = n->n_ep;
        return(n);
      }
      n = n->n_last;
    }
  return(NULL);
}

struct node *proceed_node(n,btnode)
struct node *n,*btnode;
{
        register struct node *m;
        int have_nextgoal();
        struct node *next_goal();

        for (m = n; m != NULL; m = m->n_link)
        {
          Trace_True2(m);
          if (have_nextgoal(m))
            return(next_goal(m,n,btnode));
        }
        return(NULL);
}

int have_nextgoal(n)            /* called by proceed_node() */
struct node *n;
{
        if (n->n_clause == NULL) return(FALSE);
        if (n->n_clause->c_link != NULL) return(TRUE);
        return(FALSE);
}

struct node *next_goal(m,oldnode,btnode) /* called by proceed_node() */
struct node *m, *oldnode,*btnode;
{
        struct node *n;

        n = Newnode(m->n_clause->c_link, oldnode->n_constraint,
                    m->n_env, m->n_link, btnode);
        n->n_count = oldnode->n_count +1;
        return(n);
}

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  print answer & binding
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int Panswer(root, vlist)        /* print solution */
struct node *root;
struct term *vlist;
{
  Trace_Answer(root);
  Pbinding(vlist, root->n_env);
  if (root->n_constraint != (struct eclause *)NULL) {
    tprint0("\n where ");
     Peclause(root->n_constraint);
  }

  if (Last_BT == NULL) {
    tprint0("\n");
    return(FALSE); /* no backtrack point */
  }
  if ((fp != stdin) || (!keyread(';')))
    return(FALSE);
  return(TRUE); /* more solution */
}

void Pbinding(vlist, env)       /* print var binding */
struct term *vlist;
struct pair *env;
{
        if (vlist == NULL) return;
        Pbinding(vlink(vlist),env);
        if ((strcmp(vname(vlist),"_") == 0)  ||
            (vlist->type.ident == VAR_PST_TYPE)) return;
        tprint1("  %s = ",vname(vlist));
        Pterm(vlist,env);
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  trace     interaction with user
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int Trace_Goal(n)
        struct node *n;
{
   void print_ancestors();

   if (n == NULL) return(FALSE);
   if (Is_Notrace) return(TRUE);
   if (Last_SKIP == n) Last_SKIP = NULL;
   if (Last_SKIP != NULL) return(TRUE);
   if (!(n->n_spy)) return(TRUE);
   tprint1(" [%d]>",n->n_count);
   Pgoal(n);
   if ((Is_Steptrace) || (Is_Leap && (n->n_spy)))
   {
          Steptrace_mode;
          while(1) {
               tprint0("  #(trace ?)");
               switch (getchar()) {
                 case 'a' :
```

```
                case 'A' : print_ancestors(n); getchar(); break;
                case 'b' :
                case 'B' : {
                      int  *hsave;
                      struct pair *esave;
                      struct ustack *usave = utop;
                      hsave = hp; esave = ep;
                      utop = usp;
                      if (setjmp(unbreak_reset)) {
                              utop = usave; hp = hsave; ep = esave;
                              break;
                        }
                      while(1) {
                        prolog_execution();
                      }
                    }
                case  'z' :
                case 'Z' :
                case 'q' :
                case 'Q' : NL; tprint0("\n Execution Abort \n");
                  longjmp(reset,0);
                case '?' :
                case 'h' :
                case 'H' :
   tprint0("\nn : ancestors\tb : break\th : help\tl : leap\ts : skip\n");
   tprint0("<cr> : next\tq: abort\tf : fail return\n");
                  getchar(); break;
                case 'f' :
                case 'F' : return(FALSE);
                case 'l' :
                case 'L' : tflag = 3; return(TRUE);
                case 's' :
                case 'S' : last_SKIP = n; return(TRUE);
                default :  last_SKIP = NULL; return(TRUE);
                }
            }
        }
        else NL; return(TRUE);
}

void Trace_False(n)
struct node *n;
{
        struct func *f;
        struct pair *p, *e;
        struct term *t;

        if (!(n->n_spy)) return;
        if (Is_Leap) Steptrace_mode;
        if (last_SKIP == n) last_SKIP = NULL;
        if (last_SKIP != NULL) return;
        t = n->n_clause->c_form;
        if (isvar(t)) {
          e = n->n_env;
          down(p,t,e);
          f = t->type.t_func;
          if (is_funcsys(f))
          {
                  tprint2("  <<%d] false (%s)\n",n->n_count,f->f_name);
          }
          else if (is_nofuncsys(f))
          {
                  tprint2("  <<%d] fail (%s)\n",n->n_count,f->f_name);
          }
          else                        /* user pred */
          {
                  tprint2("   <-%d no- fail %s.\n",n->n_count,f->f_name);
          }
}

void Trace_True(n)
struct node *n;
{
        struct func *f;
        struct term *t;
        struct pair *p, *e;

        if (!(n->n_spy)) return;
        if (Is_Leap) Steptrace_mode;
        if (last_SKIP == n) last_SKIP = NULL;
        if (last_SKIP != NULL) return;
        t = n->n_clause->c_form;
        e = n->n_env;
        down(p,t,e);
        f = t->type.t_func;
        if (isuser(f)) return;
        tprintl("  <<%d] ",n->n_count);
        if (is_funcsys(f))
        {
                tprintl("true (%s)\n",f->f_name);
        }
        else
        {
                tprintl("success (%s)\n",f->f_name);
        }
}

void Trace_False2(n)
struct node *n;
{
        if (!(n->n_spy)) return;
        if (Is_Leap) Steptrace_mode;
        if (last_SKIP == n){
                tprintl("  <<%d] fail\n",n->n_count);
                last_SKIP = NULL;
        }
}

void Trace_True2(n)
```

```
struct node *n;
{
        if (!(n->n_spy)) return;
        if (last_SKIP    n){
                /* tprint1("  <<%d| true",n->n_count); */
                Last_SKIP = NULL;
        }
}

void Trace_Unification(n,s)
struct node *n;
struct sel *s;
{
        n->n_scount++;
        if (!(n->n_spy)) return;
        if (Is_Leap) Steptrace_mode;
        if (last_SKIP != NULL) return;
        tprint2("   <-%d %d-",n->n_count, n->n_scount);
        Showhorn(s->s_clause, s->s_constraint, (struct pair *)NULL);
        NL;
}

void Trace_Answer(toot)
struct node *root;
{
        if (Is_Notrace) return;
        Last_SKIP = NULL;
        tprint0("success.\n");
        Peclause(root->n_clause, root->n_env);
        if (root->n_constraint != (struct eclause *)NULL) {
                tprint0(" ; ");
                Peclause(root->n_constraint);
        }
        NL;
}

void print_ancestors(n)         /* called by Trace_goal() */
struct node *n;
{
   while (n != NULL) {
     tprint1(" [%d]>",n->n_count); Pgoal(n); NL;
     n = n->n_link;
     }
 }
```

```
/*************************************************************
*               CU-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 91
*************************************************************/
/*-------------------------------------------------------------
*               << UNIFY.C >>
*               safe/unsafe/PST unification
-------------------------------------------------------------*/

#include "include.h"
jmp_buf ufail;                  /* unification fail */

#define Npstob)(Head,Env,Tail,Flag)  Neclause(Head,Env,Tail,Flag)
#define UNSAFE 0
#define SAFE 1
#define NOEXTRACT 2
#define EXTRACT 1
int Ocheck_max;

/*****************************************************************
  int tunify(t,e,u,f,flag)
    term unification between (t,e) and (u,f)
    flag = 0: unsafe
           1: safe, no extract
           2: safe, extract
  return_value --> TRUE/FALSE
*****************************************************************/
int tunify(t,e,u,f,flag)        /* term unification entry */
register struct term *t, *u;
register struct pair *e, *f;
int flag;     /* 1:safe no extract, 2:safe extract, 0:unsafe */
{
        struct ustack *usave = usp;
        int  *hsave = hp;
        struct pair *esave = ep;

        Ocheck_max = 0;
        if (setjmp(ufail))
        {
                undo(usave);
                hp  hsave;
                ep  esave;
                return(FALSE);
        }
        if (flag == 1)
                safe_unify(t,e,u,f,NOEXTRACT);
        else if (flag == 2)
                safe_unify(t,e,u,f,EXTRACT);
        else
                unify(t,e,u,f);
        return(TRUE);
}

void ocheck(p, t, e)            /* occur check of normal unification      */
/* check if var p is contained in (t,e)   */
register struct pair *p;        /* var */
register struct term *t;
register struct pair *e;
{
  register struct pair *q;
  register int i, j;

  if (Ocheck_max++ > 50) longjmp(ufail,1); /* in case of infinite loop */
/*  printf("ocheck ");
  Pterm(p->p_body, p->p_env);
  printf(" in ");
  Pterm(t,e);
  NL;*/

  if (t == NULL) return;
  down(q, t, e);

  if (q != NULL) {      /* if t is var  */
    if (p == q) /*  occured !!  --> fail */
      longjmp(ufail, 1);
    else
      return;
  }

  switch (t->type.ident) {
    case ATOMIC_TYPE:
    case CONSI_LIST_TYPE:
        return;
    case LIST_TYPE:
    case CLAUSE_TYPE:
        ocheck(p,head_of_list(t),e);
        ocheck(p,tail_of_list(t),e);
        return;
    case PST_TYPE: { /* pst */
        struct eclause *ptt;
        struct pst_item *item;
        if ((item = find_pstitem(t,e)) == (struct pst_item *)NULL) {
                ptt = ((struct pst *)t)->p_lists;
                while (ptt != (struct eclause *)NULL) {
                        t = Arg2(ptt->e_form);
                        ocheck(p,t,e);
                        ptt = ptt->e_link;
                }
        }
        else {
            ptt = item->p_lists;
            while (ptt != (struct eclause *)NULL) {
            t = Arg2(ptt->e_form);
            e = ptt->e_env;
            ocheck(p, t, e);
            ptt=ptt->e_link;
           }
        }
        return;
```

```
      }
    default: /* functor */
      for(i = 0, j = t->t_arity; i < j; i++)
        ocheck(p, Arg(t,i), e);
    }
}

void unify(t, e, u, f)
register struct term *t, *u;
register struct pair *e, *f;
{
        register struct pair *p, *q;
        int i, j;

/*      printf("unify:");
        Pterm(t,e);
        printf(" and ");
        Pterm(u,f);
        NL; */

        down(p, t, e);
        down(q, u, f);
        if(p != NULL)           /* if t = var */
          if (p==Anonymous_env) return;  /* if t = Anonymous Var */
          else if(q != NULL)            /* t:var, u:var */
                if(p == q) /* t,u : the same var */
                        return;
                else if (q==Anonymous_env) /* u : Anonymous Var */
                        return;
                else {
                        upush(&(p->p_body)); /* t->u */
                        upush(&(p->p_env));
                        p->p_body = u;
                        p->p_env = f;
                        return;
                }
                else {                  /* t:var, u:nonvar */
                        upush(&(p->p_body)); /* t->u */
                        upush(&(p->p_env));
                        p->p_body = u;
                        p->p_env = f;
                        return;
                }
          else if(q != NULL)
                if (q==Anonymous_env) return;
                else {          /* t:nonvar , u:var */
                        upush(&(q->p_body)); /* u->t */
                        upush(&(q->p_env));
                        q->p_body = t;
                        q->p_env = e;
                        return;
                        }
        /* t,u : nonvar */

        switch (u->type.ident) {
        case ATOMIC_TYPE : /* t,u: atomic (string,num,quote) */
          {
            if ((t==u) || (atomic_equal(u,t))) return;
            else longjmp(ufail,1);
          }
        case LIST_TYPE:
        case CONST_LIST_TYPE:
                if (is_list(t)) {
                        unify(head_of_list(t),e,head_of_list(u),f);
                        unify(tail_of_list(t),e,tail_of_list(u),f);
                        return;
                      }
                longjmp(ufail,1);
        case CLAUSE_TYPE:
          if (is_clause(t)) {
                while ((t != NULL) && (u != NULL)) {
                  unify(((struct clause *)t)->c_form,e,
                        ((struct clause *)u)->c_form,f);
                  t=(struct term *)((struct clause *)t)->c_link;
                  u=(struct term *)((struct clause *)u)->c_link;
                }
                if (t == u) return;
              }
          longjmp(ufail, 1);
        case PST_TYPE:
          if (is_pst(t)) {
            pst_unify(t,e,u,f,UNSAFE);
            return;
          }
          longjmp(ufail, 1);
        default : /* functor */
          if(Pred(t) == Pred(u)) {/* t,u: complex term */
            for(i = 0, j = Pred(t)->f_arity; i < j; i++)
              unify(Arg(t,i), e, Arg(u,i), f);
            /* unify each arg */
            return;
          }
          longjmp(ufail,1);
        }
}


void unify_pst_extract();

void safe_unify(t, e, u, f, extflag)    /* unify with occur check */
register struct term *t, *u;
register struct pair *e, *f;
int extflag;                    /* NOEXTRACT or EXTRACT */
{
  register struct pair *p, *q;
  int i, j;

/*  printf("safe_unify ");
  Pterm(t,e);
  printf(" and ");
```

```
  Pterm(u,f);
  NL; */

  down(p, t, e);
  down(q, u, f);
  if(p != NULL)                    /* if t = var */
    if (p==Anonymous_env) return;  /* if t = Anonymous Var */
    else if(q != NULL)             /* t:var, u:var */
      if(p == q) /* t,u : the same var */
        return;
      else if (q==Anonymous_env) /* u : Anonymous Var */
        return;
      else {
        upush(&(p->p_body)); /* t >u */
        upush(&(p->p_env));
        p->p_body = u;
        p->p_env = f;
        return;
      }
    else {                         /* t:var, u:non var */
      ocheck(p,u,f);
      upush(&(p->p_body)); /* t >u */
      upush(&(p->p_env));
      p->p_body = u;
      p->p_env = f;
      return;
    }
  else if(q != NULL)
    if (q->Anonymous_env) return;
    else {                 /*  t:nonvar , u:var  */
      ocheck(q,t,e);
      upush(&(q->p_body)); /* u->t */
      upush(&(q->p_env));
      q->p_body = t;
      q->p_env = e;
      return;
    }
  /* t,u : nonvar */
  switch (u->type.ident) {
  case ATOMIC_TYPE : /* t,u: Atomic (string,num,quote) */
    {
      if ((t== u) || (atomic_equal(u,t))) return;
      else longjmp(ufail,1);
    }
  case LIST_TYPE:
  case CONST_LIST_TYPE:
    if (is_list(t)) {
      safe_unify(head_of_list(t),e,head_of_list(u),f,extflag);
      safe_unify(tail_of_list(t),e,tail_of_list(u),f,extflag);
      return;
    }
    longjmp(ufail,1);
  case CLAUSE_TYPE:
    if (is_clause(t)) {
      while ((t != NULL) && (u != NULL)) {
        safe_unify(((struct clause *)t)->c_form,e,
                   ((struct clause *)u)->c_form,f,extflag);
        t=(struct term *)((struct clause *)t)->c_link;
        u=(struct term *)((struct clause *)u)->c_link;
      }
      if (t == u) return;
    }
    longjmp(ufail, 1);
  case PST_TYPE:
    if (is_pst(t)) {
          if (extflag == NOEXTRACT)
                pst_unify(t,e,u,f,SAFE);
          else
                unify_pst_extract(t,e,u,f);
          return;
    }
    else longjmp(ufail, 1);
  default : /* functor */
    if(Pred(t) == Pred(u)) {/* t,u: complex term */
      for(i = 0, j = Pred(t)->f_arity;  i < j;  i++)
        safe_unify(Arg(t,i), e, Arg(u,i), f,extflag);
      /* unify each arg */
      return;
    }
    longjmp(ufail,1);
  }
}

struct pst_item *remove_pstitem(t,e)
struct term *t;
struct pair *e;
{
  struct pst_item *object, *target;
  struct pair *p;

  down(p,t,e);
  target = pstTable;
  while ((object = target->p_link) != (struct pst_item *)NULL) {
    if (object->p_var == p) {
      upush(&(target->p_link));
      target->p_link = object->p_link;
      return(object);
    }
    target = object;
  }
  return(object);
}

void pst_unify(t,e,u,f,safeflag)
register struct term *t,*u;
register struct pair *e,*f;
int safeflag;                    /* SAFE or UNSAFE */
{
  struct pst_item *target, *object;
```

```
/*  printf("pst_unify %d ",safeflag);
  Pterm(t,e);
  printf(" and ");
  Pterm(u,f);
  printf(" --> ");
*/
  target = find_pstitem(t,e);
  if (target != (struct pst_item *)NULL) {
    object = remove_pstitem(u,f);
    if (object != (struct pst_item *)NULL)
    {
        /*    printf("case1."); */
        unify_merge_psts(target,object->p_lists,safeflag);
    }
    else {
        /*    printf("case2."); */
        unify_pstlist_objects(target,((struct pst *)u)->p_lists, f,safeflag)
    }
    }
  else {
    object = find_pstitem(u,f);
    if (object != (struct pst_item *)NULL)
    {
        /*    printf("case3."); */
        unify_pstlist_objects(object,((struct pst *)t)->p_lists, e, safeflag
    }
    else {
        /*    printf("case4."); */
      target = record_pstobjects((struct pst *)t,e);
      unify_pstlist_objects(target,((struct pst *)u)->p_lists, f, safeflag);
    }
  }
          unify(((struct pst *)u)->p_var,f,((struct pst *)t)->p_var,e);
/*  Pterm(t,e);  NL; */

}

void unify_pst_extract(t,e,u,f) /* safe, extract pst unification */
struct pst *t,*u;               /* t may be changed */
struct pair *e,*f;
{
        struct pst_item *object,*target;
        struct eclause *nttbegin,*ntt,*ot,*tt;
        int i;

        target = find_pstitem(t,e);
        object = find_pstitem(u,f);
        if (target == (struct pst_item *)NULL)
                target = record_pstobjects((struct pst *)t,e);
        if (object == (struct pst_item *)NULL)
                object = record_pstobjects((struct pst *)u,f);
        nttbegin=(struct eclause *)NULL;
        for(ot=object->p_lists,tt=target->p_lists; ((ot!=NULL) && (tt!=NULL));)
        {
                i = Pred(Arg1(tt->e_form))->f_number -
                        Pred(Arg1(ot->e_form))->f_number;
                if (i == 0)
                {
                        safe_unify(tt->e_form,e,ot->e_form,f,1);
                        tt = tt->e_link;
                        ot = ot->e_link;
                }
                else if (i < 0) tt = tt->e_link;
                else
                {
                        if (nttbegin==NULL) nttbegin=ntt=ot;
                        else {
                                upush(&(ntt->e_link));
                                ntt->e_link = ot;
                                ntt = ot;
                        }
                        ot = ot->e_link;
                }
        }
        if (nttbegin != NULL) {
                upush(&(ntt->e_link));
                ntt->e_link = ot;
        }
        else nttbegin = ot;
        upush(&(target->p_lists));
        target->p_lists = nttbegin;
}

struct pst_item *record_pstobjects(t,e)
struct pst *t;
struct pair *e;
{
  struct pst_item *entry = psttable;
  struct term *tt = t->p_var;
  struct pair *p;

 down(p,tt,e);

 while(entry->p_link != (struct pst_item *)NULL) {
    if (p > entry->p_link->p_var) break;
    entry = entry->p_link;
  }
  upush(&(entry->p_link));
  entry->p_link = (struct pst_item *)
    Npst_item(p,(struct eclause *)NULL,entry->p_link);
  entry = entry->p_link;
  entry->p_lists = record_pstlists(t->p_lists,e);
  return(entry);
}

struct eclause *record_pstlists(ptt,e)
struct eclause *ptt;
struct pair *e;
{
  struct eclause *props, *pre;
```

```
  if (ptl == (struct eclause *)NULL) return(pl);
  pre = props = Npstobj(ptl->c_form, e, (struct eclause *)NULL, MEDIUM);
  for (ptl = ptl->c_link; ptl != (struct eclause *)NULL; ) {
    props->c_link =
      Npstobj(ptl->c_form, e, (struct eclause *)NULL, MEDIUM);
    props = props->c_link;
    ptl = ptl->c_link;
   }
  return(pre);
}

void unify_pstlist_objects(entry, ol, e, safeflag)
struct pst_item *entry;
struct eclause *ol;
struct pair *e;
int safeflag;                   /* SAFE or UNSAFE */
{
  int i, fnum;
  struct eclause *pl;

  if (ol==(struct eclause *)NULL) return;

  pl=entry->p_lists; /* pl must NOT be NULL */

/*  printf("unify_pstlist obj ");
  Peclause(pl);
  printf("    ");
  Peclause(ol);
  Nl;
*/

  if (pl == (struct eclause *)NULL) {
    upush(&(entry->p_lists));
    entry->p_lists=record_pstlists(ol,e);
    return;
  }

  i = Pred(Arg1(pl->c_form))->f_number - Pred(Arg1(ol->c_form))->f_number;
  if (i == 0) {
          if (safeflag == UNSAFE)
                  unify(pl->c_form,pl->c_env,ol->c_form,e);
          else
                  safe_unify(pl->c_form,pl->c_env,ol->c_form,e,NOEXTRACT);
          ol=ol->c_link;
  }
  else if (i > 0) {
    upush(&(entry->p_lists));
    entry->p_lists = Npstobj(ol->c_form,e,pl,MEDIUM);
    ol = ol->c_link;
    pl=entry->p_lists;
  }
 /*  else  goes on */

  while (ol != (struct eclause *)NULL) {
    fnum = Pred(Arg1(ol->c_form))->f_number;
    while (pl->c_link != (struct eclause *)NULL) {
     i = Pred(Arg1(pl->c_link->c_form))->f_number - fnum;
     if (i == 0) {
             if (safeflag== UNSAFE)
                     unify(pl->c_link->c_form,pl->c_link->c_env,ol->c_form,e);
             else
                     safe_unify(pl->c_link->c_form,
                             pl->c_link->c_env,ol->c_form,e,NOEXTRACT);
             break;
     }
     else if (i > 0) {
       upush(&(pl->c_link));
       pl->c_link = Npstobj(ol->c_form,e,pl->c_link,MEDIUM);
       break;
     }
     pl = pl->c_link;
    }

    if (pl->c_link == (struct eclause *)NULL) {
         upush(&(pl->c_link));
         pl->c_link = record_pstlists(ol,e);
         break;
       }
    else pl=pl->c_link;
    ol = ol->c_link;
  }
}


void unify_merge_psts(target,object,safeflag)
struct pst_item *target;
struct eclause *object;
int safeflag;                   /* SAFE or UNSAFE */
{
  int i, fnum;
  struct eclause *next, *pl;

  if (object==(struct eclause *)NULL) return;

  pl=target->p_lists;

  if (pl == (struct eclause *)NULL) {
    upush(&(target->p_lists));
    target->p_lists = object;
    return;
  }

  i = Pred(Arg1(pl->c_form))->f_number - Pred(Arg1(object->c_form))->f_number;
  if (i == 0) {
          if (safeflag == UNSAFE)
                  unify(pl->c_form,pl->c_env,object->c_form,object->c_env);
          else
                  safe_unify(pl->c_form,pl->c_env,
                          object->c_form,object->c_env,EXTRACT);
```

```
          object=object->c_link;
    }
    else if (i < 0) {
      upush(&(target->p_lists));
      target->p_lists =
        Npstobj(object->c_form,object->c_env,pl,MEDIUM);
      object = object->c_link;
      pl = target->p_lists;
    }
   /* else ques on */

    while (object != (struct eclause *)NULL) {
      fnum = Pred(Arg(object->c_form))->f_number;
      while (pl->c_link != (struct eclause *)NULL) {
        i = Pred(Arg(pl->c_link->c_form))->f_number - fnum;
        if (i == 0) {
                if (safeflag == UNSAFE)
                        unify(pl->c_link->c_form,pl->c_link->c_env,
                                object->c_form,object->c_env);
                else
                        safe_unify(pl->c_link->c_form,pl->c_link->c_env,
                                object->c_form,object->c_env,NOEXTRACT);
                break;
         }
        else if (i > 0) {
          next=pl->c_link;
          upush(&(pl->c_link));
          pl->c_link = object;
          upush(&(object->c_link));
          object->c_link = next;
          break;
          }
       pl=pl->c_link;
      }
     if (pl->c_link == (struct eclause *)NULL) {
          upush(&(pl->c_link));
          pl->c_link = object;
          return;
        }
     else pl=pl->c_link;
     object = object->c_link;
    }
}

struct pair *env(t,e)           /* from merge_pst_objects */
struct eclause *t;
struct pair *e;
{
        if (e==NULL) return(t->c_env);
        else return(e);
}

struct eclause *merge_pst_objects(target,e,object,f,safeflag)
struct eclause *target,*object;
struct pair *e,*f;
int safeflag;                   /* SAFE or UNSAFE */
{
        struct eclause *t,*o,*ntbegin=NULL,*nt;
        int i;

        if (target == NULL) return(object);
        if (object == NULL) return(target);
        for(t=target,o=object; ((t!=NULL) && (o!=NULL)); )
        {
                i = Pred(Arg(t->c_form))->f_number -
                        Pred(Arg(o->c_form))->f_number;
                if (i == 0)     /* same label */
                {
                        if(safeflag !=UNSAFE)
                                unify(t->c_form,env(t,e),o->c_form,env(o,f));
                        else
                                safe_unify(t->c_form,env(t,e),
                                        o->c_form,env(o,f),NOEXTRACT);
                        if (ntbegin == NULL)  ntbegin=nt=
                                Npstobj(t->c_form,env(t,e),NULL,MEDIUM);
                        else  nt->c_link =
                                Npstobj(t->c_form,env(t,e),NULL,MEDIUM);
                        t = t->c_link;
                        o = o->c_link;
                }
                else if (i < 0) /* t < o */
                {
                        if (ntbegin == NULL)  ntbegin nt=
                                Npstobj(t->c_form,env(t,e),NULL,MEDIUM);
                        else  nt->c_link =
                                Npstobj(t->c_form,env(t,e),NULL,MEDIUM);
                        t = t->c_link;
                }
                else            /* t > o */
                {
                        if (ntbegin == NULL)  ntbegin=nt=
                                Npstobj(o->c_form,env(o,e),NULL,MEDIUM);
                        else  nt->c_link =
                                Npstobj(o->c_form,env(o,e),NULL,MEDIUM);
                        o = o->c_link;
                }
        }
        if (t != NULL) nt->c_link = t;
        else if (o != NULL) nt->c_link = o;
        return(ntbegin);
}
```

```
/*----------------------------------------------------------------------
*              cu Prolog III (Constraint Unification Prolog)
*    Copyright: Institute for New Generation Computer Technology,Japan
*                            1989-91
------------------------------------------------------------------------*/
/*----------------------------------------------------------------------
*              << defsysp.c >>
*              (define system predicate entry)
------------------------------------------------------------------------*/

#define SYSPRED 1

#include "include.h"
#if SUN4 == 1
#include <sys/time.h>
#else
#if CPUTIME != 0
#include <sys/types.h>
#include <sys/times.h>
#endif
#endif

#define XF  0210
#define YF  0200
#define FX  0101
#define FY  0100
#define XFY 0110
#define YFX 0301
#define XFX 0111

#define Def1(F,N,A,P) (F = Nfunc(TYPE1SYS,N,A))->def.f_sysfunc=P
#define Def1Red(F,N,A,P) (F = Nfunc(TYPE1SYS_REDUCED,N,A)) ->def.f_sysfunc=P
#define Def2(F,N,A,P) (F = Nfunc(TYPE2SYS,N,A)) ->def.f_sysfunc=P
#define Def2Red(F,N,A,P) (F = Nfunc(TYPE2SYS_REDUCED,N,A)) ->def.f_sysfunc=P
#define Deftemp(F,N,A)  F = Nfunc(TEMPFUN,N,A)
#define Defatom(F,N) \
  ((F = Nterm(0,ETERNAL)) ->type.t_func = Nfunc(TYPE1SYS,N,0))

#define Npstobj(Head,Env,Tail,Flag)  Nsclause(Head,Env,Tail,Flag)

long OLD_TIME = 0L;

/* cf. TYPE1SYS : system functional predicate (has only one solution)
       TYPE2SYS : system non-functional predicate (has many solutions) */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  init_syspred()
   initialize system predicates : called by init_status()
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void init_syspred () {           /* initialize system predicates */
        void init_froles();

        defsyspred ();            /* define system predicate */
        init_atoms();             /* init atoms */
        init_operator();          /* init operator */
        init_category();          /* init cat() functro   */
        init_froles();            /* init components */
}

void defsyspred()        /*      define system embedded predicate        */
{
        Def1Red(ABOLISH_P,"abolish",2,abolish_pred); /* abolish of def */
        Def1Red(ARG_P,"arg",3,arg_pred);  /* arg(Pos,Term,Argument) */
        Def1Red(ASSERT_P,"assert",1,assert_pred); /* assert(PredName) */
        Def1Red(ASSERTA_P,"asserta",1,assert_pred);
        Def1Red(ASSERTZ_P,"assertz",1,assertz_pred);
        Def1Red(ASSERT_P,"assert",2,assert_pred); /* assert(Pred,Defs) */
        Def1Red(ASSERTA_P,"asserta",2,assert_pred);
        Def1Red(ASSERTZ_P,"assertz",2,assertz_pred);
        Def1Red(ASSERT_P,"assert",3,assert_pred); /* assert(Pred,Defs,Const) */
        Def1Red(ASSERTA_P,"asserta",3,assert_pred);
        Def2(ATTACH_P,"attach_constraint",1,attach_pred);
        Def1Red(ASSERTZ_P,"assertz",3,assertz_pred);
        /* cat(Pos,Form,Adjacent,Adjunct,Subcat,Sem) */
        Def1Red(CNAME_P,"condname",2, cname_pred); /* constraint name */
        Def2Red(CLAUSE_P,"clause",3, clause_pred); /* clause(head,body,const) */
        Def1Red(CLOSE_P,"close",1,close_pred);  /* file close */
        Def1Red(CMP_P,"compare",3,compare_pred);
        Def2Red(CONCAT_P,"concat",3, concat_pred);  /* string concatenate */
        Def1Red(CONCAT2_P,"concat2", 2, concat2_pred);
        Def1Red(COUNT_P,"count",1, count_pred); /* counter */
        Def1Red(CUNIFY,"unify",2, cunify_pred);
        Def1(CUT_P,"!", 0, cut_pred);
        Def1Red(DEFAULT_P,"default",3,default_pred);
        Def1Red(DIVSTR_P,"divstr",4,divstr_pred);
        Def1Red(EQUAL_P,"equal",2, equal_pred); /* = (check, substitution) */
        Def1Red(EQ_P,"eq",2, eq_pred);
        Def2(EXECUTE_P,"execute",1,or_pred);  /* execute(List) */
        Def1Red(FAIL_P,"fail", 0, NULL);  /* fail, forever */
        Def1Red(GENSYM_P,"gensym",2, gensym_pred);
        Def1Red(FUNCTOR_P,"functor",3, functor_pred); /* functor(a(X,Y),a,2) */
        Def1Red(GENSYM_P,"gensym",1, gensym_pred); /* gensym */
        Def1Red(GEQ_P,"geq",2,geq_pred);    /* >= */
        Def1Red(GREATER_P,"greater",2,greater_pred); /* > */
        Def1Red(HALT_P,"halt",0, halt_pred);
        Def2Red(ISOP_P,"isop",3,isop_pred);
        Def1Red(LEQ_P,"leq",2,leq_pred);   /* <= */
        Def1Red(LESS_P,"less",2,less_pred),   /* < */
        Def1Red(MAKELIST_P,"ml",2, makelist_pred); /* a(X,Y)=..[a,X,Y] */
        /* Def1(MODULAR_P,"modularize",2, cunify_pred);*/
        Def2Red(MEMB_P,"memb",2,memb_pred); /* system 'member' */
        Def1Red(MULTIPLY_P,"multiply",3, multiply_pred);
                /*multiply(X,Y,Z) is X*Y=Z */
        Def1Red(NAME_P,"name",2, name_pred);     /* array<->string */
        Def1Red(NL_P,"nl",0, nl_pred);  /* print CR */
        Def1Red(NL_P,"nl",1, nl_pred);  /* print CR */
        Def1Red(OP_P,"op",3,op_pred);      /* operator def */
        Def1Red(OPEN_P,"open",3,open_pred);  /* file open */
        Def2(OR_P,"or",2, or_pred);    /* or */
        Def2(OR_P,"or",3,or_pred);
        Def2(OR_P,"or",4,or_pred);
```

```
        Def2(OR_P,"or",5,or_pred);
        Def1(PCONSTRAINT_P ,"pcon",0, pcon_pred); /* print constraint */
        Def1Red(READ_P,"read",1,read_pred);  /* read TERM */
        Def1Red(READ_P,"read",2,read_pred);
        Def2Red(RETRACT_P,"retract",1,retract_pred); /* retract(Head) */
        Def2Red(RETRACT_P,"retract",2,retract_pred); /* retract(Head,Defs) */
        Def2Red(RETRACT_P,"retract",3,retract_pred); /* retract(H,D,Constr) */
        Def1Red(STAY_P,"stayflag",1,stay_pred); /* set stayflag */
        Def1Red(SEE_P,"see",1,see_pred);  /* input file open */
        Def1Red(SEEN_P,"seen",0,seen_pred);  /* input file close */
        Def1Red(STRLEN_P,"strlen",2,strlen_pred); /* length of string */
        Def1Red(STRCMP_P,"strcmp",3,strcmp_pred);
        Def1Red(SUBSTR_P,"substring",3,substr_pred); /* substring(Str,F,S) */
        Def1Red(SUBSTR_P,"substring",4,substr_pred); /* substring(Str,F,N,S) */
        Def1Red(SUM_P,"sum",3,sum_pred);       /*  sum(X,Y,Z) is X+Y=Z */
        Def1Red(TAB_P,"tab",0,tab_pred);         /* print tab */
        Def1Red(TAB_P,"tab",1,tab_pred);         /* print tab */
        Def1Red(TELL_P,"tell",1,tell_pred);  /* output file open */
        Def1Red(TOLD_P,"told",0,told_pred);  /* output file close */
        Def1Red(TREE_P,"tree",1,tree_pred);      /*  tree print */
        Def2Red(TRUE_P,"true",0,true_pred);  /* true, forever */
        Def1Red(UNBREAK_P,"unbreak",0,unbreak_pred); /* back to traceMode */
        Def1Red(VAR_P,"var",1, var_pred);        /*  var() pred */
        Def1Red(WRITE_P,"write",1,write_pred);  /* write TERM */
        Def1Red(WRITE_P,"write",2,write_pred);
        Def1Red(ABOMB_P,"atomicbomb",0,abomb_pred);
        Def1Red(PNAMES_P,"pnames",2,pnames_pred); /* get Property Names */
        Def1Red(PVALUE_P,"pvalue",3,pvalue_pred);  /* pvalue(PST,PNAME,VAL) */
        Def1Red(TYPE_P,"type",2,type_pred);  /* what type is it? */
        Def1Red(RESET_TIMER_P,"reset_timer",0,reset_timer_pred);
        Def1Red(TIMER_P,"timer",2,timer_pred);

        /* predicate not in hash-table */
        Deftemp(MODULAR_P,"modularize",2);
        Deftemp(INTEG_P,"integrate",2);
}

void init_atoms()
{
    Def1(CAT_P,"cat",6, NULL);  /*  category    */
    Def1(T_P,"t",3,NULL);       /*  three list t(M,L,R) */
    Def1(LIST,".",2,NULL);      /*  list  */
    NIL = Nterm(0,ETERNAL);     /* NIL = [] : end of list*/
    NIL->type.t_func = Nfunc(TEMPFUN,"NIL", 0);
    NIL->type.ident = ATOMIC_TYPE;
    (FAIL = Nterm(0,ETERNAL))->type.t_func = FAIL_P;
    Defatom(END_OF_FILE, "end_of_file");
    Anonymous_var = (struct term *)(new(var));
    Anonymous_var->type.ident = VAR_VOID_TYPE;
    ((struct var *)Anonymous_var)->v_name = "_";
    (Anonymous_env = new(pair))->p_body = NULL;
    MFAIL = Nclause(FAIL,(struct clause *)NULL,ETERNAL);
    Defatom(S_GLOBAL_VAR,"global var");
    Defatom(S_VAR,"var");
    Defatom(S_INTEGER,"integer");
    Defatom(S_FLOAT,"float");
    Defatom(S_STRING,"string");
    Defatom(S_FILE_POINTER,"file pointer");
    Defatom(S_PST,"pst");
    Defatom(S_PSTORU,"pst_proplist");
    Defatom(S_CLAUSE,"clause");
    Defatom(S_LIST,"list");
    Defatom(S_FUNCTOR,"functor");
    Defatom(S_ATOM,"atom");
    S_GREATER = Nterm(0,ETERNAL);
    S_GREATER->type.t_func = Nfunc(TYPESYS,">",0);
    S_LESS = Nterm(0,ETERNAL);
    S_LESS->type.t_func = Nfunc(TYPESYS,"<",0);
    S_EQ = Nterm(0,ETERNAL);
    S_EQ->type.t_func = Nfunc(TYPESYS,"=",0);
}

void init_operator()
{
        Defatom(XF_P, "xf");
        Defatom(YF_P, "yf");
        Defatom(FX_P, "fx");
        Defatom(FY_P, "fy");
        Defatom(XFX_P, "xfx");
        Defatom(XFY_P, "xfy");
        Defatom(YFX_P, "yfx");
        Def1(DEF_P, ":-",2,NULL);
        Def1(QUERY1_P, ":-",1,NULL);
        Def1(QUERY2_P, "?-",1,NULL);
        Def1Red(NOT_P, "not",1,not_pred);
        Def1(EQSIGN_P, "<=>",2, NULL);
        Def1Red(EQ2_P, "=",2,equal_pred);
        Def1Red(MKLIST_P, "=..",2,makelist_pred);
        Def1(CONSTRAINT_P, ";",2,NULL);
        Def1(CONSTRAINT2_P, "where",2,NULL);
        Def1Red(GREATER2_P, ">",2,greater_pred);
        Def1Red(GEQ2_P,">=",2,geq_pred);
        Def1Red(LESS2_P,"<",2,less_pred);
        Def1Red(LEQ2_P,"<=",2,leq_pred);
        Def1Red(EQUAL2_P,"==",2,eq_pred);
        Def1(PNAME_P,"/",2,NULL);   /* property/value */

        index_op(DEF_P, XFX ,1200);
        index_op(QUERY1_P, FX , 1200);
        index_op(QUERY2_P, FX, 1200);
        index_op(CONSTRAINT_P, YFX, 1200);
        index_op(CONSTRAINT2_P, YFX, 1200);
        index_op(EQSIGN_P, XFX, 1200);
        index_op(NOT_P, FY, 900);
        index_op(MKLIST_P, XFX, 700);
        index_op(GREATER2_P, XFX, 700);
        index_op(GEQ2_P, XFX, 700);
        index_op(LESS2_P, XFX, 700);
        index_op(LEQ2_P, XFX, 700);
```

```
        index_op(EQUAL2_P, XFX, 700);
        index_op(EQ2_P, XFX, 700);

        index_op(PNAME_P,XFY, 900);
      }

/* execution of system predicate:
   (t,e) : goal literal
   return value :
        SYSND :: 't' is not system pred.
        SYSTRUE :: (t,e) succeed
        SYSFAIL :: (t,e) fail
*/

/*---  -- initialize components of system predicates      --  */
struct component *NOT_VACUOUS;

void init_proles()
{
        void init_system_component();
        MEMORY_ALLOC(NOT_VACUOUS,component,ETERNAL);
        init_system_component(ARG_P,07);
        init_system_component(CLAUSE_P,06);
        init_system_component(CONCAT_P,07);
        init_system_component(CONCAT2_P,03);
        init_system_component(CONNF_P,01);
        init_system_component(EQ2_P,01);
        init_system_component(FUNCTOR_P,07);
        init_system_component(GENSYM_P,01);
        init_system_component(ISOP_P,07);
        init_system_component(MAKELIST_P,03);
        init_system_component(MEMB_P,02);
        init_system_component(MULTIPLY_P,07);
        init_system_component(NAME_P,01);
        init_system_component(STRLEN_P,02);
/*      init_system_component(SUM_P,07); */
        init_system_component(PNAMES_P,02);
        init_system_component(PNAME_P,02);
        init_system_component(TYPE_P,02);
}

void init_system_component(f,a) /* initialize system component */
struct func *f;
unsigned long a;                /* non vacuous bit pattern */
{
        register int i,arity;

        for (arity = f->f_arity, i = 0; i < arity; i++, a >>= 1)
                if ((a & 01) != 0) Component(f,i) = NOT_VACUOUS;
}

int system_function(t,e,n)      /* solve system functional predicate */
struct term *t;
struct pair *e;
struct node *n;
{
        SYSFUNC comp;
        struct func *f;

        f = t->type.t_func;
        comp = f->def_f_systune;
        if (comp == NULL) {
          if (f == FAIL_P) return(SYSFAIL);
          if (Handle_Undefined == TRUE) {
            sprintf(mbuf,">>> %s <<< is UNDEFINED!",f->f_name);
            error(mbuf);
          }
          else return(SYSFAIL);
        }
        if (isreduced(f)) return((*comp)(t,e));
        else return((*comp)(t,e,n));
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  system_pred()
   process system predicates
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int system_pred(t,e,n,m,status) /* system (multi valued) predicate */
struct term *t;
struct pair *e;
struct node *n, *m;
int status;                     /* search status UP,DOWN,BACKTRACK */
{
        SYSFUNC comp;
        struct func *f;

        f = t->type.t_func;
        comp = f->def_f_systune;
        if (comp == NULL) {
          if (Handle_Undefined == TRUE) {
            sprintf(mbuf,">>> %s <<< is UNDEFINED!",f->f_name);
            error(mbuf);
          }
          else return(SYSFAIL);
        }
        if (isreduced(f)) return((*comp)(t,e,n,status));
        else return((*comp)(t,e,n,m,status));
}

int cut_pred(t,e,n)
struct term *t;
struct pair *e;
struct node *n;
{
  if (n->n_link != NULL) n->n_link->n_set = NULL; /* OR cut */
  n->n_last = n->n_link;
  Last_BT = n->n_link;
  return(SYSTRUE);
}
```

```
/* always fail */ /*
int fail_pred(t,e)
struct term *t;
struct pair *e;
{
        return(SYSFAIL);
} */


int halt_pred(t,e)
struct term *t;
struct pair *e;
{
  quit_prolog();
  return(SYSTRUE);
}

int abomb_pred(t,e)
struct term *t;
struct pair *e;
{
        tprint0("\n Quit on prolog !!!!\n");
        exit(1);
}

int true_pred(t,e,n,status)
struct term *t;
struct pair *e;
struct node *n;
int status;
{
  n->n_set = EMPTY_DEF;
  return(SYSTRUE);
}

int op_pred(t,e)
register struct term *t;
register struct pair *e;
{
  struct term *tt;
  register struct pair *p, *ee;
  register struct func *f;
  int prec, otype;

  tt = Arg1(t); ee = e;
  down(p,tt,ee);
  if (! is_int(tt))
    error_detail(t,e,
                 "op/3: Illegal Argument -- Precedence should be integer");
  prec = num_value(tt);
  if ((prec < 0) || (prec > 1000))
    error_detail(t,e,
        "op/3: Illegal Argument -- Precedence should be from 0 to 1000");
  tt = Arg2(t); ee = e;
  down(p,tt,ee);
  if ((p != NULL) || (! is_functor(tt)) || (! isatom(tt)))
    error_detail(t,e,"op/3: Illegal Argument as type");

  f = Pred(tt);
  if (f == Pred(XF_P)) otype = XF;
  else if (f == Pred(YF_P)) otype = YF;
  else if (f == Pred(FX_P)) otype = FX;
  else if (f == Pred(FY_P)) otype = FY;
  else if (f == Pred(XFX_P)) otype = XFX;
  else if (f == Pred(XFY_P)) otype = XFY;
  else if (f == Pred(YFX_P)) otype = YFX;
  else error_detail(t,e,"op/3:Illegal Argument as type");

  tt = Arg3(t);
  down(p,tt,e);
  while (is_list(tt)) {
    t = head_of_list(tt);
    down(p,t,e);
    if ((p != NULL) || (! is_functor(t)) || (! isatom(t)))
      error_detail(t,e,
                   "op/3: Illegal Argument -- operator should be functor");
    index_op(t->type.t_func, otype, prec);
    tt = tail_of_list(tt);
    down(p,tt,e);
  }
  if (tt != NIL)
    if (is_functor(tt) && isatom(tt))
      index_op(tt->type.t_func, otype, prec);
    else
      error_detail(t,e,
                   "op/3: Illegal Argument -- operator should be functor");
  return(SYSTRUE);
}


void index_op(t, type, prec)
register struct func *f;
int type, prec;
{
  register struct operator *o, *olast;

  if ((type & INFIX) == INFIX) { /* INFIX operator */
        if (f->f_arity != 2) f = Predicate(f->f_name,2);
      }
  else
        if (f->f_arity != 1) f = Predicate(f->f_name,1);

  for(olast = o = o_list; o!=NULL; olast = o, o=o->o_link)
    if ((f == o->o_func) && ((type & INFIX) == (o->o_type & INFIX))) {
      if (prec==0)
        if (o==o_list) o_list=o->o_link;
        else olast->o_link=o->o_link;
      else o->o_prec=prec;
      break;
    }
```

```
  if ((o == NULL) && (prec != 0)) {
    o=new(operator);
    o->o_func = f;
    o->o_prec = prec;
    o->o_type = type;
    o->o_link = o_list;
    o_list = o;
  }
}

int isop_pred(t,e,n,status)
struct term *t;
struct pair *e;
struct node *n;
int status;
{
  struct operator *o;
  struct ustack *usave;
  int *hsave;
  struct pair *esave;
  struct term *tt;

  if (status == BACKTRACK)
    o = (struct operator *)n->n_set;
  else o = o_list;

  usave = usp;
  hsave = hp;
  esave = ep;

  while (o != NULL) {
    tt = Mterm(0,TEMPORAL);
    tt->type.l_func = Predicate(o->o_func->f_name,0);
    if (tunify(Arg1(t),e,tt,(struct pair *)NULL,0) == FALSE)
    {
            undo(usave); hp = hsave; ep = esave; o=o->o_link;
            continue;
    }
    tt = Mnum_val((float)o->o_prec,TEMPORAL);
    if (tunify(Arg1(t),e,tt,(struct pair *)NULL,0) == FALSE)
    {
            undo(usave); hp = hsave; ep = esave; o=o->o_link;
            continue;
    }
    switch (o->o_type) {
    case FX: tt = FX_P; break;
    case FY: tt = FY_P; break;
    case XF: tt = XF_P; break;
    case YF: tt = YF_P; break;
    case XFX: tt = XFX_P; break;
    case XFY: tt = XFY_P; break;
    case YFX:
    default : tt = YFX_P;
    }
    if (tunify(Arg2(t),e,tt,(struct pair *)NULL,0) == FALSE)
    {
            undo(usave); hp = hsave; ep = esave; o=o->o_link;
            continue;
    }
    n->n_set = (struct set *)o->o_link;
    return(SYSTRUE);
  }
  return(SYSFAIL);
}

int not_pred(t,e)
struct term *t;
struct pair *e;
{
  struct node *goal;
  int *hsave = hp;
  struct pair *esave;
  struct ustack *usave = usp;
  struct term *tt = Arg1(t);
  int refute();

  esave = ep;
  if (is_clause(tt))
    goal = Newnode((struct clause *)tt,(struct eclause *)NULL,
                   e,(struct node *)NULL,(struct node *)NULL);
  else if (is_functor(tt))
    goal = Newnode(Neclause(tt,(struct clause *)NULL,TEMPORAL),
                   (struct eclause *)NULL, e,(struct node *)NULL,
                   (struct node *)NULL);
  else error_detail(t,e,"not/1: Illegal Argument");

  if (refute(goal,goal,(FWN)==FALSE) return(SYSTRUE);
  /* else */
  undo(usave);
  hp = hsave;
  ep = esave;
  return(SYSFAIL);
}

int unbreak_pred(t,e)
struct term *t;
struct pair *e;
{
        longjmp(unbreak_reset,1);
}

/* pnames([a/1,b/2,c/X], Y) -> Y=[a,b,c] */
int pnames_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *ps;
  register struct eclause *pl;
  struct pst_item *target;
```

```
  struct pair *pp, *ee;
  struct term *ls = NIL;

  ps = Arg1(t);
  ee = e;
  down(pp, ps, ee);
  if (! is_pst(ps)) error_detail(t,e,"pname=1/2: 1st arg is not PST");

  target = find_pstitem(ps,ee);
  if (target != (struct pst_item *)NULL)
    pl = target->p_lists;
  else pl = ((struct pst *)ps)->p_lists;

  while (pl != (struct eclause *)NULL) {
    ls = (struct term *)Nlist(Arg1(pl->e_form), (struct clause *)ls,TEMPORAL);
    pl = pl->e_link;
  }

  return(equalpred(Arg2(t),e,ls,(struct pair *)NULL));
}

/* pvalue([a/1,b/2,c/x],b,Z) -> Z=2 */
int pvalue_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *ps;
  register struct pair *pp;
  register struct eclause *pl;
  struct term *pn;
  struct pair *ee, *etemp;
  struct pst_item *target;
  int f,i;

  ps = Arg1(t); pn = Arg2(t);
  ee = e;
  down(pp, ps, ee);
  if (! is_pst(ps)) error_detail(t,e,"pvalue/3: 1st arg is not PST");
  etemp = e;
  down(pp, pn, etemp);
  if ((! is_functor(pn))|| (pn->t_arity != 0))
    error_detail(t,e,"pvalue/3: 2nd arg is not ATOM");
  f = Pred(pn)->f_number;

  target = find_pstitem(ps,ee);
  if (target != (struct pst_item *)NULL) {
    pl = target->p_lists;
    while(pl != (struct eclause *)NULL) {
      i = Pred(Arg1(pl->e_form))->f_number - f;
      if (i==0)
        return(equalpred(Arg3(t),e, Arg2(pl->e_form),pl->e_env));
      if (i < 0) return(SYSFAIL);
      pl = pl->e_link;
    }
  }
  else {
    pl = ((struct pst *)ps)->p_lists;
    while (pl != (struct eclause *)NULL) {
      i = Pred(Arg1(pl->e_form))->f_number - f;
      if (i==0)
        return(equalpred(Arg3(t),e,Arg2(pl->e_form),ee));
      if (i < 0) return(SYSFAIL);
      pl = pl->e_link;
    }
  }
  return(SYSFAIL);
}

int default_pred(t,e)
struct term *t;
struct pair *e;
{
  struct term *ps, *templ, *dfs;
  struct pair *p, *ee, *et, *ed;
  static char *message = "default/3: %s arg is not PST";

  ps = Arg(t,0); templ = Arg(t,1); dfs = Arg(t,2);
  et = ed = ee = e;
  down(p,templ,et);
  if (! is_pst(templ)) {
    sprintf(nbuf,message,"2nd");
    error_detail(t,e, nbuf);
  }
  down(p,dfs,ed);
  if (! is_pst(dfs)) {
    sprintf(nbuf,message,"3rd");
    error_detail(t,e,nbuf);
  }
  down(p,ps,ee);
  if (! is_pst(ps)) {
    sprintf(nbuf,message,"1st");
    error_detail(t,e,nbuf);
  }

  if (subsume(ps, ee, templ, et, FALSE)==FALSE)
    return(SYSFAIL);
  pst_add_unify(ps,ee,dfs,ed);
  return(SYSTRUE);
}

int subsume(t, e, u, f, flag)
register struct term *t, *u;
register struct pair *e, *f;
int flag;       /* TRUE if Var subsumes everything, otherwise FALSE */
{
  register struct pair *p, *q;
  int i, j;

  down(p, t, e);
  down(q, u, f);
```

```
  if(p != NULL)
        if ((q != NULL) || (flag == TRUE)) return(TRUE);
        else return(FALSE);
  if (q != NULL) return(FALSE);

  switch (u->type_ident) {
    case ATOMIC_TYPE: /* t,u: atomic (string,num,quote) */
        if ((t==u) || (atomic_equal(u,t))) return(TRUE);
        else return(TRUE);
    case LIST_TYPE:
    case CONST_LIST_TYPE:
        if (is_list(t))
           if (subsume(head_of_list(t),e,head_of_list(u),f,flag)==TRUE)
              return(subsume(tail_of_list(t),e,tail_of_list(u),f,flag));
        return(FALSE);
    case CLAUSE_TYPE:
        if (is_clause(t)) {
           while ((t != NULL) && (u != NULL)) {
                if (subsume(((struct clause *)t)->c_form,e,
                            ((struct clause *)u)->c_form,f,flag) == FALSE)
                     return(FALSE);
                t=(struct term *)((struct clause *)t)->c_link;
                u=(struct term *)((struct clause *)u)->c_link;
                }
                if (t == u) return(TRUE);
              }
         return(FALSE);
        case PST_TYPE:
          if (is_pst(t)) return(subsume_pst(t,e,u,f,flag));
          return(FALSE);
        default : /* functor */
          if(Pred(t) == Pred(u)) {/* t,u: complex term */
            for(i = 0, j = Pred(t)->f_arity; i < j; i++) {
              if (subsume(Arg(t,i), e, Arg(u,i), f,flag) == FALSE)
                return(FALSE);
              }
            return(TRUE);
          }
          return(FALSE);
        }
}

int subsume_pst(t,e,u,f,flag)
register struct term *t,*u;
register struct pair *e,*f;
int flag;
{
  struct pst_item *target, *object;

  target = find_pstitem(t,e);
  if (target == (struct pst_item *)NULL)
        target = record_pstobjects((struct pst *)t,e);
  object = remove_pstitem(u,f);
  if (object != (struct pst_item *)NULL)
        return(subsume_pstlist(target->p_lists,object->p_lists,
                               (struct pair *)NULL,flag));
  return(subsume_pstlist(target->p_lists,((struct pst *)u)->p_lists, f,flag));
}

int subsume_pstlist(x,y,e,flag)
struct eclause *x,*y;
struct pair *e;
int flag;
{
  int i,fnum;
  while (y != (struct eclause *)NULL) {
    fnum = Pred(Arg1(y->c_form))->f_number;
    while (x->c_link != (struct eclause *)NULL) {
      i = Pred(Arg1(x->c_form))->f_number - fnum;
      if (i == 0) {
         if (e != (struct pair *)NULL) {
                if (subsume(x->c_form,x->c_env,y->c_form,e,flag)==FALSE)
                        return(FALSE);
                }
         else {
           if (subsume(x->c_form,x->c_env,y->c_form,y->c_env,flag)==FALSE)
                return(FALSE);
         }
         break;
      }
      else if (i > 0) return(FALSE);
      x = x->c_link;
    }
    y = y->c_link;
    x = x->c_link;
  }
  return(TRUE);
}

void pst_add_unify(t,e,u,f)
register struct term *t,*u;
register struct pair *e,*f;
{
  struct pst_item *target, *object;

  target = find_pstitem(t,e);
  if (target == (struct pst_item *)NULL)
        target = record_pstobjects((struct pst *)t,e);
  object = remove_pstitem(u,f);
  if (object != (struct pst_item *)NULL)
     pst_add_unify_sub(target,object->p_lists,(struct pair *)NULL);
  else pst_add_unify_sub(target,((struct pst *)u)->p_lists, f);
}

void pst_add_unify_sub(entry, ol, e)
struct pst_item *entry;
struct eclause *ol;
struct pair *e;
{
  int i, fnum;
```

```
  struct eclause *pl;

  if (ol==(struct eclause *)NULL) return;

  pl=entry->p_lists;

  if (pl == (struct eclause *)NULL) {
    upush(&(entry->p_lists));
    entry->p_lists=record_pstlists(ol,e);
    return;
  }

  i = Pred(Arql(pl->c_form))->f_number - Pred(Arql(ol->c_form))->f_number;
  if (i == 0) ol = ol->c_link;
  else if (i > 0) {
    upush(&(entry->p_lists));
    entry->p_lists = Npstobj(ol->c_form,e,pl,MEDIUM);
    ol = ol->c_link;
    pl=entry->p_lists;
  }

  while (ol != (struct eclause *)NULL) {
    fnum = Pred(Arql(ol->c_form))->f_number;
    while (pl->c_link != (struct eclause *)NULL) {
      i = Pred(Arql(pl->c_link->c_form))->f_number - fnum;
      if (i == 0) break;
      else if (i > 0) {
        upush(&(pl->c_link));
        pl->c_link = Npstobj(ol->c_form,e,pl->c_link,MEDIUM);
        break;
      }
      pl = pl->c_link;
    }

    if (pl->c_link == (struct eclause *)NULL) {
        upush(&(pl->c_link));
        pl->c_link = record_pstlists(ol,e);
        break;
      }
    else pl=pl->c_link;
    ol = ol->c_link;
  }
}

int type_pred(t,e)
struct term *t;
struct pair *e;
{
  struct term *tt,*type;
  struct pair *p, *el = e;

  tt = Arg(t,0);
  down(p,tt,el);

  switch(tt->type.ident) {
    case VAR_VOID_TYPE:
    case VAR_PST_TYPE:
    case VAR_GLOBAL_TYPE:
      return(equalpred(Arg(t,1),e,S_VAR,(struct pair *)NULL));
    case ATOMIC_TYPE:
      switch(tt->t_arity) {
        case FLOAT_NUM:
          return(equalpred(Arg(t,1),e,S_FLOAT,(struct pair *)NULL));
        case INT_NUM:
          return(equalpred(Arg(t,1),e,S_INTEGER,(struct pair *)NULL));
        case STRING:
          return(equalpred(Arg(t,1),e,S_STRING,(struct pair *)NULL));
        default:
          return(equalpred(Arg(t,1),e,S_FILE_POINTER,(struct pair *)NULL));
        }
    case PST_TYPE:
          return(equalpred(Arg(t,1),e,S_PST,(struct pair *)NULL));
    case ECLAUSE_TYPE:
          return(equalpred(Arg(t,1),e,S_PSTORJ,(struct pair *)NULL));
    case CLAUSE_TYPE:
          return(equalpred(Arg(t,1),e,S_CLAUSE,(struct pair *)NULL));
    case LIST_TYPE:
    case CONST_LIST_TYPE:
          return(equalpred(Arg(t,1),e,S_LIST,(struct pair *)NULL));
    default :
      if (t->t_arity == 0)
         return(equalpred(Arg(t,1),e,S_ATOM,(struct pair *)NULL));
      return(equalpred(Arg(t,1),e,S_FUNCTOR,(struct pair *)NULL));
    }
}

int reset_timer_pred(t,e)
struct term *t;
struct pair *e;
{
#if SUN4 == 1
  OLD_TIME = clock();
#else
#if CPUTIME == 0
  OLD_TIME = 0L;
#else
struct tms TIMES;                /* cf. times() */
  times(&TIMES);
  OLD_TIME = TIMES.tms_stime + TIMES.tms_utime;
#endif
#endif

  CONSTRAINT_HANDLING_TIME = 0L;
  return(SYSTRUE);
}

int timer_pred(t,e)
struct term *t;
struct pair *e;
{
```

```
#if CPUTIME != 0
  struct tms TIMES;
#endif
  static char *emsg = "time*/2: %s is not VAR";
  register struct pair *p1, *p2, *ee;
  struct term *t1, *t2;

  ee = e;
  t1 = Arg1(t); down(p1,t1,ee);
  if (p1 == NULL) {
    sprintf(nbuf,emsg,"1st");
    error_detail(t,e,nbuf);
  }
  ee = e;
  t2 = Arg2(t); down(p2,t2,ee);
  if (p2 == NULL) {
    sprintf(nbuf,emsg,"2nd");
    error_detail(t,e,nbuf);
  }
#if SUN4 == 1
  t1 = Num_val(((float)(clock())-OLD_TIME))/1000000.0,TEMPORAL);
  t2 = Num_val(((float)CONSTRAINT_HANDLING_TIME)/1000000.0,TEMPORAL);
#else
#if CPUTIME == 0
  t1 = t2 = Num_val(0.0,TEMPORAL);
#else
  times(&TIMES);
  t1 = Num_val(((float)(TIMES.tms_stime+TIMES.tms_utime-OLD_TIME))/CPUTIME.0,
               TEMPORAL);
  t2 = Num_val(((float)CONSTRAINT_HANDLING_TIME)/CPUTIME.0,TEMPORAL);
#endif
#endif
  upush(&(p1->p_body)); upush(&(p1->p_env));
  upush(&(p2->p_body)); upush(&(p2->p_env));
  p1->p_body = t1;      p1->p_env = (struct pair *)NULL;
  p2->p_body = t2;      p2->p_env = (struct pair *)NULL;
  return(SYSTRUE);
}

int stay_pred(t,e)
register struct term *t;
register struct pair *e;
{
  struct term *t1, *t2;
  register struct pair *p, *ee;
  register struct func *f;
  int prec, otype;

  t1 = Arg1(t); ee = e;
  down(p,t1,ee);

  if ((p != NULL) && (! is_functor(t1)) && (! isatom(t1)))
    error_detail(t,e,"stayflag/3: Illegal Argument as functor");

  tt = Arg2(t); ee = e;
  down(p,tt,ee);
  if (! is_int(tt))
    error_detail(t,e,
                 "stayflag/3: Illegal Argument as Arity");

  f = funcsearch(Pred(t1),t_name,(int)num_value(tt));
  if (f == NULL) error_detail(t,e,"stayflag/3: No such a predicate");

  tt = Arg1(t); ee = e;
  down(p,tt,ee);
  if (p != NULL) { /* 2nd arg is a variable */
    if (((f->f_mark) & NON_UNFOLDABLE) != 0) {
      if (((f->f_mark) & STAY_IF) == 0) { /* stay if false */
        t1 = Nterm(0,TEMPORAL);
        Pred(t1) = FAIL_P;
        return(equalpred(tt,ee,t1,(struct pair *)NULL));
      }
      else { /* stay if true */
        t1 = Nterm(0,TEMPORAL);
        Pred(t1) = TRUE_P;
        return(equalpred(tt,ee,t1,(struct pair *)NULL));
      }
    }
    return(SYSFAIL);
  }

  if (Pred(tt) == TRUE_P)
    f->f_mark |= STAY_IF_TRUE_PRED;
  else if (Pred(tt) == FAIL_P)
    f->f_mark |= STAY_IF_FALSE_PRED;
  else error_detail(t,e,"stayflag/3: Illegal Argument as TRUE/FAIL");
  return(SYSTRUE);
}
```

```
/*------------------------------------------------------------------------
*               cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                       1989--91
------------------------------------------------------------------------ */
/*------------------------------------------------------------------------
*               << syspred1.c >>
*               (system predicates No.1)
------------------------------------------------------------------------ */

#include "include.h"

/* for LtoC(), CtoL() pred  */
#define FROM_NAME 1
#define FROM_CONC 0

int memb_pred(t,e,n,status)     /* system 'member' pred */
struct term *t;
struct pair *e;
struct node *n;
int status;
{
        register struct term *tt;
        struct ustack *usave;
        int *hsave;
        struct pair *esave;
        register struct pair *p,*pp,*ee;

        if (status != BACKTRACK)
          {
            pp = Kenv(1);
            n->n_hp = hp;
            n->n_ep = ep;
            n->n_usp = usp;
            tt = Arg2(t);
            ee = e;
          }
        else
          {
            pp = (struct pair *)n->n_set;
            tt = pp->p_body;
            ee = pp->p_env;
          }
        down(p,tt,ee);

        usave = usp;
        hsave = hp;
        esave = ep;
        while(tt != NIL)
        {
                if (! is_list(tt)) return(SYSFAIL);
                if (tunify(Arg1(t),e,head_of_list(tt),ee,0) == FALSE)
                {
                        undo(usave);
                        hp = hsave;
                        ep = esave;
                        tt = tail_of_list(tt);
                        down(p,tt,ee);
                        continue;
                }
                pp->p_body = tail_of_list(tt);
                pp->p_env = ee;
                n->n_set = (struct set *)pp;
                return(SYSTRUE);
        }
        return(SYSFAIL);
}

int or_pred(t,e,n,m,status)
struct term *t;
register struct pair *e;
struct node *n, *m;
int status;
{
  register struct term *tt;
  register struct pair *e0;
  struct pair *p;
  struct clause *c0;
  struct clause *convert_list_to_clause();
  int arity, next = 0;

  if (status == BACKTRACK)
    next = (int)n->n_set;

  tt = Arg(t,next+1);
  e0 = e; down(p,tt,e0);

  if ((arity = t->t_arity) < 0) arity = -arity;
  n->n_set = (next < arity) ? (struct set *)next : NULL;

  if (p != NULL) {
        sprintf(nbuf,"or*/%d: %d th argument is real VAR",arity,next+1);
        error_detail(t,e,nbuf);
  }
  else if ((tt == NIL) || (tt == NULL)) return(SYSTRUE);

  if (is_list(tt)) {
    sprintf(nbuf,"or*/%d: %d th argument is not List",arity, (next+1));
    e0 = convert_list_to_clause(t,e,tt,e0,&p,nbuf);
  }
  else {
    p = e0;
    if (! is_clause(tt))
      e0 = Nclause(tt,(struct clause *)NULL,TEMPORAL);
    else  e0 = (struct clause *)tt;
    }

  m->n_clause = c0;
  m->n_env = p;
  m->n_usp = usp;
```

```
  m->n_hp = hp;
  m->n_ep = ep;
  m->n_set = init_set(m);
  return(SYSTRUE);
}

struct clause *convert_list_to_clause(t,e,tt,ee,p,emsg)
struct term *t, *tt;
struct pair *e, *ee, **p;
char *emsg;
{
    struct clause *c, *cc;
    register struct pair *pp;

    v_number = 0;  v_list = NULL;
    *p = Nenv(0);
    c = cc = Nclause(NULL,(struct clause *)NULL,TEMPORAL);

    while(1) {
      if (isconst(head_of_list(tt))) cc->c_form = head_of_list(tt);
      else {
        pp = Nenv(1);
        cc->c_form = Nvar(Anonymous_VarName,TEMPORAL);
        pp->p_body = head_of_list(tt);
        pp->p_env = ee;
      }

      tt = tail_of_list(tt);
      down(pp,tt,ee);
      if ((tt == NIL) || (tt == NULL)) break;
      cc->c_link = Nclause(NULL, (struct clause *)NULL, TEMPORAL);
      cc = cc->c_link;
    }
    return(c);
}

int read_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *tt, *target;
  register struct pair *p, *ee;
  FILE *filep;
  int arity;

  if ((arity = t->t_arity) < 0) arity = -arity;
  filep = fp;
  if (arity == 2) {
    tt = Arg2(t);
    ee = e;
    down(p,tt,ee);
    if (! is_file(tt)) error("read*/2: Illegal file pointer");
    fp = filep_value(tt);
    if (! is_readable(fp))
      {
        fp = filep;
        error("read*/2: file not open");
      }
  }

  v_number = 0;  v_list = NULL;
  p_number = 0;
  advance;
  if (check(EOF)){
    target = END_OF_FILE;
    fclose(fp);
    }
  else {
    target = Sterm(1200,TEMPORAL);
    if (tokentype!=FULLSTOP) {
        error_detail(target,(struct pair *)NULL,"Syntax error --- . expected");
        }
    skipline;
  }
  fp = filep;
  ee = Nenv(v_number+p_number);
  return(equalpred(Arg1(t),e,target,ee));
}

#define SPECIFIED 0
#define INPUT     1
#define OUTPUT    2

int open_pred(t,e)
struct term *t;
struct pair *e;
{
  return(file_open_pred(t,e,SPECIFIED));
      }

int see_pred(t,e)
struct term *t;
struct pair *e;
{
  return(file_open_pred(t,e,INPUT));
      }

int tell_pred(t,e)
struct term *t;
struct pair *e;
{
    return(file_open_pred(t,e,OUTPUT));
        }

int file_open_pred(t,e,openmode)
register struct term *t;
register struct pair *e;
int openmode;
{
  static char *emsg = "open/3: Illegal argument --- should not be variable";
```

```
  register struct pair *p, *ee;
  register struct term *tt;
  char *mode, *fname;
  FILE *filep, *fopen();

  tt = Arg1(t);
  ee = e;
  down(p,tt,ee);
  if (p != NULL) error_detail(t,e,emsg);

  if (is_string(tt)) fname=str_value(tt);
  else if (!is_atomic(tt)) fname=tt->type.t_func->f_name;
  else error_detail(t,e,"open/1: Illegal file name");

  switch (openmode) {
  case INPUT:
    mode = "r";
    break;
  case OUTPUT:
    mode = "w";
    break;
  case SPECIFIED:
    tt = Arg2(t);
    ee = e;
    down(p,tt,ee);
    if (p != NULL) error_detail(t,e, emsg);

    mode = (is_string(tt)) ? str_value(tt) : tt->type.t_func->f_name;
    if (((mode[0] != 'r') && (mode[0] != 'w')) || mode[1] != '\0') {
     sprintf(nbuf,"open/3: Illegal mode >> %s << should be 'r' or 'w'",mode);
     error(nbuf);
        }
  }
  if ((filep = fopen(fname, mode)) == NULL)
    error("open/3: can't open the file");
  switch (openmode) {
    case INPUT: fp=filep; return(SYSTRUE);
    case OUTPUT: wfp=filep; return(SYSTRUE);
    }
  tt = NTerm(0,TEMPORAL);
  tt->type.ident = FILE_TYPE;
  tt->tag.s_value = (char *)filep;
  return(equalpred(Arg3(t),e,tt,(struct pair *)NULL));
}

int seen_pred(t,e)
struct term *t;
struct pair *e;
{
    FILE *f = wfp;
    if (fp != stdin) fclose(fp);
        else {
                wfp = stderr;
                tprint0("Warning: no file is opened for input\n");
                wfp = f;
        }
    fp = stdin;
    return(SYSTRUE);
}

int told_pred(t,e)
struct term *t;
struct pair *e;
{
    if (wfp != stdout) fclose(wfp);
        else {
                wfp = stderr;
                tprint0("Warning: no file is opened for output\n");
                wfp = stdout;
        }
    wfp = stdout;
    return(SYSTRUE);
}

int close_pred(t,e)
register struct term *t;
register struct pair *e;
{
  FILE *filep;
  register struct pair *p;

  t = Arg1(t);
  down(p,t,e);
  if (! is_file(t)) error("close/1: Illegal argument");
  filep = filep_value(t);
  if ((filep == stdin) || (filep == stdout))
    error("close/1: stdin/stdout cannot be closed!");
  fclose(filep);
  return(SYSTRUE);
}

int pcon_pred(t,e,n)    /* print constraint */
struct term *t;
struct pair *e;
register struct node *n;
{
        Peclause(n->n_constraint);
        return(SYSTRUE);
}

int attach_pred(t,e,n,m,status) /* attach constraints */
struct term *t;
struct pair *e;
struct node *n,*m;
int status;
{
  struct pair *p, *ee;
  struct term *tt;
  register struct clause *c;
  struct eclause *ec;
```

```
  static char *emesg = "attach_constraint/1: Illegal Argument";
  struct clause *convert_list_to_clause();

  tt = Arg1(t);
  ee = e;
  down(p,tt,ee);
  if (is_list(tt)) {
    c = convert_list_to_clause(t,e,tt,ee,&p,emesg);
  }
  else if (is_clause(tt)) {
    c = (struct clause *)tt;
    p = ee;
  }
  else if (tt==NIL) return(SYSTRUE);
  else if (is_functor(tt)) {
    c = Nclause(tt, (struct clause *)NULL, TEMPORAL);
    p = ee;
  }
  else error_detail(t,e,emesg);

  ec = transform(m->n_constraint, c, p);
  if (ec == (struct eclause *)SFAIL)
    return(SYSFAIL);
  upush(&(m->n_constraint));
  m->n_constraint=ec;
  return(SYSTRUE);
}

int cunify_pred(t,e) /*  c.u.  unify() */
register struct term *t;
register struct pair *e;
{
        if (cu(t,e) != FALSE )
                return(SYSTRUE);        /* success */
        else
                return(SYSFAIL);        /* fail */
}

int write_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct pair *p, *ee;
  register struct term *tt;
  FILE *filep;
  int arity;

  if ((arity = t->t_arity) < 0) arity = -arity;
  filep = wfp;
  if (arity == 2) {
    tt = Arg2(t);
    ee = e;
    down(p,tt,ee);
    if (! is_file(tt)) error("write*/2: Illegal file pointer");
    wfp = filep_value(tt);
    if (! is_writable(wfp))
      {
        wfp = filep;
        error("write*/2: file not open");
      }
  }

  tt = Arg1(t);
  down(p,tt,e);
  if (is_string(tt)) fprintf("%s",str_value(tt))
  else Pterm(tt, e);
  wfp = filep;
  return(SYSTRUE);
}

int nl_pred(t,e)
register struct term *t;
register struct pair *e;
{
  register struct pair *p;
  FILE *filep;
  int arity;

  filep = wfp;
  if ((arity = t->t_arity) < 0) arity = -arity;
  if (arity != 0) {
    t = Arg1(t);
    down(p,t,e);
    if (! is_file(t)) error("nl*/1: Illegal file pointer");
    wfp = filep_value(t);
    if (! is_writable(wfp))
      {
        wfp = filep;
        error("nl*/2: file not open");
      }
  }

  NL;
  wfp = filep;
  return(SYSTRUE);
}

int tab_pred(t,e)
register struct term *t;
register struct pair *e;
{
  register struct pair *p;
  FILE *filep;
  int arity;

  filep = wfp;
  if ((arity = t->t_arity) < 0) arity = -arity;
  if (arity != 0) {
    t = Arg1(t);
    down(p,t,e);
```

```
    if (! is_file(t)) error("tab*/1: illegal file pointer");
    wfp = filep_value(t);
    if (! is_writable(wfp))
      {
        wfp = filep;
        error("tab*/2: file not open");
      }
  }

  tprint0("\t");
  wfp = filep;
  return(SYSTRUE);
}

int var_pred(t,e)
struct term *t;
register struct pair *e;
{
        register struct pair *p;
        register struct term *tt;

        tt = Arg(t,0);
        down(p,tt,e);
        if (p != NULL) return(SYSTRUE);         /* (t,e) is var */
        else return(SYSFAIL);                   /* (t,e) is not var */
}

/* equal ( = ) predicate :
        equal(t1,t2) = SYSTRUE : if t1/e = t2/e
        else SYSFAIL
*/

int equal_pred(t,e)
register struct term *t;
struct pair *e;
{
        return(equalpred(Arg(t,0),e,Arg(t,1),e));
}

int eq_pred(t,e)
register struct term *t;
struct pair *e;
{
        return(eq_pred_sub(Arg(t,0),Arg(t,1),e,e));
}

int eq_pred_sub(x,y,ex,ey)
register struct term *x, *y;
register struct pair *ex, *ey;
{
        register struct pair *p;

        down(p,x,ex);
        down(p,y,ey);
        if ((x == y) && (ex == ey)) return(SYSTRUE);
        if (isvar(x) || (p != NULL)) return(SYSFAIL);

        if (x->type.ident != y->type.ident) return(SYSFAIL);
        if (is_atomic(x)) {
          if (atomic_equal(x,y)) return(SYSTRUE);
          else return(SYSFAIL);
        }
        if (is_pst(x))
          return(eq_pred_sub(((struct pst *)x)->p_var,((struct pst *)y)->p_var,
                             ex,ey));

        if (is_clause(x) || is_list(x)) {
          do {
            if (eq_pred_sub(head_of_list(x),head_of_list(y),ex,ey) == SYSFAIL)
              return(SYSFAIL);
            x = tail_of_list(x);
            y = tail_of_list(y);
          } while ((x != NULL) && (x != NIL) && (y != NULL) && (y != NIL));
          return(SYSTRUE);
        }
        if (is_functor(x) && is_functor(y)) {
            register int i, a = x->t_arity;
            if (a != y->t_arity) return(SYSFAIL);
            if (a < 0) a = -a;
            for(i=0;i < a; i++) {
              if (eq_pred_sub(Arg(x,i),Arg(y,i),ex,ey) == SYSFAIL)
                return(SYSFAIL);
            }
        }
        return(SYSTRUE);
}


int equalpred(t1,e1,t2,e2)
register struct term *t1, *t2;
register struct pair *e1, *e2;
{
        int *hsave;
        struct pair *esave;
        struct ustack *usave;

        esave = ep;
        hsave = hp;
        usave = usp;

        if (tunify(t1,e1,t2,e2,0) == FALSE)
        {
                undo(usave);
                hp = hsave;
                ep = esave;
                return(SYSFAIL);
        }
        return(SYSTRUE);
}
```

```
int assertz_pred(t,e)
struct term *t;
struct pair *e;
{
  general_assert(t,e,'z');
  return(SYSTRUE);
}

int assert_pred(t,e)
struct term *t;
struct pair *e;
{
  general_assert(t,e,'a');
  return(SYSTRUE);
}

void general_assert(t,e,flag)
struct term *t;
struct pair *e;
char flag;                          /* 'a'(first) or 'z'(last) */
{
    struct term *pred, *defs, *con;
    register struct pair *p, *ee;
    struct clause *c_head, *c_con;
    struct ustack *usave;
    int arity;

    pred = Arg(t,0);
    ee = e;
    down(p,pred,ee);
    if ((p != NULL) || is_atomic(pred)) {
        error_detail(t,e,"assert*/1: illegal argument");
        }
    if (issystem(pred->type,t_func)) {
        error_detail(t,e,"assert*/1: system function cannot be asserted");
        }

    v_list = NULL; v_number = 0;
    pv_list = NULL; p_number = 0;
    usave = usp;
    if ((arity = t->t_arity) < 0) arity = -arity;
          /* make first clause (head) */
    con = (arity == 3) ? Arg(t,2) : NULL;
    defs = (arity > 1) ? Arg(t,1) : NULL;
    c_head = Nclause(termset(pred,ee,ETERNAL),
                     list_to_clause(defs,e), ETERNAL);
    c_con = list_to_clause(con,e);
    if (p_number != 0) renum_pvars((struct pstvar *)pv_list,v_number);
    index_set(c_head,c_con,flag);
    undo(usave);
}

struct clause *list_to_clause(t,e)
register struct term *t;
{
  struct clause *croot, *cbefore, *cc;
  register struct pair *p;
  int *ssave = shp;

  if (t != NULL) down(p,t,e);
  if ((t == NULL) || (t == Nil)) return(NULL);
  croot = cnew(clause);
  croot->c_type = CLAUSE_TYPE;
  cbefore = cc = croot;
  while(1)
    {
      if(! is_list(t)) {
        shp = ssave;
        error_detail(t,e,
                "in assert or execute: illegal argument ... should be LIST");
        }
      cc->c_form = termset(head_of_list(t),e,ETERNAL);
      t = tail_of_list(t);
      down(p,t,e);
      if (t == Nil) break;
      cbefore = cc;
      cc = cnew(clause);
      cc->c_type = CLAUSE_TYPE;
      cbefore->c_link = cc;
    }
  cc->c_link = NULL;
  return(croot);
}


int retract_pred(t,e,n,status)
struct term *t;
struct pair *e;
struct node *n;
int status;
{
        register struct set *ss, *foreset;
        register struct pair *p, *et;
        register struct ustack *usave;
        struct term *tt;
        struct term *c_defs, *c_con;
        struct term *defs, *con;
        struct pair *newenv;
        int arity;

        if ((arity = t->t_arity) < 0) arity = -arity;
        tt = Arg(t,0);
        et = e;
        down(p,tt,et);
        if (isvar(tt) || is_atomic(tt))
          error("retract*/1: illegal argument");
        if (!issuser(tt->type,t_func)) return(SYSFAIL);
        if ((status == BACKTRACK) && (n->n_set != DUMMY_DEF))
```

```
                {
                  foreset = n->n_set;
                  ss = foreset->s_link;
                }
        else
          {
            foreset = NULL;
            ss = Pred(tt)->def.f_set;
          }

        usave = usp;

        con = (arity == 3) ? Arg3(t) : NIL;
        defs = (arity >= 2) ? Arg2(t) : NIL;
        while(ss != NULL)
          {
            newenv = Nenv((int)ss->s_annumber);
            if (tunify(tt,et,ss->s_clause->c_form,newenv,0)==FALSE)
            {
              undo(usave); foreset = ss; ss = ss->s_link;
              continue;
            }
            e_defs = tolist(ss->s_clause->c_link,TEMPORAL);
            if (tunify(defs,e,e_defs,newenv,0) == FALSE)
            {
              undo(usave); foreset = ss; ss = ss->s_link;
              continue;
            }
            e_con = tolist(ss->s_constraint,TEMPORAL);
            if (tunify(con,e,e_con,newenv,0) == FALSE)
            {
              undo(usave); foreset = ss; ss = ss->s_link;
              continue;
            }
            if (foreset == NULL)        /* set the next goal */
              {
                Pred(tt)->def.f_set = ss->s_link;
                n->n_set = DUMMY_DEF;
              }
            else
              {
                foreset->s_link = ss->s_link;
                n->n_set = foreset;
              }
            ((struct func *)tt->type.t_func)->f_setcount--;
            if is_unitclause(ss)
              ((struct func *)tt->type.t_func)->f_unitcount--;
            ((struct func *)tt->type.t_func)->f_mark = VACUITY_NOTHERE;
            Def_Modified = 1;
            return(SYSTRUE);
          }
        return(SYSFAIL);
}

void clear_predicate(f)         /* clear user predicate */
register struct func *f;
{
        register int i;

        f->def.f_set = NULL;
        f->f_setcount = 0;
        f->f_unitcount = 0;
        for (i = 0; i < f->f_arity; i++) Component(f,i) = NULL;
/*      f->f_roles[0] = 0; */
}

int abolish_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *f, *a;
  register struct pair *ef, *ea, *p;
  struct func *fun;

  f = Arg(t,0);
  a = Arg(t,1);
  ef = ea = e;
  down(p,f,ef); down(p,a,ea);

  if ((f->type.ident < CONST_LIST_TYPE) || (! is_int(a))) {
        error_detail(t,e,"abolish/2: Illegal argument.");
        }

  fun = funcsearch(Predname(f),(int)(num_value(a)));
  if (fun != NULL)
    {
      if (issystem(fun)) {
        error_detail(t,e,"abolish/2: System predicates cannot be abolished");
        }
      clear_predicate(fun);
      Def_Modified = 1;         /* def modified ! */
    }
  return(SYSTRUE);
}

int makelist_pred(t,e)   /* for predicate ' ml(Pred,List)  (-,-) ' */
struct term *t;
struct pair *e;
{
        struct term *t0, *t1, *lt, *tfun;
        register struct pair *e0, *el, *efun, *p;
        int nvars, depth = 0;

        t0 = Arg(t,0);
        t1 = Arg(t,1);
        e0 = el = e;
        down(p,t0,e0);
        down(p,t1,el);

        /* 1st arg is var */
```

```
        if( isvar(t0) ){
           if (isvar(t1)) return(SYSPA1);
           if (! is_list(t1)) {
               error_detail(t,e,"ml/2: Illegal argument");
               }

       tfun = head_of_list(t1);  /* tfun : functor name */
           efun = el;
           down(p,tfun,efun);
           if (isvar(tfun) || (! is_functor(tfun))) {
               error_detail(t,e,"ml/2:Illegal term for functor.");
               }

           tl = tail_of_list(t1);
           depth=Llevel(tl,el,&nvars);
           if (Pred(tfun) == LIST) {
             if (depth != 2) {
               error_detail(t,e,"ml/2: Illegal argument for LIST");
               }
             tt = (struct term *)
               Nlist(head_of_list(tl),
                     (struct clause *)tail_of_list(tl),TEMPORAL);
             return(equalpred(t0,e0,tt,efun));
           }
           tt = Nterm(depth,TEMPORAL);
           Pred(tt) = Predicate(Predname(tfun), depth);
           if (tl != NIL )
             [
               efun = Nenv(0);
               LtoP(tl,el,tt,depth);
             ]
           return(equalpred(t0,e0,tt,efun));
         }

         /* 1st arg is term */
         if (is_atomic(t0)) tfun=t0;
         else if (is_list(t0)) {
           Pred(tfun)=LIST;
           tt = (struct term *)Nlist(tfun,(struct clause *)t0,TEMPORAL);
           return(equalpred(t1,el,tt,e0));
         }
         else if (is_functor(t0))
           {
             tfun = Nterm(0,TEMPORAL);
             tfun->type.t_func = Predicate(Predname(t0),0);
           }
         else error("ml/2:Illegal argument");
         tt = (struct term *)Nlist(tfun,Ptol(t0),TEMPORAL);
         return(equalpred(t1,el,tt,e0));
}

int Llevel(t,e,nv)   /* from makelist() : Listlevel -> Depth (int) */
register struct term *t;
register struct pair *e;
int *nv;
{
  register struct pair *pp;
  int depth=0;

  *nv = 0;

  if (isvar(t)) down(pp,t,e);
  while( t != NIL )
    {
      if (! is_list(t)) error("ml/2: cdr is real var");
      if (! isconst(head_of_list(t))) (*nv)++;
      t = tail_of_list(t);
      depth++;
      if (isvar(t)) down(pp,t,e);
    };
  return(depth);
}


void LtoP(t,e,tt,depth) /* from makelist() : List -> Predicate */
register struct term *t, *tt;
register struct pair *e;
int depth;
{
        register struct pair *p;
        register int i;

        v_list = NULL; v_number = 0;
        for(i = 0; i < depth ; i++)
          [
            if (isvar(t)) down(p,t,e);
            if (isconst(head_of_list(t))) Arg(tt,i)=head_of_list(t);
            else {
              Nvar(Anonymous VarName,TEMPORAL);
              p = Nenv(1);
              p->p_body = head_of_list(t);
              p->p_env = e;
              Arg(tt,i)=(struct term *)v_list;
            }
            t = tail_of_list(t);
          }
        return;
}


struct clause *Ptol(t)    /* from makelist() : Predicate -> List */
struct term *t;
{
  struct clause *root;
  register struct term *tt, *temp;
        int pos = 0, arity;

        if (is_atomic(t)) return((struct clause *)NIL);
        if ((arity = t->t_arity)==0) return((struct clause *)NIL);
        if (arity < 0) arity = -arity;
```

```
        root = Nlist(NIL,(struct clause *)NIL,TEMPORAL);
        tt = (struct term *)root;
        while(1) {
          head_of_list(tt) = Arg(t,pos);
          pos++;
          if (pos >= arity) break;
          tail_of_list(tt) = temp =
            (struct term *)Nlist(NIL,(struct clause *)NIL,TEMPORAL);
          tt = temp;
        }
        return(root);
      }

int name_pred(t,e)      /* for predicate ' name(String,List) ' */
struct term *t;
struct pair *e;
{
        register struct term *tt,*arg0,*arg1;
        register struct pair *p,*e0,*el;

        arg0 = Arg(t,0);
        arg1 = Arg(t,1);
        e0 = el = e;
        *nbuf = '\0';
        down(p,arg0,e0);
        down(p,arg1,el);

        /* 1st arg is var */
        if (isvar(arg0)){
              if (isvar(arg1)) return(SYSFAIL);
              LtoC(arg1,el,0,FROM_NAME);    /* List -> (char)nbuf[] */
              if (alldigit(nbuf)) tt = Ntem(nbuf,TEMPORAL);
              else {
                tt = Ntem(0,TEMPORAL);
                Pred(tt) = Predicate(nbuf,0);
              }
     return(equalpred(arg0,e0,tt,(struct pair *)NULL));
        }

    /* 1st arg is constant */
        if (is_num(arg0))
          {
            sprintf(nbuf,"%d",(int)num_value(arg0));
            tt = CtoL(nbuf, FROM_NAME);
          }
        else if (is_string(arg0))
          tt = CtoL( str_value(arg0), FROM_NAME );
        else tt = CtoL(Predname(arg0), FROM_NAME);
        return(equalpred(arg1,el,tt,(struct pair *)NULL));
}

void ltoC(t,e,pos, flag)        /* from name_pred() : List -> Character */
struct term *t;
struct pair *e;
int pos, flag;   /* flag = 0(FROM_CONC) /char, 1(FROM_NAME) /int */
{
    register struct pair *e0, *el, *p;
    register struct term *arg0, *arg1;

    if (is_string(t))
      { strcpy(nbuf, str_value(t)); return; }
    if (! is_list(t)) error("name/2: 2nd arg is illegal term.");
    arg0 = head_of_list(t);
    arg1 = tail_of_list(t);
    e0 = el = e;
    down(p,arg0,e0);
    down(p,arg1,el);

    if (isvar(arg0) || (! isatom(arg0)) || isvar(arg1)) {
      sprintf(nbuf,"%s/2: 2nd arg is real VAR",
              ((flag) ? "name" : "concat2"));
      error_detail(t,e,nbuf);
    }
    if (flag) {
      if (! is_int(arg0))
        error("name/2: 2nd arg contains illegal term.");
      else nbuf[pos++] = (int)num_value(arg0);
    }
    else
      {
        if (is_string(arg0))
          strcat(nbuf, str_value(arg0));
        else if ((is_functor(arg0)) && (isatom(arg0)))
          strcat(nbuf,Predname(arg0));
        else if (is_int(arg0)) {
          int len = strlen(nbuf);
          nbuf[len++]=(int)num_value(arg0);
          nbuf[len]='\0';
        }
        else {
          error_detail(arg0,e0,"concat2/2: illegal arg");
        }
      }
    if (arg1 != NIL) ltoC(arg1,el,pos,flag);
    else if (flag) nbuf[pos] = '\0';
    return;
}

struct term *CtoL(nbuf, flag)
        /* from name_pred() : Character -> List */
char *nbuf;
int flag;  /* 0(FROM_CONC) -> char, 1(FROM_NAME)  -> int */
{
  struct term *root, *t;
  char s[2];
  register int pos = 0;

  root = t = (struct term *)Nlist(NIL,(struct clause *)NIL,TEMPORAL);
```

```
  while (1)
    {
      if (flag) {
        head_of_list(t) =Num_val((float)nbuf[pos++],TEMPORAL);
      }
      else {
        *s = nbuf[pos++];
        *(s+1) = '\0';
        head_of_list(t) = Nstr(s, TEMPORAL);
      }
      if (nbuf[pos] == '\0') return(root);
      t = (tail_of_list(t) =
          (struct Term *)Nlist(NIL,(struct clause *)NIL,TEMPORAL));
    }
}

int arg_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *pos, *tt, *var;
  register struct pair *p, *ep, *et, *ev;
  int i, arity;

  pos = Arg(t,0);
  tt = Arg(t,1);
  var = Arg(t,2);

  ep = et = ev = e;
  down(p,pos,ep); down(p,tt,et); down(p,var,ev);

  if (isvar(pos) || isvar(tt)) return(SYSFAIL);
  if (! is_int(pos)) {
        error_detail(t,e,"arg/3: illegal argument");
        }
  i = num_value(pos);
  if (is_list(tt))
    switch (i) {
      case 1: return(equalpred(head_of_list(tt),et,var,ev));
      case 2: return(equalpred(tail_of_list(tt),et,var,ev));
      default:
        error_detail(t,e,"arg/3:Illegal argument for position");
      }
  else if (! is_functor(tt)) {
        error_detail(t,e,"arg/3:Illegal argument for functor");
        }

  if((arity = tt->t_arity) < 0) arity = -arity;
  if ((i <= 0) || (tt->type.ident == 0) || i > arity) {
        error_detail(t,e,"arg/3: illegal argument");
        }
  return(equalpred(Arg(tt,i-1),et,var,ev));
}
```

```
int functor_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *tt, *fun, *ari;
  register struct pair *p, *et, *ef, *ea;

  tt = Arg(t,0); fun = Arg(t,1); ari = Arg(t,2);
  ea = ef = et = e;

  down(p,tt,et); down(p,fun,ef); down(p,ari, ea);

  if (isvar(tt)) return(make_func(fun,ari,tt,et));

  if ((! is_functor(tt)) && (! is_list(tt)))
    error_detail(t,e,"functor/3: 1st argument is not appropriate");

  return(match_func(tt,et,fun,ef,ari,ea));
}

int make_func(f,a,t,e)
struct term *f, *a, *t;
struct pair *e;
{
  struct term *temp;
  struct pair *env;
  int i,arity;

  if (isvar(f) || isvar(a)) return(SYSFAIL);
  if (! (isatom(f))) {
        error_detail(t,e,"functor/3: 2nd argument is not atom");
        }
  if (! (is_int(a))) {
        error_detail(t,e,"functor/3: 3rd argument is not integer");
        }
  if ((arity = (int)(num_value(a))) < 0) {
        error_detail(t,e,"functor/3: 3rd argument is illegal number");
        }

  if (arity==0)  return(equalpred(t,e,f,e));

  v_list = NULL;  v_number = 0;
  env = Nenv(arity);
  if ((arity == 2) && (Pred(f)==LIST))
        temp = (struct term *)
                Nlist(Nvar(Anonymous_VarName,TEMPORAL),
                      Nvar(Anonymous_VarName,TEMPORAL),TEMPORAL);
  else {
   temp = Nterm(arity,TEMPORAL);
   Pred(temp) = Predicate(Predname(f), arity);
   for (i=0; i < arity; i++)
        Arg(temp,i)=Nvar(Anonymous_VarName,TEMPORAL);
    }
  return(equalpred(t,e,temp,env));
}
```

```
int match_func(t,e,f,ef,a,ea)
struct term *t, *f, *a;
struct pair *e, *ef, *ea;
{
  struct term *temp;
  int arity, *hsave;
  struct pair *esave;
  struct ustack *usave;

        hsave = hp;
        esave = ep;
        usave = usp;

  arity = t->t_arity;
  if (arity < 0) arity = -arity;

  if (is_list(t))
        temp =Num_val(2.0, TEMPORAL);
  else temp = Num_val((float)arity,TEMPORAL);

  if (funify(a,ea,temp,(struct pair *)NULL,0) == FALSE)
  {
         undo(usave);    hp = hsave;     ep = esave;
         return(SYSFAIL);
  }
  temp = Nterm(0,TEMPORAL);
  if (is_list(t))
    Pred(temp)=LIST;
  else Pred(temp) = Predicate(Predname(t), 0);

  if (funify(f,ef,temp,(struct pair *)NULL,0) == FALSE) {
    undo(usave);          hp = hsave;     ep = esave;
    return(SYSFAIL);
  }
  return(SYSTRUE);
}

int clause_pred(t,e,n,status)   /* clause(P,B,C) P:nonvar*/
struct term *t;
struct pair *e;
struct node *n;
int status;
{
        register struct pair *ee, *p, *newenv,
        register struct term *tt;
        struct term *t_body, *t_con;
        struct ustack *usave;
        struct set *s;
        int *hsave;
        struct pair *esave;

        ee = e;
        tt = Arg(t,0);   /* head */
        down(p,tt,ee);
        if (isvar(tt)) return(SYSFAIL);
        if (status != BACKTRACK)
                n->n_set = (t->type.t_func->def.f_set;
        if (n->n_set == NULL)  return(SYSFAIL);

        esave = usp;
        hsave = hp;
        esave = ep;
        for (s = n->n_set; s != NULL; s = s->s_link)
        {
                newenv = Nenv((int)s->s_anumber);
                if (funify(tt,ee,s->s_clause->c_form,newenv,0) == FALSE)
                {
                        undo(usave);hp = hsave; ep = esave;continue;
                }

                t_body = tolist(s->s_clause->c_link,TEMPORAL);
                tt = Arg(t,1);
                ee = e;
                down(p,tt,ee);
                if (funify(tt, ee, t_body, newenv,0)==FALSE)
                {
                        undo(usave);hp = hsave; ep = esave;continue;
                }

                t_con = tolist(s->s_constraint,TEMPORAL);
                tt = Arg(t,2);
                ee = e;
                down(p,tt,ee);
                if (funify(tt, ee, t_con, newenv,0) == FALSE)
                {
                        undo(usave);hp = hsave; ep = esave;continue;
                }

                n->n_set = s->s_link; /* next goal */
                return(SYSTRUE);
        }
        return(SYSFAIL);
}
```

```
/*************************************************************************
*                 cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 -91
*************************************************************************/
/*-----------------------------------------------------------------------
*                 << syspred2.c >>
*                 (system predicates No.2 : string, number)
-----------------------------------------------------------------------*/

#include "include.h"

/* for IsCC(), CtoL() pred  */
#define FROM_NAME 1
#define FROM_CONC 0

int sum_pred(t,e)
struct term *t;
struct pair *e;
{
  return(calc_pred(t,e,'+'));
}

int multiply_pred(t,e)
struct term *t;
struct pair *e;
{
  return(calc_pred(t,e,'*'));
}


int calc_pred(t,e,op)
struct term *t;
struct pair *e;
char op;
{
  register struct term *x, *y, *z;
  register struct pair *e0, *e1, *e2, *p;

  e0 = e1 = e2 = e;
  x = Arg(t,0); y = Arg(t,1); z = Arg(t,2);
  down(p,x,e0); down(p,y,e1); down(p,z,e2);

  if(isvar(x)) return(calc_2(y,z,x,e0,op));
  if(isvar(y)) return(calc_2(x,z,y,e1,op));
  else return(calc_1(x,y,z,e2,op));
}

int calc_1(x,y,z,e,op)
struct term *x,*y,*z;
struct pair *e;
char op;
{
  struct term *result;
  register float sum;

/*  if (isvar(x) || isvar(y)) return(SYSFAIL); */

  if (! (is_num(x))) {
    sprintf(nbuf,"%s/3: Illegal argument as 1st argument",
            ((op == '+') ? "sum" : "multiply") );
    error_detail(x,(struct pair *)NULL,nbuf);
  }
  if (! (is_num(y))) {
    sprintf(nbuf,"%s/3: Illegal argument as 2nd argument",
            ((op == '+') ? "sum" : "multiply") );
    error_detail(y,(struct pair *)NULL,nbuf);
  }

  if (op=='+') sum = num_value(x) + num_value(y);
  else if (op=='*') sum = num_value(x) * num_value(y);
  else error("system error at calc pred");

  result = Nnum_val(sum,TEMPORAL);
  return(equalpred(z,e,result,(struct pair *)NULL));
}

int calc_2(x,z,y,e,op)
struct term *x,*y,*z;
struct pair *e;
char op;
{
  struct term *result;
  register float temp;

  if (isvar(x) || isvar(z)) return(SYSFAIL);

  if (! (is_num(x))) {
    sprintf(nbuf,"%s/3: Illegal argument as 1st argument",
            ((op == '+') ? "sum" : "multiply") );
    error_detail(x,(struct pair *)NULL,nbuf);
  }
  if (! (is_num(z))) {
    sprintf(nbuf,"%s/3: Illegal argument as 2nd argument",
            ((op == '+') ? "sum" : "multiply") );
    error_detail(z,(struct pair *)NULL,nbuf);
  }

  temp = num_value(x);
  if ((op=='*') && (temp==0.0)) error("multiply/3: zero division");

  if (op=='+')  temp = -num_value(z) + temp;
  else if (op=='*')  temp = num_value(z)/temp;

  result = Nnum_val(temp,TEMPORAL);
  return(equalpred(y,e,result,(struct pair *)NULL));
}

int greater_pred(t,e)
struct term *t;
```

```
struct pair *e;
{
  return(numcomp_pred(t,e,0));
}

int less_pred(t,e)
struct term *t;
struct pair *e;
{
  return(numcomp_pred(t,e,1));
}

int geq_pred(t,e)
struct term *t;
struct pair *e;
{
  return(numcomp_pred(t,e,2));
}

int leq_pred(t,e)
struct term *t;
struct pair *e;
{
  return(numcomp_pred(t,e,3));
}

static char* compare_predicates[] = {
  "greater", "less:", "geq", "leq" };

int numcomp_pred(t,e,op)
struct term *t;
struct pair *e;
int op;
{
  register struct term *x, *y;
  register struct pair *e0, *e1, *p;
  int q, l;

  e0 = e1 = e;
  x = Arg(t,0); y = Arg(t,1);
  down(p,x,e0); down(p,y,e1);

  if(isvar(x) || (p != NULL))
    return(SYSFAIL);

  if (! (is_num(x))) {
    sprintf(nbuf,"%s/2: Illegal argument as 1st Arg",
            compare_predicates[op]);
    error_detail(x,e0,nbuf);
  }
  if (! (is_num(y))) {
    sprintf(nbuf,"%s/2: Illegal argument as 2nd Arg",
            compare_predicates[op]);
    error_detail(y,e1,nbuf);
  }
  q = (num_value(x) > num_value(y)) ? SYSTRUE : SYSFAIL;
  l = (num_value(x) < num_value(y)) ? SYSTRUE : SYSFAIL;

  switch (op) {
    case 0: return(q);
    case 1: return(l);
    case 2: return((l==SYSFAIL) ? SYSTRUE : SYSFAIL);
    case 3: return((q==SYSFAIL) ? SYSTRUE : SYSFAIL);
    }
}

/* concat("ab","cde",X)  > X = "abcde" */
int concat_pred(t,e,n,status)
struct term *t;
struct pair *e;
struct node *n;
int status;
{
  register struct term *x, *y, *z;
  register struct pair *px, *py, *p;
  struct pair *ex, *ey, *ez;
  int len;
  char *buf;

  x = Arg(t,0);
  y = Arg(t,1);
  z = Arg(t,2);
  ex = ey = ez = e;
  down(px,x,ex); down(py,y,ey); down(p,z,ez);

  if (isvar(x) && isvar(y)) {
    if (status ==BACKTRACK) { /* X,Y are Vars, and Z is CONST */
      if ((len = (int)n->n_set -1) < 0)  return(SYSFAIL);
/* copy status chars from z to nbuf */
      strncpy(nbuf,str_value(z),len);
      nbuf[len] = '\0'; /* due to BUG of SUN4 */
      upush(&(px->p_body));
      upush(&(px->p_env));
      px->p_body = Nstr(nbuf,TEMPORAL);
      px->p_env = (struct pair *)NULL;
      buf = str_value(z);
      upush(&(py->p_body));
      upush(&(py->p_env));
      buf += len;
      py->p_body = Nstr(buf,TEMPORAL);
      py->p_env = (struct pair *)NULL;
      n->n_set = (struct set *)len;
      return(SYSTRUE);
    }
    else {
      if (isvar(z)) return(SYSFAIL);
      if (! is_string(z)) {
        error_detail(z,ez,"concat/2: Illegal 3rd argument");
      }
```

```
      len = strlen(str_value(z));
      upush(&(px->p_body));
      upush(&(px->p_env));
      px->p_body = z;
      px->p_env = ez;
      nbuf[0] = '\0';
      upush(&(py->p_body));
      upush(&(py->p_env));
      py->p_body = Nstr(nbuf,TEMPORAL);
      py->p_env = (struct pair *)NULL;
      n->n_set = (struct set *)len; /* memorize the position */
      return(SYSTRUE);
    }
  }

  if(isvar(x)) return(diff_str(y,z,x,ex,0));
  if(isvar(y)) return(diff_str(x,z,y,ey,1));
  else return(app_str(x,y,z,ez));
}

int app_str(x,y,z,ez)
struct term *x, *y, *z;
struct pair *ez;
{
  struct term *result;

  if (! (is_string(x) && is_string(y))) error("concat/3: illegal term");
  if ((strlen(str_value(x))+strlen(str_value(y))) > NAME_MAX)
    error("concat/3: too long string");
  strcpy(nbuf,str_value(x));
  strcat(nbuf,str_value(y));
  result = Nstr(nbuf,TEMPORAL);
  return(equalpred(z,ez,result,(struct pair *)NULL));
}

int diff_str(x,z,y,e,first)
struct term *x, *y, *z;
struct pair *e;
int first;  /* assuming 0/last_half, 1/first half is designated */
{
  struct term *result;
  int lx, lz, dif;
  char *ex, *ez;

  if (isvar(z)) return(SYSFAIL);

  if (! (is_string(z)  && (isvar(x) || is_string(x))))
     error("concat/3: illegal term");

  ex = str_value(x); ez = str_value(z);
  if ((lz = strlen(ez)) < (lx = strlen(ex)))
    error("concat/3: not appropriate args");

  if (first) /* find last half */
    {
      register int pos;

      for (pos = 0; pos < lx; pos++)
        if (ex[pos] != ez[pos]) return(SYSFAIL);
      ez += pos;
      result = Nstr(ez,TEMPORAL);
    }
  else /* find first half */
    {
      register int pos;

      dif = lz - lx;
      for (pos = dif; pos < lz; pos++)
        if (ex[pos-dif] != ez[pos]) return(SYSFAIL);
      /*  strcpy(nbuf, ez, dif); this must be bug.  */
      strncpy(nbuf, ez, dif);
      nbuf[dif] = '\0';
      result = Nstr(nbuf,TEMPORAL);
    }
  return(equalpred(y,e,result,(struct pair *)NULL));
}

/* concat2("abcde",X) -> X = ["a","b","c","d","e"] */
int concat2_pred(t,e)
struct term *t;
struct pair *e;
{
  struct term *x, *y;
  struct pair *ex, *ey, *p;
  struct term *tt;

  x = Arg(t,0);
  y = Arg(t,1);
  ex = ey = e;
  down(p,x,ex), down(p,y,ey);
  *nbuf = '\0';

  if (isvar(x)) {
    if (isvar(y)) return(SYSFAIL);
    ltoC(y,ey,0,FROM_CONC);
    tt = Nstr(nbuf, TEMPORAL);
    return(equalpred(x,ex,tt,(struct pair *)NULL));
  }
  if (is_num(x)) {
      sprintf(nbuf, "%d",(int)num_value(x));
      tt = CtoL(nbuf, FROM_CONC);
    }
  else if (is_string(x)) tt = CtoL(str_value(x), FROM_CONC);
  else tt   CtoL(x->type.t_func->f_name, FROM_CONC);
  return(equalpred(y,ey,tt,(struct pair *)NULL));
}


int strlen_pred(t,e)
struct term *t;
```

```
struct pair *e;
{
  struct term *s, *l;
  struct pair *es, *el, *p;
  int len;

  s = Arg(t,0);
  l = Arg(t,1);

  es = el = e;
  down(p,l,el);
  down(p,s,es);

  if (p != NULL) return(SYSFAIL);
  if (! is_string(s)) {
    error_detail(t,e,"strlen/1: 1st arg is not string");
  }
  if  (! (isvar(l) || is_num(l))) {
    error_detail(t,e,"strlen/2: 2nd arg is neither Var nor Number");
  }
  len = strlen(str_value(s));
  t = Num_val((float)len,TEMPORAL);
  return(equalpred(l,el,t,(struct pair *)NULL));
}

/* substring("abcde",2,X) -> X = "cde"
   substring("abcde",-3,2,X) -> X = "cd" */
int substr_pred(t,e)
struct term *t;
struct pair *e;
{
  static char *emesg = "substring/%d: %s arg is not %s";
   struct term *s, *tmp;
   register struct pair *p, *ee;
   int arity,start,numb,len;
   char *sr;

   arity = t->t_arity;
   if (arity < 0) arity = -arity;

   s = Arg(t,0);
   ee = e;
   down(p,s,ee);

   if (! is_string(s)) {
     sprintf(nbuf,emesg,arity, "1st", "string");
     error_detail(t,e,nbuf);
   }
   tmp = Arg(t,1);
   ee = e;
   down(p,tmp,ee);
   if (! is_int(tmp))  {
     sprintf(nbuf,emesg, arity, "2nd","integer");
     error_detail(t,e,nbuf);
   }
   start = num_value(tmp);
   len = strlen(str_value(s));
   if (start < 0) start += len;

   if (arity == 4) {
     tmp = Arg(t,2);
     ee = e;
     down(p,tmp,ee);
     if (! is_int(tmp)) {
       sprintf(nbuf,emesg,4,"3rd","integer");
       error_detail(t,e,nbuf);
     }
     numb = num_value(tmp);
     if (numb < 0) numb+=len;
   }
   else  { /* arity == 3 */
     numb = len-start;
   }
   if ( (start > len) || (numb > len) || (start < 0)) {
     sprintf(nbuf,"substring/%d: illegal argument value",arity);
     error_detail(t,e,nbuf);
   }

   sr = str_value(s);
   sr += start;
   strncpy(nbuf,sr,numb);
   nbuf[numb] = '\0';
   tmp = Nstr(nbuf,TEMPORAL);
   return(equalpred(Arg(t,arity-1),e,tmp,(struct pair *)NULL));
 }

/* divstr("abcd",2,X,Y) -> X = "ab", Y = "cd" */
/* divstr(4,4,?,?) or divstr(4, ,4,?) */
int divstr_pred(t,e)
struct term *t;
struct pair *e;
{
   static char *emesg = "divstr/4: %s is not %s";
   register struct pair *p, *ee, *el;
   struct term *str, *temp, *first;
   int n,len, firsthalf();
   char *sr, *sf;

   str = Arg(t,0);
   ee = e;      down(p,str,ee);
   if (! is_string(str)) {
     sprintf(nbuf,emesg,"1st","string");
     error_detail(t,e,nbuf);
   }
   sr = str_value(str);
   len = strlen(sr);

   temp = Arg(t,1);
   ee = e;      down(p,temp,ee);
   if (p != NULL) { /* 2nd arg is var */
```

```
    el = e;
    first = Arg(t,2);
    down(p,first,el);
    if (! is_string(first)) {
      sprintf(mbuf,emesg,"2nd","integer and 3rd arg is var");
      error_detail(t,e,mbuf);
    }
    sf = str_value(first);
    n = strlen(sf);
    if ((n <= len) && (firsthalf(sf,sr)==TRUE) &&
        (equalpred(temp,ee,Num_val((float)n,TEMPORAL),(struct pair *)NULL)
         == SYSTRUE)){
      sr += n;
      return(equalpred(Arg(t,3),e,Str(sr,TEMPORAL),(struct pair *)NULL));
    }
    return(SYSFAIL);
  }
  else if (! is_int(temp)) {
    sprintf(mbuf,emesg,"2nd","integer");
    error_detail(t,e,mbuf);
  }

  n = num_value(temp);
  if (n < 0) n += len;
  if ((n > len) || (n < 0)) {
    sprintf(mbuf,emesg,"2nd","appropriate");
    error_detail(t,e,mbuf);
  }

  strncpy(mbuf,sr,n);
  mbuf[n] = '\0';
  temp = Str(mbuf,TEMPORAL);
  if (equalpred(Arg(t,2),e,temp,(struct pair *)NULL) == SYSFAIL)
    return(SYSFAIL);

  sr += n;
  temp = Str(sr,TEMPORAL);
  return(equalpred(Arg(t,3),e,temp,(struct pair *)NULL));
}


int firsthalf(h,w)
char h[], w[];
{
  register int i;
  for (i = 0; h[i] == w[i]; i++);
  if (h[i] == '\0') return(TRUE);
  else return(FALSE);
}


/* stremp("ab","abc", X)  > X = '<' */
/* stremp(+,+,-) */
int stremp_pred(t,e)
struct term *t;
struct pair *e;
{
  static char *emesg = "stremp/3: %s is not string";
  register struct pair *p, *ee;
  struct term *a, *b;
  int result;

  a = Arg(t,0);        b = Arg(t,1);
  ee = e; deref(p,a,ee);
  if (! is_string(a)) {
    sprintf(mbuf,emesg,"1st");
    error_detail(t,e,mbuf);
  }
  ee = e; deref(p,b,ee);
  if (! is_string(b)) {
    sprintf(mbuf,emesg,"2nd");
    error_detail(t,e,mbuf);
  }
  result = strcmp(str_value(a),str_value(b));
  if (result < 0)
    return(equalpred(Arg(t,2),e,S_LESS,(struct pair *)NULL));
  else if (result == 0)
    return(equalpred(Arg(t,2),e,S_EQ,(struct pair *)NULL));
  else /* result > 0 */
    return(equalpred(Arg(t,2),e,S_GREATER,(struct pair *)NULL));
}

int compare_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct pair *p, *ee;
  struct term *a, *b;
  float j;
  int i;

  ee = e;
  a = Arg(t,0);
  deref(p,a,ee);

  ee = e;
  b = Arg(t,1);
  deref(p,b,ee);

  if (is_num(a) && is_num(b)) {
    j = num_value(a) - num_value(b);
    if (j > 0.0)
      return(equalpred(Arg(t,2),e,S_GREATER,(struct pair *)NULL));
    else if (j == 0.0)
      return(equalpred(Arg(t,2),e,S_EQ,(struct pair *)NULL));
    return(equalpred(Arg(t,2),e,S_LESS,(struct pair *)NULL));
  }

  if (is_string(a) && is_string(b)) {
    i = strcmp(str_value(a),str_value(b));
```

```
    if (i > 0)
      return(equalpred(Arg(t,2),e,S_GREATER,(struct pair *)NULL));
    else if (i == 0)
      return(equalpred(Arg(t,2),e,S_EQ,(struct pair *)NULL));
    return(equalpred(Arg(t,2),e,S_LESS,(struct pair *)NULL));
  }

  error_detail(t,e,"compare*/3: Args are mismatched");
}


/* count() predicate :
        count(X)  -> X = 0,1,2,...
        count(3) -> set COUNTNUMBER in 3

*/
long COUNTNUMBER = 0;   /* used for count(gensym) predicate */

int count_pred(t,e)
struct term *t;
struct pair *e;
{
   register struct pair *p;
   struct term *result;

   t = Arg(t,0);
   down(p,t,e);

   if (p != NULL) {
        result = Num_val((float)COUNTNUMBER,TEMPORAL);
        COUNTNUMBER++;
        return(equalpred(t,e,result,(struct pair *)NULL));
             }
   if (is_int(t)) {
       COUNTNUMBER=(long)num_value(t);
       return(SYSTRUE);
     }
   error_detail(t,e,"count/1: illegal argument.");
}

int gensym_pred(t,e)
struct term *t;
struct pair *e;
{
  register struct term *tt;
  register struct pair *p, *ee;
  struct term *result;
  char *newname[8];

  if (t->t_arity == 2) {
    tt = Arg(t,0);
    ee = e;
    down(p,tt,ee);
    if (is_functor(tt))
      strncpy(newname, tt->type.t_func->f_name,8);
    else if (is_string(tt))
      strncpy(newname, str_value(tt), 8);
    else error_detail(t,e,"gensym/2: 1st Argument should be atom");

    tt = Arg(t,1);
    ee = e;
  }
  else { /* gensym/1 */
    tt = Arg(t,0);
    ee = e;
    strcpy(newname,genname);
  }

  down(p,tt,ee);
  if (p != NULL) {
  /* new function name is generated in nbuf[] */
      while (1) {
        sprintf(nbuf,"%s%d", newname, GENSYM1++);
        if (exist_fname(nbuf) == NULL) break;
      }
      result = Kterm(0,TEMPORAL);
      result->type.t_func = Predicate(nbuf,0);
      return(equalpred(tt,ee,result,(struct pair *)NULL));
          }
    else error_detail(t,e,"gensym/1:Argument should be Variable");
}
```

```
/*************************************************************
*                 CHI Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989-91
*************************************************************/
/*************************************************************
*                 << jpsgsub.c >>
*                 (system predicates for JPSG parser)
*************************************************************/

#include "include.h"

/*   tree() system predicate  */
void Ptree(),treeprint(),oldlink(),PCAT(),Psubcat();
int null_or_nil();

#define TREEMAX 20              /*      tree depth max  */
int treehist[TREEMAX];
int Category_size = 0;
#define CATMAX 30
int cattype[CATMAX];
int catname[CATMAX][8];

/* category type */
#define Normal 1
#define CatSingle   2
#define CatSet  3

void show_category()            /* for debug */
{
        register int i;

        tpute('[');
        for (i = 0; ;)
        {
                tprint2("%s:%d",catname[i],cattype[i]);
                i++;
                if (i >= Category_size) break;
                tpute(',');
        }
        tpute(']');
}

void init_category()
{
        Category_size = CAT_P->f_arity = 6;
        strcpy(catname[0],"POS");cattype[0] = Normal;
        strcpy(catname[1],"FORM");cattype[1] = Normal;
        strcpy(catname[2],"AUX");cattype[2] = CatSingle;
        strcpy(catname[3],"AUX");cattype[3] = CatSingle;
        strcpy(catname[4],"SC");cattype[4] = CatSet;
        strcpy(catname[5],"SEM");cattype[5] = Normal;
}

void list_to_cat(t0,n)          /* set cattype[],catname[] */
struct term *t0;
int n;
{
        register int i;
        register struct term *t;

        for (t = t0, i = 0; i < n; i++, t = Arg(t,1))
        {
                if (! is_list(t)) error("illegal format (feature type)");
                if (!isvar(head_of_list(t))) error("illegal feature name");
                strcpy(catname[i],vname(head_of_list(t)));
                t = tail_of_list(t);
                if (! is_list(t)) error("illegal format (feature type)");
                if (!is_num(head_of_list(t))) error("illegal feature type");
                cattype[i] = num_value(head_of_list(t));
        }
}

void set_category()             /* %C command */
{
        struct term *t;
        v_number = 0;
        v_list = NULL;
        advance;
        t = Rterm(1200,TERMFORM);
        skipline;
        if ((v_number < 3)  || (v_number > 30)){
                tprint0("illegal feature number (>3,<30)");
                init_category();
                return;
        }
        list_to_cat(t,v_number);
        Category_size = v_number;
        CAT_P->f_arity = v_number;
}

int tree_pred(t,e)
struct term *t;
struct pair *e;
{
        Ptree(Arg(t,0),e);
        return(SYSTRUE);
}

void Ptree(t,e)         /* print tree entry */
struct term *t;
struct pair *e;
{
        int i;
        struct pair *p;

        down(p,t,e);
        if (t->type.t_func != T_P) {
                Pterm(t,e);
                return;
```

```
        }
        for (i = 0; i < TREEMAX; i++)
                treehist[i] = 0;        /* array initialize */
        treeprint(t,e,0);NL;
}

void oldlink(n)         /* print old link in depth n */
int n;
{
        int i;

        tprint0(" ");
        for (i = 0; i < n; i++) {
                if (treehist[i] != 0) {
                        tprint0("|   ");
                }
                else {
                        tprint0("    ");
                }
        }
}

void treeprint(t,e,n)           /* print tree main */
struct term *t;
struct pair *e;
int n;                  /* depth */
{
        struct pair *p;

        down(p,t,e);
        if ((t->type.t_func != T_P) || (n > TREEMAX) ){
                PCat(t,e,0);
                return;
        }
        treeprint(Arg(t,0),e,n + 1);
        if (Arg(t,2) == NIL) {
                tprint0("---");
                Pterm(Arg(t,1),e);
                return;
        }
        NL;
        treehist[n] = 1;
        oldlink(n);tprint0("|  ");NL;
        oldlink(n);tprint0("|--");treeprint(Arg(t,1),e,n + 1);NL;
        oldlink(n);tprint0("|  ");NL;
        treehist[n] = 0;
        oldlink(n);tprint0("|_ ");treeprint(Arg(t,2),e,n + 1);
}

int null_or_nil(t,e)
register struct term *t;
register struct pair *e;
{
        register struct pair *p;
        down(p,t,e);
        if (t == NULL) return(TRUE);
        else if (t == NIL) return(TRUE);
        else return(FALSE);
}


void PCat(t,e,f)                /* print category */
struct term *t;
struct pair *e;
int f;          /*  if f = 1  ,does not print SEM */
{
        struct pair *p;
        register int i;

        down(p,t,e);
        if (t->type.t_func != CAT_P) {
                Pterm(t,e);
                return;
        }
        Pterm(Arg(t,0),e);              /* print pos */
        tprint0("|");
        Pterm(Arg(t,1),e);              /* print form */
        if (f == 1)
        {
                tputc('|');
                return;
        }
        for (i = 2; i < (Category_size - 1); i++)
        {
                if (null_or_nil(Arg(t,i),e)) continue;
                tprint1(", %s:",catname[i]);
                if (cattype[i] == CatSingle)
                        Psubcat(Arg(t,i),e);
                else if (cattype[i] == CatSet)
                {
                        tputc('{');
                        Psubcat(Arg(t,i),e);
                        tputc('}');
                }
                else    Pterm(Arg(t,i),e); /* type = Normal */

        }
        tprint0("|");
        Pterm(Arg(t,Category_size - 1),e);              /* print sem */
}

void Psubcat(t,e)               /* print subcat etc. */
struct term *t;
struct pair *e;
{
        struct pair *p;

        down(p,t,e);
        if (t == NIL) return;
        if (t == NULL) return;
```

```
        if (p != NULL){                 /* if t is var */
                /*  Pterm(t,e); */
                fprintf("%s",vname(t));
                return;
        }
        if (! is_list(t)) {
                fputc('?'); /*  Pterm(t,e);  */
                return;
        }
        PCat(head_of_list(t),e,1);
        if (tail_of_list(t) == NIL) return;
        fprint0(", ");
        Psubcat(tail_of_list(t),e);
}



#define CnstMax 20
char *cnametmp[CnstMax];        /* work array */

char *termname(t,e)             /* return functor name of t */
struct term *t;
struct pair *e;
{
        struct pair *p;

        down(p,t,e);
        if (p != NULL) return(vname(t));        /* if t is var */
        return(t->type.t_func->f_name);
}

int pickname(t,e)       /* pick up constraint name in cnametmp[] */
struct term *t;
struct pair *e;
{
        struct pair *p;
        int i;

        for (i = 0;;i++){
                down(p,t,e);
                if (t == NIL) return(i);
                if (t == NULL) return(i);
                if (! is_list(t)){
                        fprint0("constraint error");
                        return(0);
                }
                cnametmp[i] = termname(head_of_list(t),e);
                t = tail_of_list(t);
        }
}

struct term *cnlistmake(n)
int n;
{
        int i;
        struct term *listtop;
        struct term *t,*ct;

        for(i = 0,listtop = NIL; i < n; i++){
          ct = Mterm(0,BIGRNAL);
          ct->type.t_func = Nfunc(USERFUN,cnametmp[i],0);
          t = (struct term *)Nlist(ct,(struct clause *)listtop,TEMPORAL);
          listtop = t;
        }
        return(listtop);
}


int cname_pred(t,e,nn)  /* construct constraint list cnst->cn */
struct term *t;
struct pair *e;
struct node *nn;
{
  struct term *cnst,*cn;
  struct pair *el,*q;
  int n;

  cnst = Arg(t,0);
  cn = Arg(t,1);
  el = e;
  down(q,cn,el);
  if (q == NULL) return(SYSFAIL);       /* if cn isn't var */
  n = pickname(cnst, e);
  if (n > CnstMax) n = CnstMax;
  q->p_body = cnlistmake(n);
  return(SYSTRUE);
}
```

```
/*************************************************************
*               cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                        1989-91
*************************************************************/
/*-----------------------------------------------------------
*               << modular.c >>
*               constraint transformation entry, tools
-----------------------------------------------------------*/

#include "include.h"

#if SUN4 == 1
#include <sys/time.h>
#else
#if CPUTIME != 0
#include <sys/types.h>
#include <sys/times.h>
#endif
#endif

#define in_cheap(X) (( &cheap[0] <= ((int *)X)) && (((int *)X) < clp))
#define in_upper_heap(X,Y) ( in_cheap(X) || ( (Y==MEDIUM) && in_cheap(X)) )

long CONSTRAINT_OLD_TIME;

/*************************************************************
  modular(c)
   @ mode entry
*************************************************************/
void modular(c,vlist,anum)      /* constraint trans. from top level (@) */
struct clause *c;
struct term *vlist;
int anum;
{
        struct clause *sol;

        sol = startmodular(c, vlist, anum); /* tranformation */
        tprint0("solution = ");
        if (sol == MFAIL){      /* fail transformation */
                tprint0("fail.\n");
        }
        else if (sol == NULL){  /* nil constraint */
                tprint0("nil (true).\n");
        }
        else                    /* c.t. success */
        {
                Pclause(sol, (struct pair *)NULL); NL;
                show_newdefs(); /* print DEF_list */
        }
}


/*************************************************************
   cu(t,e)
     process unify() built in predicate
*************************************************************/
/* constraint transformation embedded in Prolog  : unify() pred. */
jmp_buf tolist_fail;

int cu(t,e)             /*  0: cu fail  1: success   */
struct term *t;
struct pair *e;
{
        register struct pair *p, *q;
        struct pair *es;
        struct term *tt;
        struct clause *c, *clist;
#if CPUTIME != 0
        struct tms TIMES;
#endif

        if (t == NULL) return(TRUE);

        if (! isvar(Arg2(t))) return(FALSE);    /* second arg = var */
        p = &e[vnumber(Arg2(t))];
        if (p->p_body != NULL) return(FALSE);   /* second arg-->var */

#if SUN4 == 1
        CONSTRAINT_OLD_TIME = clock();
#else
#if CPUTIME != 0
        times(&TIMES);
        CONSTRAINT_OLD_TIME =  TIMES.tms_stime + TIMES.tms_utime;
#endif
#endif
        p->p_env = Nenv(0);     /*  cf. 'q' in termset()  */
        up_init();
        tt = Arg1(t);
        ee = e;
        down(q,tt,ee);
        if (tt == NIL){
          p->p_body = NIL;
          p->p_env = NULL;
#if SUN4 == 1
          CONSTRAINT_HANDLING_TIME += clock()-CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
          times(&TIMES);
          CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
            CONSTRAINT_OLD_TIME;
#endif
#endif
          return(TRUE);
        }

        clist = c = Nclause(termset(head_of_list(tt),ee,TEMPORAL),
                            (struct clause *)NULL,TEMPORAL);
        while (1) {
          tt = tail_of_list(tt);
```

```
        down(q,tt,ee);
        if ((tt == NIL) || (! is_list(tt))) break;
        e->e_link = Nclause(termset(head_of_list(tt),ee,TEMPORAL),
                            (struct clause *)NULL,TEMPORAL);
        e = e->e_link;
      }

      if (tt != NIL) {
            up_restore();
            p->p_env = NULL;
            error("Illegal form of constraint list.");
      }

      if (p_number != 0) {
        renum_pvars((struct pstvar *)pv_list,v_number);
        q=Nenv(p_number);
      }
      up_restore();

      e = startmodular(clist,v_list,v_number(p_number); /* transformation */

      if (e == MFAIL){        /* fail transformation */
            p->p_env = NULL;
#if SUN4 == 1
            CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
            times(&TIMES);
            CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
              CONSTRAINT_OLD_TIME;
#endif
#endif
            return(FALSE);  /* fail */
      }
      else if (e == NULL)
      {
            p->p_body = NIL;
            p->p_env = NULL;
#if SUN4 == 1
            CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
            times(&TIMES);
            CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime
              CONSTRAINT_OLD_TIME;
#endif
#endif
            return(TRUE);
      }
      else
      {
            p->p_body = tolist(e,STINGY);
TB
            tprint0("   ===> ");
            Plem(p->p_body, p->p_env);
TE
#if SUN4 == 1
            CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
            times(&TIMES);
            CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
              CONSTRAINT_OLD_TIME;
#endif
#endif
            return(TRUE);   /* success */
      }
}

/*  change clause(t1,t2,t3)  to list([t1,t2,t3])  <- eu() */
struct term *tolist(c,flag)
struct clause *c;
int flag;
{
  register struct clause *cc, *croot;

  if (c == NULL) return(NIL);

  switch (flag) {
    case STINGY:
        for (cc = c; cc->e_link != NULL; cc=cc->e_link)
            cc->e_type = LIST_TYPE;
        cc->e_link = (struct clause *)NIL;
        return((struct term *)c);
    default:
        croot=cc=Nlist(head_of_list(c),(struct clause *)NIL,flag);
        while (c->e_link != NULL) {
           c=c->e_link;
           cc->e_link=Nlist(head_of_list(c),(struct clause *)NIL,flag);
           cc = cc->e_link;
         }
        return((struct term *)croot);
     }
}

/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   transform(precond,newc,newenv)
   constraint solver of CAMC . called by resolve() in refute.c
      precond: old constraint (from goal)
      newc, newenv: new constraint (from program)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/* constraint transformation entry <- resolve() */
struct eclause *transform(precond, newc, newenv)
struct eclause *precond;
struct clause *newc;
struct pair *newenv;
{
    struct eclause *eclause_Append();
        struct eclause *cond;
        struct clause *c,*clist;
```

```
        struct pair *env,*q;
#if CPUTIME != 0
        struct tms TIMES;
#endif

        if (precond == NULL && newe == NULL) return(NULL);

#if SUN4 == 1
        CONSTRAINT_OLD_TIME = clock();
#else
#if CPUTIME != 0
        times(&TIMES);
        CONSTRAINT_OLD_TIME =  TIMES.tms_stime + TIMES.tms_utime;
#endif
#endif
  cond = eclause_append(precond, reduce_clause(newe,newenv));
    if (cond == (struct eclause *)MFAIL) /* reduce clause failure */
    {
            TB tprint0("fail (reduction)"); TE
            return((struct eclause *)MFAIL);
    }
  env = Nenv(0);
  up_init();
  clist = up_eclause(cond,MEDIUM); /* set clause */
  up_restore();
  if (p_number != 0) {
    renum_pvars((struct pstvar *)pv_list,v_number);
    q=Nenv(p_number);  /* 1991-03-10 */
  }
  if (clist == NULL) {
#if SUN4 == 1
          CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
          times(&TIMES);
          CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
            CONSTRAINT_OLD_TIME;
#endif
#endif

          return(NULL); /* no constraint */
        }
TB
  tprint0(">>transform: ");
  Peclause(cond);
  tprint0(" ==> ");
TE
  c = startmodular(clist,v_list,v_number(p_number);

  if (c == MFAIL) {     /* constraint transformation failure */
          TB   tprint0("fail"); TE
#if SUN4 == 1
          CONSTRAINT_HANDLING_TIME += clock()-CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
          times(&TIMES);
          CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
            CONSTRAINT_OLD_TIME;
#endif
#endif
        return((struct eclause *)MFAIL);
  }

  cond = reduce_clause(c,env);
  if (cond == (struct eclause *)NULL) {
          TB tprint0("null"); TE
  }
  else if (cond == (struct eclause *)MFAIL)
  {
          TB tprint0("fail (reduction)"); TE
  }
  else  {
          TB Peclause(cond); TE
  }
#if SUN4 == 1
    CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
    times(&TIMES);
    CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
      CONSTRAINT_OLD_TIME;
#endif
#endif
  return(cond);
}

struct eclause *eclause_append(head,tail) /* <- transform() */
register struct eclause *head, *tail;
{
  if (head == (struct eclause *)MFAIL ||
      tail == (struct eclause *)MFAIL)
          return((struct eclause *)MFAIL);
  while (head != (struct eclause *)NULL) {
    tail = Neclause(head->e_form,head->e_env,tail,TEMPORAL);
    head = head->e_link;
  }
  return(tail);
}


/*  -----  --  term set ----------  --  */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  struct term *termset(t,e,flag)
  struct clause *up_eclause(ec,flag)
   make variant of terms with an environment
   Before termset, up_init() and after termset,  up_restore().
   Before termset, set p=Nenv(0), then p will be a unifier.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct up_log
{
        struct term *u_old,*u_new;
```

```
        struct pair *u_oldenv;
        struct up_log *u_link;
};

struct ustack *ustack_save_up; /* save old up */
struct up_log *UP_log,*UP_pstlog;                /* up log list */

void up_init()
{
        ustack_save_up = up;
        v_number = 0; v_list = NULL;
        p_number = 0; pv_list = NULL;
        UP_log = UP_pstlog = NULL;
}

void up_restore()
{
        UP_log = UP_pstlog = NULL;
        undo(ustack_save_up);
}

void push_log(oldt,oldenv,newt)
struct term *oldt,*newt;
struct pair *oldenv;
{
        struct up_log *u;
        MEMORY_ALLOC(u,up_log,TEMPORAL);
        u->u_old = oldt;
        u->u_new = newt;
        u->u_oldenv = oldenv;
        u->u_link = UP_log;
        UP_log = u;
}

struct term *search_log(t,e)    /* search (t,e) in UP_log */
struct term *t;
struct pair *e;
{
        register struct up_log *u;
        for (u = UP_log; u != NULL; u = u->u_link)
                if (u->u_old == t && u->u_oldenv == e)
                        return(u->u_new);
        return(NULL);
}

void push_pstlog(oldt,oldenv,newt)
struct term *oldt,*newt;
struct pair *oldenv;
{
        struct up_log *u;
        MEMORY_ALLOC(u,up_log,TEMPORAL);
        u->u_old = oldt;
        u->u_new = newt;
        u->u_oldenv = oldenv;
        u->u_link = UP_pstlog;
        UP_pstlog = u;
}

struct term *search_pstlog(t,e) /* search (t,e) in UP_log */
struct term *t;
struct pair *e;
{
        register struct up_log *u;
        for (u = UP_pstlog; u != NULL; u = u->u_link)
                if (u->u_old == t && u->u_oldenv == e)
                        return(u->u_new);
        return(NULL);
}

void check_constant_term(t)
struct term *t;
{
        register int i;

        if (t->t_arity < 0) t->t_arity = -t->t_arity;
        for (i = 0; i < Prod(t)->t_arity; i++)
                if (!isconst(Arg(t,i))) return;
        t->t_arity = -t->t_arity;
}

/* term prepare for cu() ; v_list and v_number are changed */
struct term *termset(t,e,flag)
register struct term *t;
register struct pair *e;
int flag;
{
  register struct pair *p,*q;
  register struct term *nt;

  if (t == NULL) return(t);
  down(p,t,e);
  if (p != NULL)          {       /*      if t is var     */
    if (p == Anonymous_env)
      return(Anonymous_var);
    if (p->p_env == NULL)       {/*     if t is new var */
      /* use p->p_env as work area */
      upush(&(p->p_env));
      Nvar(vname(t),flag);
      p->p_env = (struct pair *)v_list;
      q = Nenv(1);
      q->p_body = t;
      q->p_env  = e;
      return(v_list);
    }
    else{
      ((struct var *)p->p_env)->v_occurrence++;
      return( (struct term *)p->p_env );
    }
  }
  if (t->type.ident == PST_TYPE)
```

```
    return(up_pst(((struct pst *)t),e,flag));
  if ((nt = search_log(t,e)) != NULL) return(nt); /* already set */
  switch (t->type.ident) {
  case ATOMIC_TYPE:
    nt = up_atomic(t,flag); /* constant term */
    break;
  case CLAUSE_TYPE:
    nt = (struct term *)Nclause(termset(head_of_list(t),e,flag),
          (struct clause *)termset(tail_of_list(t),e,flag),flag);
    break;
  case LIST_TYPE:
  case CONST_LIST_TYPE:
    nt = (struct term *)Nlist(termset(head_of_list(t),e,flag),
           (struct clause *)termset(tail_of_list(t),e,flag),flag);
    break;
  default:
    if (isconst_functor(t))
    {
        nt = up_const_functor(t,flag);
        break;
    }
    nt = Nterm(t->t_arity,flag);
    nt->type.t_func = t->type.t_func;
    {
      register int i;
      for (i = 0; i < t->t_arity; i++)
        Arg(nt,i) = termset(Arg(t,i), e, flag);
    }
    check_constant_term(nt);    /* check if nt is constant term */
  }
  push_log(t,e,nt);
  return(nt);
}

struct term *up_pst(pt,e,flag)
struct pst *pt;
struct pair *e;
int flag;
{
  struct pst_item *target;
  struct pst *nt;
  struct pstvar *pv;
  struct eclause *t0,*targetobj;
  struct pair *e0;
  struct term *oldt;

  if ((target = find_pstitem((struct term *)pt,e)) != (struct pst_item *)NULL)
  {
        t0 = target->p_lists;
        e0 = NULL;
        if ((oldt = search_pstlog(t0,e0)) != NULL) return(oldt);
        targetobj = termset_pstobj(target->p_lists,flag);
  }
  else
  {
        t0 = pt->p_lists;
        e0 = e;
        if ((oldt = search_pstlog(t0,e0)) != NULL) return(oldt);
        targetobj = termset_pstobj_sub(pt->p_lists,e,flag);
  }
  MEMORY_ALLOC(nt,pst,flag);
  nt->type = PST_TYPE;

  MEMORY_ALLOC(pv,pstvar,flag);
  pv->v_type = VAR_PST_TYPE;
  pv->v_name = vname("Anonymous var");
  pv->v_number = p_number++;
  pv->v_link = pv_list;
  pv->old_var = pt->p_var;
  nt->p_var = pv_list = (struct term *)pv;
  nt->p_lists = targetobj;
  push_pstlog(t0,e0,(struct term *)nt);
  return((struct term *)nt);
}


struct eclause *termset_pstobj(pobj,flag)
struct eclause *pobj;
int flag;
{
  if (pobj==(struct eclause *)NULL) return(pobj);
  else
    return(Npstobj(termset(pobj->e_form,pobj->e_env,flag),
                   (struct pair *)NULL,
                   termset_pstobj(pobj->e_link,flag),
                   flag));
}

struct eclause *termset_pstobj_sub(pobj,e,flag)
struct eclause *pobj;
struct pair *e;
int flag;
{
  struct eclause *pl, *ptop;

  if (pobj == (struct eclause *)NULL)
    return(pobj);
  ptop = pl = Npstobj(termset(pobj->e_form,e,flag),
                      (struct pair *)NULL,(struct eclause *)NULL,flag);
  while (pobj->e_link != (struct eclause *)NULL) {
    pl->e_link=
      Npstobj(termset(pobj->e_link->e_form,e,flag),
              (struct pair *)NULL,(struct eclause *)NULL,flag);
    pobj = pobj->e_link;
    pl = pl->e_link;
  }
  return(ptop);
}

struct term *up_const(t,flag)
```

```
register struct term *t;
int flag;
{
  struct term *up_atomic(), *up_const_functor();

  switch (t->type.ident) {
    case ATOMIC_TYPE:
      return(up_atomic(t,flag));
    case CONST_LIST_TYPE:
     return((struct term *)Nlist(up_const(head_of_list(t),flag),
                                 (struct clause *)up_const(tail_of_list(t),flag),
                                 flag));
    default: /* functor */
      return(up_const_functor(t,flag));
    }
}

struct term *up_atomic(t,flag)
register struct term *t;
int flag;
{
  register struct term *tt;

  if (in_upper_heap(t,flag)) return(t);
  tt = Nterm(0,flag);
  tt->type.ident = t->type.ident;
  tt->t_arity = t->t_arity;
  if (is_int(t)) num_value(tt) = num_value(t);
  else if (! is_string(t)) num_value(tt) = num_value(t);
  else str_value(tt) = malloc(str_value(t),flag);
  return(tt);
}

struct term *up_const_functor(t,flag)
register struct term *t;
int flag;
{
     register struct term *tt;
     register int i;

     i = -t->t_arity;
     if (i == 0)        /* constant */
     {
       if (in_upper_heap(t,flag)) return(t);
       tt = Nterm(0,flag);
       Pred(tt) = Pred(t);
       return(tt);
     }
     tt = Nterm(i,flag);

     Pred(tt) = Pred(t);
     tt->t_arity = -i;

     while (i > 0) {
          i--;
          Arg(tt,i) = up_const(Arg(t,i),flag);
     }
     return(tt);
}

struct clause *up_eclause(ec,flag)
struct eclause *ec;
int flag;
{
  if (ec == NULL) return(NULL);
  return(Nclause(termset(ec->c_form, ec->c_env,flag),
                 up_eclause(ec->c_link,flag),
                 flag));
}

struct clause *up_itrace_clause(cl,anum)
struct clause *cl;
int anum;
{
        register struct variant *va;
        struct variant *variant();

        va = variant(cl,ETERNAL);
        return(va->v_clause);
}

/*************************************************************************
  try_fold(c,n)
    try fold transformation
       c: clause
       n: # of vars + # of psts in c
    if there is HC->C in new predicate derivation clauses and Cu=cu,
       return Hu (u is a variable replacement).
    else return NULL

try_fold() is called by new_constraint() in trans.c
.    .    try_fold
.    .    .   match
.    .    .    .   match_term
.    .    .   termnumber
*************************************************************************/
struct term *try_fold(c,n)      /* fold transformation */
struct clause *c;               /* target clause */
int n;                          /* # of different vars and psts in c */
{
  register struct itrace *it;
  struct ustack *usave;
  struct term *t,*head;
  struct pair *e;
  struct pair *esave = ep;
  int j, count,termnumber();

  if (c == NULL) return(NULL);
  if (newf_list == NULL) return(NULL);
```

```
  count = literalnumber(c);  /* number of terms in c */
  usave = usp;
  esave = ep;
  e = Menv(n);
  for (it = newf_list; it != NULL; it = it->it_link ) {
        if ((it->it_anumber == n) && (it->it_cnumber == count) )
                if (match(c, it->it_clause->c_link, e) == FALSE)
                {
                        undo(usave);
                        ep = esave;
                        continue;
                }
                else{
                        if (it->it_clause->c_term == FAIL) {
                                undo(usave); ep = esave;
                                return(FAIL);
                        }
                        head = it->it_clause->c_form;
                        t = NTerm(Pred(head)->f_arity, MEDIUM);
                        Pred(t) = Pred(head);
                        for (j = 0; j < Pred(head)->f_arity ; j++) {
                              Arg(t,j) = e[termnumber(Arg(head,j))].p_body;
                              /* patch for PST var 1991 03 02 */
                              if (Arg(t,j)==NULL) Arg(t,j)=Anonymous var;
                        }
                        undo(usave); ep = esave; /* patch 1991 03-03 */
                        return(t);    /*  found something  */
                }
  }
  return(NULL);
}

int termnumber(t)
struct term *t;                 /* var or PST */
{
        if (isvar(t)) return(vnumber(t));
        else if (is_pst(t)) return(vnumber( ((struct pst *)t)->p_var));
        else printf("illegal term type : termnumber");
}

jmp_buf eqfail;         /* clause unification fail  */

int match(clo,clt,e)            /*  clause matcher  */
struct clause *clo,*clt;
struct pair *e;
{
        register struct clause *cl,*c2;

        for (cl = clo,c2 = clt; cl != NULL ; cl = cl->c_link,c2 = c2->c_link)
                if (Pred(cl->c_form) != Pred(c2->c_form))
                        return(FALSE); /* fail */

        if (setjmp(eqfail)) return(FALSE);      /* if match_term() fails */

        for (cl = clo, c2 = clt; cl != NULL; cl = cl->c_link, c2 = c2->c_link)
                match_term(cl->c_form , c2->c_form , e);
        return(TRUE);           /* success */
}

void match_term(t1,t2,e)        /*  term unification (t1,e) = (t2,e) */
struct term *t1,*t2;
struct pair *e;                 /*  return envs */
{
        register struct pair *p;

        if (isvar(t2)) {
                if (!isvar(t1)) longjmp(eqfail,1);
                p = &e[vnumber(t2)];
                if (p->p_body == NULL) {
                        p->p_body = t1;
                        return;
                }
                else if(p->p_body == t1) return;
                else longjmp(eqfail,1);
        }
        else if (isvar(t1)) longjmp(eqfail,1);

        if (Pred(t1) != Pred(t2)) longjmp(eqfail,1);
        switch (t1->type.ident) {
        case ATOMIC_TYPE:
          if ((t1==t2) ||(atomic_equal(t1,t2))) return;
          else longjmp(eqfail,1);
        case LIST_TYPE:
        case CONST_LIST_TYPE:
          match_term(head_of_list(t1),head_of_list(t2),e);
          match_term(tail_of_list(t1),tail_of_list(t2),e);
          return;
        case CLAUSE_TYPE:
          while ((t1!=NULL) && (t2!=NULL)) {
            match_term(head_of_list(t1),head_of_list(t2),e);
            t1=tail_of_list(t1);
            t2=tail_of_list(t2);
          }
          if (t1==t2) return;
          else longjmp(eqfail,1);
        case PST_TYPE:
          if (t2->type.ident != PST_TYPE) longjmp(eqfail,1);
          p = &e[termnumber(t2)];
          if (p->p_body == NULL)
          {
                match_term((struct term *)((struct pst *)t1)->p_lists,
                           (struct term *)((struct pst *)t2)->p_lists,e);
                p->p_body = t1;
                return;
          }
          else if (p->p_body == t1) return;
          else if (p->p_body != t1) longjmp(eqfail,1);
        case RCLAUSE_TYPE: /* pst_objects */
          while ((t1!=NULL) && (t2!=NULL)) {
            match_term(((struct rclause *)t1)->c_form,
```

```
            ((struct eclause *)t2)->c_form,e);
          t1=(struct term *)((struct eclause *)t1)->c_link;
          t2=(struct term *)((struct eclause *)t2)->c_link;
        }
        if (t1==t2) return;
        else longjmp(eqfail,1);
      default: {
        register int i,j = t1->t_arity;
        if (j < 0) j = -j;
        for(i = 0 ; i<j ; i++)
          match_term(Arg(t1,i), Arg(t2,i), e);
      }
    }
}
```

```
/*-------------------------------------------------------------------
*               cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989--91
-------------------------------------------------------------------- */
/*-------------------------------------------------------------------
*               << modular.c >>
*               constraint transformation entry, tools
-------------------------------------------------------------------*/

#include "include.h"

#if SUN4 == 1
#include <sys/time.h>
#else
#if CPUTIME != 0
#include <sys/types.h>
#include <sys/times.h>
#endif
#endif

#define in_cheap(X) (( &cheap[0] <= ((int *)X)) && (((int *)X) < clp))
#define in_upper_heap(X,Y) ( in_sheap(X) || ( (Y==MEDIUM) && in_cheap(X)) )

long CONSTRAINT_OLD_TIME;

/*-------------------------------------------------------------------
  modular(c)
   c mode entry
-------------------------------------------------------------------*/
void modular(c,vlist,anum)      /* constraint transf. from top level (c) */
struct clause *c;
struct term *vlist;
int anum;
{
        struct clause *sol;

        sol = startmodular(c, vlist, anum); /* tranformation */
        tprint0("solution = ");
        if (sol == MFAIL){      /* fail transformation */
                tprint0("fail.\n");
        }
        else if (sol == NULL){  /* nil constraint */
                tprint0("nil (true).\n");
        }
        else                    /* e.t. success */
        {
                Pclause(sol, (struct pair *)NULL); NL;
                show_newdefs(); /* print NEF_list */
        }
}


/*-------------------------------------------------------------------
   cu(t,e)
     process unify() built-in predicate
-------------------------------------------------------------------*/
/* constraint transformation embedded in Prolog : unify() pred. */
jmp_buf tolist_fail;

int cu(t,e)             /*  0: cu fail  1: success   */
struct term *t;
struct pair *e;
{
        register struct pair *p, *q;
        struct pair *ee;
        struct term *tt;
        struct clause *c, *clist;
#if CPUTIME != 0
        struct tms TIMES;
#endif

        if (t == NULL) return(TRUE);

        if (! isvar(Arg2(t))) return(FALSE);    /* second arg = var */
        p = &e[vnumber(Arg2(t))];
        if (p->p_body != NULL) return(FALSE);   /* second arg-->var */

#if SUN4 == 1
        CONSTRAINT_OLD_TIME = clock();
#else
#if CPUTIME != 0
        times(&TIMES);
        CONSTRAINT_OLD_TIME =  TIMES.tms_stime + TIMES.tms_utime;
#endif
#endif
        p->p_env = Newv(0);     /*  cf. 'q' in termset()  */
        up_init();
        tt = Arg1(t);
        ee = e;
        down(q,tt,ee);
        if (tt == NIL){
          p->p_body = NIL;
          p->p_env = NULL;
#if SUN4 == 1
          CONSTRAINT_HANDLING_TIME += clock()-CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
          times(&TIMES);
          CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
            CONSTRAINT_OLD_TIME;
#endif
#endif
          return(TRUE);
        }

        clist = c = Nclause(termset(head_of_list(tt),ee,TEMPORAL),
                            (struct clause *)NULL,TEMPORAL);
        while (1) {
          tt = tail_of_list(tt);
```

```
        down(q,t1,ee);
        if ((t1 == NIL) || (! is_list(t1))) break;
        e->e_link = Nclause(termcat(head_of_list(t1),ee,TEMPORAL),
                            (struct clause *)NULL,TEMPORAL);
        e = e->e_link;
      }

      if (t1 != NIL) {
              up_restore();
              p->p_env = NULL;
              error("Illegal form of constraint list.");
      }

      if (p_number != 0) {
        renum_pvars((struct petvar *)pv_list.v_number);
        q=Nenv(p_number);
      }
      up_restore();

      e = startmodular(clist,v_list,v_number(p_number); /* transformation */

      if (e == MFAIL){        /* fail transformation */
              p->p_env = NULL;
#if SUN4 == 1
              CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
              times(&TIMES);
              CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime
                CONSTRAINT_OLD_TIME;
#endif
#endif
              return(FALSE);  /* fail */
      }
      else if (e == NULL)
      {
              p->p_body = NIL;
              p->p_env = NULL;
#if SUN4 == 1
              CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
              times(&TIMES);
              CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime
                CONSTRAINT_OLD_TIME;
#endif
#endif
              return(TRUE);
      }
      else
      {
              p->p_body = tolist(e,STINGY);
TB
              fprintf("   ---> ");
              Pterm(p->p_body, p->p_env);
IE
#if SUN4 == 1
              CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
              times(&TIMES);
              CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime
                CONSTRAINT_OLD_TIME;
#endif
#endif
              return(TRUE);   /* success */
      }
}

/*  change clause(t1,t2,t3)  to list([t1,t2,t3])  <- cu() */
struct term *tolist(e,flag)
struct clause *e;
int flag;
{
  register struct clause *ee, *eroot;

  if (e == NULL) return(NIL);

  switch (flag) {
    case STINGY:
       for (ee = e; ee->e_link != NULL; ee=ee->e_link)
           ee->e_type = LIST_TYPE;
       ee->e_link = (struct clause *)NIL;
       return((struct term *)e);
    default:
       eroot=ee=Nlist(head_of_list(e),(struct clause *)NIL,flag);
       while (e->e_link != NULL) {
          e=e->e_link;
          ee->e_link=Nlist(head_of_list(e),(struct clause *)NIL,flag);
          ee = ee->e_link;
       }
       return((struct term *)eroot);
    }
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  transform(precond,newc,newenv)
  constraint solver of CAIC , called by resolve() in refute.c
    precond: old constraint (from goal)
    newc, newenv: new constraint (from program)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* constraint transformation entry <- resolve() */
struct eclause *transform(precond, newc, newenv)
struct eclause *precond;
struct clause *newc;
struct pair *newenv;
{
    struct eclause *eclause_append();
        struct eclause *cond;
        struct clause *e,*clist;
```

```
        struct pair *env,*q;
#if CPUTIME != 0
        struct tms TIMES;
#endif

        if (precond == NULL && newc == NULL) return(NULL);

#if SUN4 == 1
        CONSTRAINT_OLD_TIME = clock();
#else
#if CPUTIME != 0
        times(&TIMES);
        CONSTRAINT_OLD_TIME = TIMES.tms_stime + TIMES.tms_utime;
#endif
#endif
  cond = eclause_append(precond, reduce_clause(newc,newenv));
    if (cond == (struct eclause *)MFAIL) /* reduce_clause failure */
    {
            TB tprint0("fail (reduction)"); TE
            return((struct eclause *)MFAIL);
    }
  env = Nenv(0);
  up_init();
  clist = up_eclause(cond,MEDIUM); /* set clause */
  up_restore();
  if (p_number != 0) {
    tenum_pvars((struct pstvar *)pv_list.v_number);
    q=Nenv(p_number);  /* 1991 03 10 */
  }
  if (clist == NULL) {
#if SUN4 == 1
          CONSTRAINT_HANDLING_TIME += clock() CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
          times(&TIMES);
          CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime
            CONSTRAINT_OLD_TIME;
#endif
#endif
          return(NULL); /* no constraint */
        }
TB
  tprint0(">>transform: ");
  Peclause(cond);
  tprint0(" ==> ");
TE
  c = startmodular(clist,v_list.v_number+p_number);

  if (c == MFAIL) {      /* constraint transformation failure */
          TB   tprint0("fail"); TE
#if SUN4 == 1
        - CONSTRAINT_HANDLING_TIME += clock()-CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
          times(&TIMES);
          CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
            CONSTRAINT_OLD_TIME;
#endif
#endif
          return((struct eclause *)MFAIL);
  }

  cond = reduce_clause(c,env);
  if (cond == (struct eclause *)NULL) {
          TB tprint0("null"); TE
  }
  else if (cond == (struct eclause *)MFAIL)
  {
          TB tprint0("fail (reduction)"); TE
  }
  else  {
          TB Peclause(cond); TE
  }
#if SUN4 == 1
    CONSTRAINT_HANDLING_TIME += clock()-CONSTRAINT_OLD_TIME;
#else
#if CPUTIME != 0
    times(&TIMES);
    CONSTRAINT_HANDLING_TIME += TIMES.tms_stime + TIMES.tms_utime -
      CONSTRAINT_OLD_TIME;
#endif
#endif
  return(cond);
}

struct eclause *eclause_append(head,tail) /* <- transform() */
register struct eclause *head, *tail;
{
  if (head == (struct eclause *)MFAIL ||
      tail == (struct eclause *)MFAIL)
          return((struct eclause *)MFAIL);
  while (head != (struct eclause *)NULL) {
    tail = Neclause(head->e_form,head->e_env,tail,TEMPORAL);
    head = head->e_link;
  }
  return(tail);
}


/*  -----     term set  -----  */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  struct term *termset(t,e,flaq)
  struct clause *up_eclause(ec,flaq)
   make variant of forms with an environment
   Before termset, up_init() and after termset,  up_restore().
   Before termset, set p=Nenv(0), then p will be a unifier.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct up_log
{
        struct term *u_old,*u_new;
```

```
        struct pair *u_oldenv;
        struct up_log *u_link;
};

struct ustack *ustack_save_up; /* save old usp */
struct up_log *UP_log,*UP_pstlog;                /* up log list */

void up_init()
{
        ustack_save_up = usp;
        v_number = 0; v_list = NULL;
        p_number = 0; pv_list = NULL;
        UP_log = UP_pstlog = NULL;
}

void up_restore()
{
        UP_log = UP_pstlog = NULL;
        undo(ustack_save_up);
}

void push_log(oldt,oldenv,newt)
struct term *oldt,*newt;
struct pair *oldenv;
{
        struct up_log *u;
        MEMORY_ALLOC(u,up_log,TEMPORAL);
        u->u_old = oldt;
        u->u_new = newt;
        u->u_oldenv = oldenv;
        u->u_link = UP_log;
        UP_log = u;
}

struct term *search_log(t,e)    /* search (t,e) in UP_log */
struct term *t;
struct pair *e;
{
        register struct up_log *u;
        for (u = UP_log; u != NULL; u = u->u_link)
                if (u->u_old == t && u->u_oldenv == e)
                        return(u->u_new);
        return(NULL);
}

void push_pstlog(oldt,oldenv,newt)
struct term *oldt,*newt;
struct pair *oldenv;
{
        struct up_log *u;
        MEMORY_ALLOC(u,up_log,TEMPORAL);
        u->u_old = oldt;
        u->u_new = newt;
        u->u_oldenv = oldenv;
        u->u_link = UP_pstlog;
        UP_pstlog = u;
}

struct term *search_pstlog(t,e) /* search (t,e) in UP_log */
struct term *t;
struct pair *e;
{
        register struct up_log *u;
        for (u = UP_pstlog; u != NULL; u = u->u_link)
                if (u->u_old == t && u->u_oldenv == e)
                        return(u->u_new);
        return(NULL);
}

void check_constant_term(t)
struct term *t;
{
        register int i;

        if (t->t_arity < 0) t->t_arity = -t->t_arity;
        for (i = 0; i < Pred(t)->f_arity; i++)
                if (!isGround(Arg(t,i))) return;
        t->t_arity = -t->t_arity;
}

/* term_prepare_for_eu() : v_list and v_number are changed */
struct term *termset(t,e,flag)
register struct term *t;
register struct pair *e;
int flag;
{
  register struct pair *p,*q;
  register struct term *nt;

  if (t == NULL) return(t);
  down(p,t,e);
  if (p != NULL)          |       /*      if t is var     */
    if (p == Anonymous_env)
      return(Anonymous_var);
    if (p->p_env == NULL)       [/*     if t is new var */
      /* use p->p_env as work area */
      upush(&(p->p_env));
      Nvar(vname(t),flag);
      p->p_env = (struct pair *)v_list;
      q = Newv(1);
      q->p_body = t;
      q->p_env  = e;
      return(v_list);
    }
    else{
      ((struct var *)p->p_env)->v_occurrence++;
      return( (struct term *)p->p_env );
    }
  }
  if (t->type.ident == PST_TYPE)
```

```
    return(up_pst(((struct pst *)t),e,flag));
  if ((nt = search_log(t,e)) != NULL) return(nt), /* already set */
  switch (t->type.ident) {
  case ATOMIC_TYPE:
    nt = up_atomic(t,flag); /* constant term */
    break;
  case CLAUSE_TYPE:
    nt = (struct term *)Nclause(termset(head_of_list(t),e,flag),
           (struct clause *)termset(tail_of_list(t),e,flag),flag);
    break;
  case LIST_TYPE:
  case CONST_LIST_TYPE:
    nt = (struct term *)Nlist(termset(head_of_list(t),e,flag),
           (struct clause *)termset(tail_of_list(t),e,flag),flag);
    break;
  default:
    if (isvcnst_functor(t))
    {
        nt = up_const_functor(t,flag);
        break;
    }
    nt = Nterm(t->t_arity,flag);
    nt->type.t_func = t->type.t_func;
    {
      register int i;
      for (i = 0; i < t->t_arity; i++)
        Arg(nt,i) = termset(Arg(t,i), e, flag);
    }
    check_constant_term(nt);    /* check if nt is constant term */
  }
  push_log(t,e,nt);
  return(nt);
}

struct term *up_pst(pt,e,flag)
struct pst *pt;
struct pair *e;
int flag;
{
  struct pst_item *target;
  struct pst *nt;
  struct pstvar *pv;
  struct eclause *t0,*targetobj;
  struct pair *e0;
  struct term *oldt;

  if ((target = find_pstitem((struct term *)pt,e)) != (struct pst_item *)NULL)
  {
        t0 = target->p_lists;
        e0 = NULL;
        if ((oldt = search_pstlog(t0,e0)) != NULL) return(oldt);
        targetobj = termset_pstobj(target->p_lists,flag);
  }
  else
  {
        t0 = pt->p_lists;
        e0 = e;
        if ((oldt = search_pstlog(t0,e0)) != NULL) return(oldt);
        targetobj = termset_pstobj_sub(pt->p_lists,e,flag);
  }
  MEMORY_ALLOC(nt,pst,flag);
  nt->type = PST_TYPE;

  MEMORY_ALLOC(pv,pstvar,flag);
  pv->v_type = VAR_PST_TYPE;
  pv->v_name = vname(Anonymous_var);
  pv->v_number = p_number++;
  pv->v_link = pv_list;
  pv->old_var = pt->p_var;
  nt->p_var = pv_list = (struct term *)pv;
  nt->p_lists = targetobj;
  push_pstlog(t0,e0,(struct term *)nt);
  return((struct term *)nt);
}


struct eclause *termset_pstobj(pobj,flag)
struct eclause *pobj;
int flag;
{
  if (pobj==(struct eclause *)NULL) return(pobj);
  else
    return(Npstobj(termset(pobj->e_form,pobj->e_env,flag),
                   (struct pair *)NULL,
                   termset_pstobj(pobj->e_link,flag),
                   flag));
}

struct eclause *termset_pstobj_sub(pobj,e,flag)
struct eclause *pobj;
struct pair *e;
int flag;
{
  struct eclause *pl, *ptop;

  if (pobj == (struct eclause *)NULL)
    return(pobj);
  ptop = pl = Npstobj(termset(pobj->e_form,e,flag),
                      (struct pair *)NULL,(struct eclause *)NULL,flag);
  while (pobj->e_link != (struct eclause *)NULL) {
    pl->e_link=
      Npstobj(termset(pobj->e_link->e_form,e,flag),
              (struct pair *)NULL,(struct eclause *)NULL,flag);
    pobj = pobj->e_link;
    pl = pl->e_link;
  }
  return(ptop);
}

struct term *up_const(t,flag)
```

```
register struct term *t;
int flag;
{
  struct term *up_atomic(), *up_const_functor();

  switch (t->type.ident) {
    case ATOMIC_TYPE:
      return(up_atomic(t,flag));
    case CONST_LIST_TYPE:
     return((struct term *)Nlist(up_const(head_of_list(t),flag),
                                 (struct clause *)up_const(tail_of_list(t),flag),
                                 flag));
    default: /* functor */
      return(up_const_functor(t,flag));
    }
}

struct term *up_atomic(t,flag)
register struct term *t;
int flag;
{
  register struct term *tt;

  if (in_upper_heap(t,flag)) return(t);
  tt = Nterm(0,flag);
  tt->type.ident = t->type.ident;
  tt->t_arity = t->t_arity;
  if (is_int(t)) num_value(tt) = num_value(t);
  else if (! is_string(t)) num_value(tt) = num_value(t);
  else str_value(tt) = nalloc(str_value(t),flag);
  return(tt);
}

struct term *up_const_functor(t,flag)
register struct term *t;
int flag;
{
     register struct term *tt;
     register int i;

     i = -t->t_arity;
     if (i == 0)        /* constant */
     {
       if (in_upper_heap(t,flag)) return(t);
       tt = Nterm(0,flag);
       Pred(tt) = Pred(t);
       return(tt);
     }
     tt = Nterm(i,flag);

     Pred(tt) = Pred(t);
     tt->t_arity = -i;

     while (i > 0) {
          i--;
          Arg(tt,i) = up_const(Arg(t,i),flag);
        }
     return(tt);
}

struct clause *up_eclause(ec,flag)
struct eclause *ec;
int flag;
{
  if (ec == NULL) return(NULL);
  return(Nclause(termset(ec->e_form, ec->e_env,flag),
                 up_eclause(ec->e_link,flag),
                 flag));
}

struct clause *up_itrace_clause(cl,anum)
struct clause *cl;
int anum;
{
        register struct variant *va;
        struct variant *variant();

        va = variant(cl,ETERNAL);
        return(va->v_clause);
}

/*************************************************************************
  try_fold(c,n)
    try fold transformation
       c: clause
       n: # of vars + # of psts in c
    if there is H(=>C in new predicate derivation clauses and Cu=cu,
       return Hu (u is a variable replacement).
    else return NULL

try_fold() is called by new_constraint() in trans.c
.   .   .   try_fold+
.   .   .   .   match+
.   .   .   .   .   match_term+
.   .   .   .   termnumber
*************************************************************************/
struct term *try_fold(c,n)      /* fold transformation */
struct clause *c;               /* target clause */
int n;                          /* # of different vars and psts in c */
{
  register struct itrace *it;
  struct ustack *usave;
  struct term *t,*head;
  struct pair *e;
  struct pair *esave = ep;
  int j, count,termnumber();

  if (c == NULL) return(NULL);
  if (newf_list == NULL) return(NULL);
```

```
  count = literalnumber(c);  /* number of terms in c */
  usave = usp;
  esave = ep;
  e = Newv(n);
  for (it = newf_list; it != NULL; it = it->it_link ) {
        if ((it->it_number == n) && (it->it_counter < count) )
              if (match(c, it->it_clause->c_link, e) != FALSE)
              {
                    undo(usave);
                    ep = esave;
                    continue;
              }
              else{
                    if (it->it_clause->c_form == FAIL) {
                          undo(usave); ep = esave;
                          return(FAIL);
                    }
                    head = it->it_clause->c_form;
                    t = Nterm(Pred(head)->f_arity, PSETOM);
                    Pred(t) = Pred(head);
                    for (j = 0; j < Pred(head)->f_arity ; j++) {
                          Arg(t,j) = eltermnumber(Arg(head,j))[p_body;
                          /* patch for PSF var 1991 03-02 */
                          if (Arg(t,j)==NULL) Arg(t,j)=Anonymous_var;
                    }
                    undo(usave); ep = esave; /* patch 1991 03 03 */
                    return(t);    /*  found something  */
              }
  }
  return(NULL);
}

int termnumber(t)
struct term *t;                 /* var or PST */
{
      if (isvar(t)) return(vnumber(t));
      else if (is_pst(t)) return(vnumber( ((struct pst *)t)->p_var));
      else printf("illegal term type : termnumber");
}

jmp_buf eqfail;         /* clause unification fail  */

int match(clo,clt,e)            /*  clause matcher  */
struct clause *clo,*clt;
struct pair *e;
{
      register struct clause *cl,*c2;

      for (cl = clo,c2 = clt; cl != NULL ; cl = cl->c_link,c2 = c2->c_link)
            if (Pred(cl->c_form) != Pred(c2->c_form))
                  return(FALSE); /* fail */

      if (setjmp(eqfail)) return(FALSE);      /* if match_term() fails */

      for (cl = clo, c2 = clt; cl != NULL; cl = cl->c_link, c2 = c2->c_link)
            match_term(cl->c_form , c2->c_form , e);
      return(TRUE);           /* success */
}

void match_term(t1,t2,e)        /*  term unification (t1,e) = (t2,e) */
struct term *t1,*t2;
struct pair *e;                 /*  return envs */
{
      register struct pair *p;

      if (isvar(t2)) {
            if (!isvar(t1)) longjmp(eqfail,1);
            p = &e[vnumber(t2)];
            if (p->p_body == NULL) {
                  p->p_body = t1;
                  return;
            }
            else if(p->p_body == t1) return;
            else longjmp(eqfail,1);
      }
      else if (isvar(t1)) longjmp(eqfail,1);

      if (Pred(t1) != Pred(t2)) longjmp(eqfail,1);
      switch (t1->type.ident) {
      case ATOMIC_TYPE:
        if ((t1==t2) ||(atomic_equal(t1,t2))) return;
        else longjmp(eqfail,1);
      case LIST_TYPE:
      case CONST_LIST_TYPE:
        match_term(head_of_list(t1),head_of_list(t2),e);
        match_term(tail_of_list(t1),tail_of_list(t2),e);
        return;
      case CLAUSE_TYPE:
        while ((t1!=NULL) && (t2!=NULL)) {
          match_term(head_of_list(t1),head_of_list(t2),e);
          t1=tail_of_list(t1);
          t2=tail_of_list(t2);
        }
        if (t1==t2) return;
        else longjmp(eqfail,1);
      case PST_TYPE:
        if (t2->type.ident != PST_TYPE) longjmp(eqfail,1);
        p = &e[termnumber(t2)];
        if (p->p_body == NULL)
        {
              match_term((struct term *)((struct pst *)t1)->p_lists,
                         (struct term *)((struct pst *)t2)->p_lists,e);
              p->p_body = t1;
              return;
        }
        else if (p->p_body == t1) return;
        else if (p->p_body != t1) longjmp(eqfail,1);
      case ECLAUSE_TYPE: /* pst_objects */
        while ((t1!=NULL) && (t2!=NULL)) {
          match_term((struct eclause *)t1)->c_form,
```

```
                ((struct eclause *)t2)->c.form,e);
            t1=(struct term *)((struct eclause *)t1)->c.link;
            t2=(struct term *)((struct eclause *)t2)->c.link;
          }
          if (t1==t2) return;
          else longjmp(eqfail,1);
        default: {
          register int i,j = t1->t_arity;
          if (j < 0) j = -j;
          for(i = 0 ; i<j ; i++)
            match_term(Arg(t1,i), Arg(t2,i), e);
        }
      }
}
```

```
/*-----------------------------------------------------------------------
*                 on Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                              1989 91
-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------
*                    << trans.c >>
*                constraint transformation module
-----------------------------------------------------------------------*/

#include "include.h"
#define DEBUG 0                 /* if debug, 1 */

struct cset *DEF_list;          /* sets of new predicate derivation clauses */
struct cset *CSTR_list;         /* sets of non-modular clauses */
struct cset *INITDEF_list;              /* sets of non modular clauses */
int CSTR_number;
struct clause *CONST_literals, *REST_literals; /* used in split, attach */
struct ltrace *newfsave;        /* old newf list */

jmp_buf trans_fail; /* transformation failure */
jmp_buf split_fail;

/*
[startmodular]-
.   abandon_transformation
.     <index_newflist>  -> new.c
.     reducedfun!
.   add_to_set!
.     index_newflist~
.     <add_cs_to_set>   -> tr_sub.c
.   clear_up_DEF
.     remove_from_CSTR
.   end_unfoldfold!
.     <recale_component>!     -> mainsub.c
.   [oldunfold]
.     <P_status>!     ---> tr_sub.c
.     set_temporal_def
.     <skip_asking>!        -> tr_sub.c
.     check_INITDEF?
.     is_modular_head!
.       vheadoccurrence,
.     unfold_cstr!
.       [apply]
.       [target_literal]
.       from_to!
.       insert_cs!
.       reorder.
.     unfold_derivation!
.       [apply]
.       [target_literal]
.       literalnumber~
.       reorder~
.   modular_form
.     <satisfiable>         -> tr_sub.c
.     <split>!      ----->  > split.c
.     surface_copy_clause!
.     Pump!
.     new_constraint!
.       <try_fold>! ---> modular.c
.       <variant_v>!  --> tr_split.c
.       new_pred_set!
.         newpred!
.         set_new_def!
.         vpair_length!
.     set_const_pst!
.   init_unfoldfold!
*/

/*-----------------  begin init & end unfoldfold ----------------*/
void init_unfoldfold()
{
        newfsave = newf_list;   /* save newf_list */
        DEF_list = NULL;        /* derivation clauses */
        CSTR_list = NULL;       /* new clauses */
        CSTR_number = 0;        /* initial clause number  */
        INITDEF_list = NULL;    /* initial derivation clauses */
}

void end_unfoldfold()
{
        void recale_component(); /* mainsub.c */

        recale_component();
        CSTR_list = NULL;       /* new clauses */
        CSTR_number = 0;        /* initial clause number  */
        INITDEF_list = NULL;    /* initial derivation clauses */
}

                                /* when transformation fails. */
void abandon_transformation()   /* abandon all new predicates */
{
        register struct cset *cs;
        register struct lnne *f;

        newf_list = newfsave;   /* restore newf_list (init_unfoldfold) */
        for (cs = INITDEF_list; cs != NULL; cs = cs->cs_link)
                if (cs->cs_status == FALSE_REGISTERED)
                {
                        f = Pred(cs->cs_clause->c_form);
                        if (f->f_integ == NULL) continue;
                        index_func(f);
                        reducedfun(f);
                        f->def_f_set = NULL;
                        f->f_unitcount = f->f_setcount = 0;
                        f->f_integ->it_link = newf_list;
                        newf_list = f->f_integ;
                }
        newf_list = index_newflist(newf_list,newfsave);
}
```

```
void quit_transformation()      /* quit transformation (in step trave) */
{
        register struct eset *es;
        register struct func *f;

        for (es = DEF_list; es != NULL; es = es->es_link)
        {
                f = Pred(es->es_clause->e_form);
                f->f_setcount = f->f_unitcount = 0; /* reset in add_es_to_set*/
                if (es->es_status == REGISTERED ||
                    es->es_status == REMOVED)
                        index_func(f); /* register into global hash table */
                else if (es->es_status == DERIVATION)
                {
                        index_func(f); /* register into global hash table */
                        add_es_to_set(es,'a');
                }
        }
        for (es = CSTR_list; es != NULL; es = es->es_link)
                if (es->es_status != REMOVED)
                        add_es_to_set(es,'a');
}
/* ---- - - -   end init & end unfold/fold              -- */


int check_INITDEF()             /* INITDEF list is satisfiable ??*/
{
        register struct eset *es;
        for(es = INITDEF_list; es != NULL; es = es->es_link)
                if (es->es_status == FALSE_REGISTERED)
                        return(FALSE);
        return(TRUE);
}

void remove_from_CSTR()         /* used in clear up DEF */
struct func *f;
{
        register struct eset *es;
        register struct clause *c;

        for (es = CSTR_list; es != (struct eset *)NULL; es = es->es_link)
                if (es->es_status == MODULAR_DEFINED ||
                    es->es_status == UNIT_DEFINED)
                        for(c = es->es_clause; c != (struct clause *)NULL;
                            c = c->c_link)
                                if (Pred(c->e_form) == f) {
                                  es->es_status = REMOVED;
                                  Pred(es->es_clause->e_form)->f_setcount--;
                                  break;
                                }
}


void clear_up_DEF()             /* delete useless clauses */
{
        register struct eset *es;
        register struct func *f;
        int changed = 1;

        while(changed == 1)
        {
                changed = 0;
                for (es = DEF_list; es != NULL; es = es->es_link)
                {
                        if (es->es_status == REMOVED){
                                f = Pred(es->es_clause->e_form);
                                if (f->f_unitcount > 0)
                                        es->es_status = REGISTERED;
                                else if (f->f_setcount == 0)
                                {
                                        changed = 1;
                                        es->es_status = FALSE_REGISTERED;
                                        if (neg->it_clause->e_form == FAIL)
                                        remove_from_CSTR(f);
                                }
                        }
                }
        }
}

void add_to_set()               /* register definition clauses */
{
        register struct eset *es;
        register struct func *f;

        newf_list = index_new(list(newf_list,newfsave));
        for (es = DEF_list; es != NULL; es = es->es_link)
        {
                f = Pred(es->es_clause->e_form);
                if (es->es_status == REGISTERED ||
                    es->es_status == REMOVED)
                        index_func(f); /* register into global hash table */
                f->f_setcount = f->f_unitcount = 0; /* reset in add_es_to_set*/
        }
        for (es = CSTR_list; es != NULL; es = es->es_link)
                if (es->es_status == MODULAR_DEFINED ||
                    es->es_status == UNIT_DEFINED ||
                    (!g_Resolvable && (es->es_status == TEMPORAL_DEFINED)))
                                add_es_to_set(es,'a');
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      startmodular(clist,vlist,anum)
      constraint transformation entry
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct clause *startmodular(clist, vlist, anum) /* entry */
struct clause *clist;
struct term *vlist;
int anum;
```

```
{
        int result;
        register struct clause *c;

        init_unfoldfold();      /* reset global vars */
        if (clist != NULL)
          clist = modular_form(clist,vlist,anum); /* set DEF list */
        if (clist == NULL || clist == MFAIL || DEF_list == NULL)
        {
                end_unfoldfold();
                return(clist); /* no need for transformation */
        }
        INITDEF_list = DEF_list; /* initial derivation clauses */
        if (setjmp(trans_fail)) /* quit/abort transformation */
        {
                end_unfoldfold();
                /* if clist has defs */
                for (c = clist; c != NULL; c = c->c_link) {
                  if (Pred(c->c_form)->def_set == (struct set *)NULL)
                    return(MFAIL);
                }
                return(clist);
        }
        result = foldunfold();
        if (result == TRUE)     /* transformation success */
        {
                clear_up_NEW();
                add_to_set();
                end_unfoldfold();
                return(clist);
        }
        else {                  /* transformation failure */
                abandon_transformation();
                end_unfoldfold();
                return(MFAIL);
        }
}

void Pcmp(cmp)                  /* print compartment */
struct compartment *cmp;
{
        for (; cmp != NULL; cmp = cmp->cmp_link)
        {
                Pclause(cmp->cmp_clause,NULL);NL;
        }
        if (CONST_literals != NULL) {
                printf("ground = ");
                Pclause(CONST_literals,NULL);NL;
        }
        if (REST_literals != NULL)
        {
                printf("rest = ");
                Pclause(REST_literals, NULL);NL;
        }
}
```

```
/* define new predicates */
struct clause *modular_form(clist, vlist, anum)
struct clause *clist;
struct term *vlist;
int anum;
{
        struct compartment *cmp,*cm;
        struct clause *crest, *cc;
        register struct clause *c;
        void set_const_pol();

        if (setjmp(split_fail)) /* quit/abort transformation */
          return(MFAIL);
        }

        cmp = split(surface_copy_clause(clist,TEMPORAL), vlist, anum);

        /* global vars */
        crest = REST_literals;

/*      printf("split ");
        Pclause(clist,NULL);
        printf("into\n");
        Pcmp(cmp);
        NL;
*/
        if (CONST_literals != NULL)
                if (! satisfiable(CONST_literals,anum)) return(MFAIL);
        for (cm = cmp; cm != NULL; cm = cm->cmp_link) {
          cc = c = new_constraint(cm);
          if (c == MFAIL) return(MFAIL);
          if (c == NULL) continue;
          while (c->c_link != NULL) c = c->c_link; /* c <- end of cc */
          c->c_link = crest;
          crest = cc;
        }
        return(crest);
}

/*************************************************************
  new_constraint(cmp)
  change cmp into surface modular form by
    making new predicates or folding

.   .   new_constraint
.   .   .   try_fold
.   .   .   .   match
.   .   .   .   .   match_term
.   .   .   .   termnumber
.   .   .   variant_v
.   .   .   .   Pvariant
.   .   .   new_pred_set
.   .   .   .   newpred
.   .   .   .   set_new_def
```

```
.   .   .   .   vpair length
.   .   .  variables
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct clause *new_constraint(cmp) /* make new pred or folding */
struct compartment *cmp;
{
        struct term *t;
        struct clause *c,*new_pred_set();
        struct variant *ocopy,*variant_v();

        c = cmp->cmp_clause;
        if (c == NULL) return(NULL);
        if (is_modular_clause(c)) return(c);
/* here works in future */
/*      if (!need_trans(cmp))
          return(restore_head(c->c_func,cmp->cmp_vp,cmp->cmp_vp_size)); */
        ocopy = variant_v(c,MEDIUM);
        if (ocopy->v_anum > MODULARMAX) /* heuristics  */
        {
                FILE *f = wfp;
                wfp = stderr;
                tprint0("Warning: Transformation failure by exceeding the");
                tprint0("limit of the number of variables (cl) %PM command\n");
                Pterm(c,(struct pair *)NULL),NL;
                wfp = f;
                return(MFAIL);
        }
        t = try_fold(c, ocopy->v_anum);
        if (t != NULL)
        {
TTR
                tprint0("*folding "); Pclause(c,NULL); tprint0(" --> ");
                Pterm(t,NULL);
TTE
                if (t == FAIL) return(MFAIL);
                else return(Nclause(t,NULL,MEDIUM));
        }
        return(new_pred_set(ocopy));
}

struct clause *new_pred_set(cc) /* set new pred & return */
struct variant *cc;
{
        struct func *newfunc;
        struct term *t1,*t2;
        struct clause *c1,*c2;
        register struct vpair *a;
        register int i;
        int arity,vpair_length();       /* tr_split.c */

      . while (1) {
          sprintf(nbuf, "%s%d", genname, GENSYM++);
          if (exist_fname(nbuf) == NULL) break;
        }
        arity = vpair_length(cc->v_pair);
        newfunc = Nfunc(TEMPFUN, nbuf, arity);
        newpred(newfunc);
        t1 = Nterm(arity,MEDIUM)); t1->type.t_func = newfunc; /* orig. head */
        t2 = Nterm(arity,MEDIUM)); t2->type.t_func = newfunc; /* copy head */
        c1 = Nclause(t1,NULL,MEDIUM); /* new constraint */
        c2 = Nclause(t2,cc->v_clause,MEDIUM); /* new head <- body */
        for (i = arity - 1, a = cc->v_pair; 0 <= i; i--,a=a->v_link)
        {
                Arg(t1,i) = a->v1;
                Arg(t2,i) = a->v2;
        }
        set_new_def(c2,cc->v_var,cc->v_anum); /* to DEF_list */
        return(c1);
}

void set_new_def(c,vl,anum)     /* add c to DEF_list */
struct clause *c;               /* head <- body. */
struct term *vl;
int anum;
{
        register struct cset *s;

        s = Ncset(TEMPORAL);
        s->cs_status = DERIVATION;
        s->cs_clause = c;
        s->cs_link = DEF_list;
        s->cs_anumber = anum;
        s->cs_vlist = vl;
        DEF_list = s;           /* global var */
}

int is_modular_head(t)          /* check head t is modular or not */
struct term *t;
{
        struct func *f;
        register int i;
        register struct term *arg;

        if (!is_functor(t)) {
          wfp = stderr;
          Pterm(t,(struct pair *)NULL);
          error("Only Functors can be used as Constraints");
        }
        f = Pred(t);
        for (i = 0; i < f->f_arity; i++)
        {
                arg = Arg(t,i);
                if (isvar(arg))
                {
                        if (vheadoccurrence(arg) > 1) /* double occurrence */
                                return(FALSE);
                }
                else if (is_pst(arg)) continue; /* 91.10.1 ??? */
                else return(FALSE);
        }
```

```
        return(TRUE);
}
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   foldunfold()
   fold/unfold transformation loop
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int foldunfold()                /* fold unfold transformation main */
{
        register struct cset *cs;
        struct clause *body;
        register struct func *f;
        void unfold_derivation(),unfold_cstr(),set_temporal_def();

        for (;;)
        {
                clear_up_DEF();
                if (check_INITDEF() == FALSE) return(FALSE);
TTB
                P_status();     /* print stack */
                if Is_stepp
                        if (step_asking() != 0) continue; /* user's input */
TTE

        /* target def clause */
                for (cs = DEF_list; cs != NULL; cs = cs->cs_link)
                        if (cs->cs_status == DERIVATION) break;
                if (cs != NULL) /* unfold def clause */
                {
                        unfold_derivation(cs);
                        continue;
                }
                /* get one clause from CSTR_list*/
                for (cs = CSTR_list; cs != NULL; cs = cs->cs_link)
                        if (cs->cs_status == UNTOUCHED) break;

                if (cs == NULL) return(TRUE); /* no target --> fail */
                f = Pred(cs->cs_clause->c_form);
                if (is_modular_head(cs->cs_clause->c_form))
                {
                  unfold_cstr(cs);
                  continue;
                }
                body = modular_form(cs->cs_clause->c_link,
                                    cs->cs_vlist, cs->cs_anumber);
                cs->cs_clause->c_link = body;
                if (body == MFAIL) /* fail transfomation */
                {
                        cs->cs_status = REMOVED;
                        f->f_setcount--;
                }
                else if (body == NULL) /* unit definition */
                {
                        cs->cs_status = UNIT_DEFINED;
                        f->f_unitcount++;
                  /* M-solvability */
                        if (!s_Msolvable(c)) set_temporal_def(f);
TTB
                                tprintf("*reduce body <%d>",cs->cs_number);
TTE
                }
                else
                {
                        cs->cs_status = MODULAR_DEFINED;
TTB
                        tprintf("*modularize <%d>",cs->cs_number);
TTE
                }
        }
}

void set_temporal_def(f)
struct func *f;
{
  register struct cset *cs;

  for (cs = CSTR_list; cs != (struct cset *)NULL; cs = cs->cs_link) {
    switch (cs->cs_status) {
    case UNTOUCHED:
      if (f == Pred(cs->cs_clause->c_form))
        cs->cs_status = TEMPORAL_DEFINED;
      break;
    case MODULAR_DEFINED:
      if (f == Pred(cs->cs_clause->c_link->c_form)) {
        cs->cs_status = TEMPORAL_DEFINED;
        set_temporal_def(Pred(cs->cs_clause->c_form));
      }
    }
  }
}

struct clause *reorder(cl,tc)   /* used in unfold_derivation, _cstr */
struct clause *cl,*tc;          /* cl:literals, tc:target literal */
{
        register struct clause *c;

        if (cl == (struct clause *)NULL) return((struct clause *)NULL);
        for (c = cl; c != (struct clause *)NULL; c = c->c_link)
                if (c->c_link == tc) break;
        if (c == (struct clause *)NULL) return(cl);
        else {
                c->c_link = tc->c_link;
                tc->c_link = cl;
                return(tc);
        }
}

void unfold_derivation(cs)      /* unfold derivation clause (in unfoldfold) */
struct cset *cs;
{
        struct clause *tc;
```

```
        struct func *f;
        struct itrace *it;
        int res;

        te = target_literal(cs->cs_clause->c_link);
        cs->cs_clause->c_link = reorder(cs->cs_clause->c_link,te);
        if (te == NULL) /* no need for transformation */
        {
                cs->cs_status = REGISTERED;
                return;
        }
        TTR /* print target literal */
        tprintf("*unfold [%d] ",cs->cs_number);
        Pterm(te->c_form,NULL);
        TTR
        res = apply(te->c_form,cs->cs_clause->c_form,
                    te->c_link,cs->cs_anumber);
        f = Pred(cs->cs_clause->c_form);
        f->f_unteq = it = snew(itrace);
        it->it_clause = cs->cs_clause;
        it->it_anumber = cs->cs_anumber;
        it->it_cnumber = literalnumber(cs->cs_clause->c_link);
        it->it_link = newf_list; newf_list = it;
        if (res == FALSE)
        {
                it->it_clause->c_form = FAIL;
                cs->cs_status = FALSE_REGISTERED;
                remove_from_CSTR(f);
                TTR tprint0(" ->FAIL"); TTR
        }
        else
        {
                if (f->f_unitcount > 0)
                        cs->cs_status = REGISTERED;
                else cs->cs_status = REMOVED;
                TTR tprint0(" =>TRUE"); TTR
        }
}

void insert_cs(cs,newcs)        /* used in unfold_cstr */
struct cset *cs, *newcs;        /* append newcs after cs */
{
        register struct cset *c;

        if (newcs == NULL) return;
        for (c = newcs; c->cs_link != NULL; c = c->cs_link);
                                /* c <- end of newcs */
        c->cs_link = cs->cs_link;
        cs->cs_link = newcs;
}

struct cset *from_to(s1,s2)     /* used in unfold_cstr */
struct cset *s1,*s2;            /* s1->...->c->s2 ===> s1->...->c->NULL*/
{
        struct cset *c;
        if (s1 == s2) return(NULL);
        for (c = s1; c->cs_link != s2; c = c->cs_link);
        c->cs_link = NULL;
        return(s1);
}

void unfold_cstr(cs)    /* unfold CSTR clause (in unfoldfold) */
struct cset *cs;
{
        struct clause *te;
        struct func *f;
        int res;
        struct cset *cstr_save; /* old CSTR list */

        f = Pred(cs->cs_clause->c_form);
        te = target_literal(cs->cs_clause->c_link);
        cs->cs_clause->c_link = reorder(cs->cs_clause->c_link,te);
        if (te == NULL) /* no need for transformation */
        {
                cs->cs_status = MODULAR_DEFINED;
                return;
        }
TTR /* print target literal */
        tprintf("*unfold [%d] ",cs->cs_number);
        Pterm(te->c_form,NULL);
TTR
        f->f_setcount--;
        cs->cs_status = REMOVED;
        cstr_save = CSTR_list;
        res = apply(te->c_form,cs->cs_clause->c_form,
                    te->c_link,cs->cs_anumber);
        if (res == FALSE)
        {
                TTR tprint0(" ->FAIL"); TTR
        }
        else
        {
                struct cset *newcs;
                newcs = from_to(CSTR_list,cstr_save); /* new defs */
                CSTR_list = cstr_save;
                insert_cs(cs,newcs); /* put newcs after cs */
                TTR tprint0(" =>TRUE"); TTR
        }
}

/*
[apply]---------- ------  head unification & goal replacement
   [system function]
   <system_pred>  ---> defsysp.c
   <reduce_clause>  -> tr_sub.c
   apply_add_clause
   .   <up_eclause>  ---> modular.c
   .   <satisfiable>    > tr_sub.c
   .   <add_clause>!  ---> tr_sub.c
   extend_apply!
```

```
.   .   (tunify)
.   .   eclause_conc.
*/
int apply(target,head,rest,anum) /* head: target,rest. */
struct term *target,*head;
struct clause *rest;
int anum;
{
  struct pair *e0 = Nenv(anum);
  struct func *f = Pred(target);
  struct node *dummy;
  struct eclause *ec;
  struct pair *esave = ep;
  struct ustack *usave = usp;

  if (issystem(f)) {
    dummy = Newnode(NULL,NULL,NULL,NULL,NULL);
    if (isfunc(f)) {
      if (system_function(target,e0,dummy) == SYSFAIL) {
        ep = esave;
        return(FALSE);
      }
      /* has solution */
      ec = reduce_clause(rest,e0);
      if (ec == (struct eclause *)MFAIL)
      {
              ep = esave; return(FALSE);
      }
      apply_add_clause(head,e0,ec);
      ep = esave;
      return(TRUE);
    }
    else  { /* ismonfunc */
     if (system_pred(target,e0,dummy,dummy,FIRST) == SYSFAIL)
        return(FALSE);
      /* has possibly many solutions */
      do {
        ec = reduce_clause(rest,e0);
        apply_add_clause(head,e0,ec);
        undo(usave);
        e0 = Nenv(anum);
      } while (system_pred(target,e0,dummy,dummy,BACKTRACK) == SYSTRUE);
      ep = esave;
      undo(usave);
      return(TRUE);
    }
  }
/* user predicate */
    return(extend_apply(target,head,rest,e0,1,f->def.f_set));
}

int extend_apply(target,head,rest,e0,f,s) /* in apply */
struct term *target,*head;
struct clause *rest;
struct pair *e0;
struct func *f;
register struct set *s;
{
  struct eclause *ec;
  struct ustack *usave = usp;
  int *hsave,cn;
  struct pair *el;

  if (s == (struct set *)NULL) {
    if (Handle_Undefined == TRUE) {
      sprintf(nbuf,">>> %s <<< is UNDEFINED!",f->f_name);
      error(nbuf);
    }
    else return(FALSE);
  }

  for (cn = 0 ; s != NULL; s = s->s_link) {
    hsave = hp;
    el = Nenv(s->s_number);
    if (tunify(target,e0,s->s_clause->c_term,el,2)==FALSE) /* safe unify */
    {
      undo(usave); hp = hsave;
      continue;
     }
    ec = eclause_conc(reduce_clause(s->s_clause->c_link, el),
                      eclause_conc(reduce_clause(s->s_constraint,el),
                                   reduce_clause(rest,e0)));
    if (apply_add_clause(head,e0,ec)!=FALSE)
            cn++;                 /* # of new clauses */
    undo(usave); /* restore environments */
  }
  if (cn == 0) return(FALSE);
  else return(TRUE);
}

int apply_add_clause(head,e0,ec) /* in apply, extend_apply */
struct term *head;
struct pair *e0;
struct eclause *ec;
{
  struct clause *newbody;
  struct term *newhead;
  struct pair *el;
  int i;

  if (ec == (struct eclause *)MFAIL) return(FALSE);
  up_init();
  newhead = termset(head, e0,MEDIUM);
  newbody = up_eclause(ec,MEDIUM);
  if (p_number != 0) {
      renum_pvars((struct pstvar *)pv_list,v_number);
      el=Nenv(p_number);
      i = p_number;
      while (i > 0) {
        i--;
```

```
        el[i].p_body = ((struct pstvar *)pv_list)->old_var;
        el[i].p_env = e0;
        pv_list = ((struct pstvar *)pv_list)->v_link;
      ]
    }
    up_restore();
    if (newbody != NULL && v_number == 0 && p_number == 0)
    [   /* contains no variables */
      if (satisfiable(newbody,0)) newbody = NULL;
      else return(FALSE);
    ]
  add_clause(Nclause(newhead,newbody,MRD(DUM), v_list, v_number(p_number);
  return(TRUE);
}
```

```
/*************************************************************************
*                cu-Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989 91
*************************************************************************/
/*-----------------------------------------------------------------------*
*                <<<< tr_sub.c >>>>
* subroutines for constraint transformation (called by transform.c)
*-----------------------------------------------------------------------*/

#include "include.h"
#define DEBUG 0                 /* when debug, 1 */

extern struct cset *DEF_list;
extern struct cset *CSTR_list;
extern struct cset *INITDEF_list;
extern int CSTR_number;
extern struct clause *CONST_literals, *RRST_literals;
extern struct ftrace *newfsave;
extern jmp_buf trans_fail;
extern jmp_buf split_fail;


int cs_status_type(st)
int st;
{
  static char cs_status_list[9] = {
    'u','r','m','i','d','q','f','x','t'};

  if (st > TEMPORAL_DEFINED) return('?');
  else return(cs_status_list[st]);
}

void Pset_def(cs)                       /* print cset */
struct cset *cs;
{
        tprint3("]%d(%c.%d)] ",cs->cs_number,
                cs_status_type((int)cs->cs_status),
                Pred(cs->cs_clause->c_func)->f_setcount);
        P_delause(cs->cs_clause,(struct pair *)NULL);
}

void Pset_cstr(cs)                      /* print cset */
struct cset *cs;
{
        tprint2("(%d(%c)) ",cs->cs_number,
                cs_status_type((int)cs->cs_status));
        P_delause(cs->cs_clause,(struct pair *)NULL);
}

void P_csnumber(cs,mode)
struct cset *cs;
int mode;
{
        register struct cset *c;
        int i = 0;

        for (c = cs; c != NULL; c = c->cs_link)
        if ((mode == 1 && c->cs_status == DERIVATION) ||
            (mode >= 2 && c->cs_status == UNTOUCHED) ||
            (mode == 3 && (c->cs_status == MODULAR_DEFINED ||
                           c->cs_status == UNIT_DEFINED)))
        {
                if (i != 0) {
                tputc(',');
                }
                i = 1;
                tprint1("%d",c->cs_number);
        }
}

/* ------------------------------------------------------------
   P_status()
   print DEF_list, CSTR_list

.   .   P_status()
.   .   .   P_csnumber()
.   .   .   Pset_cstr()
.   .   .   .   cs_status_type,
.   .   .   .   d.
.   .   .   Pset_def()
.   .   .   .   cs_status_type,
.   .   .   .   d.
------------------------------------------------------------ */
void P_status()  /* print DEF_list,CSTR_list (for debug) */
{
        register struct cset *cs;

        tprint0("****");
        tprint0("DEPS-{");P_csnumber(DEF_list,1);tprint0("}  ");
        tprint0("NON_MODULAR-{");P_csnumber(CSTR_list,2);tprint0("}  ");
        tprint0("MODULAR-{");P_csnumber(CSTR_list,3);tprint0("}****");
        for (cs = DEF_list; cs != NULL; cs = cs->cs_link)
        if (cs->cs_status != FALSE_REGISTERED){NL;Pset_def(cs);}
/*              if (cs->cs_status == DERIVATION) {NL;Pset_def(cs);}*/
        for (cs = CSTR_list; cs != NULL; cs = cs->cs_link)
        if (cs->cs_status != REMOVED) {NL;Pset_cstr(cs);}
/*              if (cs->cs_status == UNTOUCHED) {NL;Pset_cstr(cs);}*/
}


struct cset *Newset(flag)       /* allocate cset structure */
int flag;
{
        register struct cset *s;
        MEMORY_ALLOC(s,cset,flag);
        s->cs_clause = NULL;
        s->cs_link = NULL;
        s->cs_vlist = NULL;
        s->cs_anumber = s->cs_cnum = 0;
```

```
        s->es_status = UNTOUCHED; /* default status */
        s->es_number = CSTR_number++;
        s->es_mother = NULL;
        return(s);
}


/* ---- add c to CSTR_list (in apply_add_clause) ---- */
struct cset *CSTR_tail; /* tail of CSTR list used in add_clause() */

void add_clause(c, vlist, anum)
struct clause *c;
struct term *vlist;
int anum;
{
        register struct cset *cs;
        struct line *l;
        void set_temporal_def();

        l = Pred(c->c_form);
        l->l_setcount++;
        rename_occurrence(c,vlist);
        cs = Newset(PERMANENT);
        if (c->c_link == NULL)
        {
                l->l_unitcount++;
                cs->es_status = UNIT_DEFINED;
                if (is_Modifiable) set_temporal_def(l);
        }
        else if (is_modular_clause(c->c_link))
                cs->es_status = MODULAR_DEFINED;
        else cs->es_status = UNTOUCHED;
        cs->es_clause = c;
        cs->es_vlist = vlist;
        cs->es_anumber = Anum;
        cs->es_link = CSTR_list;
        CSTR_list = cs;
}
/*****************************************************************
    add_cs_to_set(cs,flag)
    register new predicates

.   .   <add_cs_to_set) --> tr_sub.c
.   .   .   [variant]
.   .   .   add_set
.   .   .   .   set_covers()
*****************************************************************/
void add_cs_to_set(cs,flag)     /* register new predicates  */
struct cset *cs;
int flag;
{
  register struct set *s;
  register struct variant *va;
  struct variant *variant();

  s = snew(set);
  va = variant(cs->cs_clause,ETERNAL);
  s->s_clause = va->v_clause;
  s->s_anumber = va->v_anum;
  s->s_vlist = va->v_var;
  s->s_constraint = NULL;
  s->s_bodynumber = 0;  /* set in add_set */
  add_set(s,flag);
}

/*      ---- reduction                   */
/*****************************************************************
    struct eclause *reduce_clause(cl,e)
    reduce predicates which have only one definition

.   reduce_clause
.   .   reduce_clause_m
.   .   .   [unify]
.   .   .   [system function]
.   .   .   Newclause
.   .   .   one_def_literal()
*****************************************************************/

jmp_buf reduce_fail;    /* (reduce_clause, reduce_clause_m,one_def_literal) */
struct eclause *reduce_clause(cl,e) /* entry */
struct clause *cl;
struct pair *e;
{
        struct ustack *usave = up;
        int *hsave = hp;
        struct pair *esave = ep;
        struct eclause *reduce_clause_m();

        if (setjmp(reduce_fail))
        {
                undo(usave);
                hp = hsave;
                ep = esave;
                return((struct eclause *)MFAIL);
        }
        return(reduce_clause_m(cl,e));
}

struct eclause *reduce_clause_m(cl,e) /* reduce_clause sub */
struct clause *cl;
struct pair *e;
{
        register struct set *s;
        struct pair *el,

        if (cl == NULL) return(NULL);
        if (is_functor(cl->c_form) &&
            ((Pred(cl->c_form)->l_mark & NON_UNFOLDABLE) == 0)) {
          if ((s = one_def_literal(Pred(cl->c_form))) != NULL) /* reduceable */
            {
```

```
                cl = Newv(s->s_anumber);
                if (unify(cl->c_form.e,s->s_clause->c_form,cl,1)==FALSE)
                        longjmp(reduce_fail,1);
                else
                  return(
                        eclause_conc(
                              reduce_clause(cl->c_link,e),
                              eclause_conc(
                                    reduce_clause(s->s_constraint,cl),
                                    reduce_clause(s->s_clause->c_link,cl))));
                }
          else if (is_funsys(Pred(cl->c_form))) {
            struct node *dummy;
            dummy = Newnode(NULL,NULL,NULL,NULL,NULL);
            if (system_function(cl->c_form.e,dummy) == SYSFAIL)
                  longjmp(reduce_fail,1);
            else return(reduce_clause(cl->c_link,e));
            }
          }

        return(Neclause(cl->c_form,e,reduce_clause(cl->c_link,e),MEMORY));
}

struct set *one_def_literal(f)  /* reduce_clause_m_sub */
struct func *f;
{
  if (isuser(f)) {
    switch (f->f_setcount) {
      case 1: return(f->def_f_set);
      case 0:
        if (Handle_Undefined == TRUE) {
          sprintf(mbuf,">>> %s <<< is UNDEFINED!",f->f_name);
          error(mbuf);
          }
        else longjmp(reduce_fail,1);
      default: return(NULL);
      }
  }
  else return(NULL);
}

void reorder_clause(cl, tc)
struct clause *cl, *tc;
{
        register struct clause *c;

        for (c = cl; c != NULL; c = c->c_link)
                if (c->c_link == tc) break;
        if (c == NULL) return;
        c->c_link = tc->c_link;
        tc->c_link = cl->c_link;
        cl->c_link = tc;
        return;
}
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
  int satisfiable(cl,anum)      ---> TRUE/FALSE
  check satisfiability of constraint

.   .   satisfiable
.   .   .   [refute]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int satisfiable(cl,anum)        /* satisfiability check */
struct clause *cl;
int anum;
{
  int tflagsave;                        /* save tflag */
        struct ustack *usave = usp;
        struct node *Last_Node, *Initial_Goal;

        tflagsave = tflag;      /* save prolog trace flag */
        Notrace_mode;           /* for satisfiable() */
        upush(&lap);upush(&ep);
        upush(&last_BT);upush(&Last_SKIP);
        Initial_Goal = Last_Node =
          Newnode(cl, (struct eclause *)NULL, Newv(anum),
                (struct node *)NULL,(struct node *)NULL);
        Last_BT = Last_SKIP = NULL;
        if (refute(Initial_Goal, Last_Node, DOWN) == FALSE)
        {
                undo(usave);
                tflag = tflagsave;
                return(FALSE);
        }
  undo(usave);
  tflag = tflagsave;
  return(TRUE);
}
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
struct clause *target_literal(cl)
   select unfolding target from cl

[target_literal]                                        ----------
.   Penergy+
.   energy
.   .   isconst
.   .   occurrence
.   .   isallunit+
.   greater_term
.   .   greater_arg
.   .   .   cmp_clause+
.   .   .   cmp_cplxt+
.   .   .   cmp_fl1+
.   .   .   cmp_fp+
.   .   .   cmp_int+
.   .   .   cmp_list+
.   .   .   cmp_var
.   .   .   cmp_pst+
.   .   .   cmp_str+
.   .   .   arg_type+
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
```

```
void Penergy(cl)
struct clause *cl;
{
        register struct clause *c;
        for (c = cl; c != NULL; c = c->c_link)
        {
                Pterm(c->c_form,NULL);
                fprintf("<%d>,",energy(c->c_form));
        }
}

struct clause *target_literal(cl)
struct clause *cl;
{
        register struct clause *ct, *c;
        register int cte,ce;

#if DEBUG >= 1
        printf("ENERGY: ");
        Penergy(cl);
        NL;
#endif
        if (cl->c_link == NULL) return(cl);
        ct = cl;
        cte = energy(ct->c_form);
        for (c = cl->c_link; c != NULL; c = c->c_link)
        {
                ce = energy(c->c_form);
                if (ce > cte)
                {
                        cte = ce;
                        ct = c;
                }
                else if (ce == cte && greater_term(ct->c_form,c->c_form))
                                continue;
        }
        return(ct);
}

int     Heuristic_modified = 0;
int     Factor_con,
        Factor_funct,
        Factor_vn,
        Factor_defs,
        Factor_units,
        Factor_rec,
        Factor_allunit;

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   int energy(term)
   energy of term
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int energy(tm)                  /* enegy function of literal t */
struct term *tm;
{
        int arity = 0, con = 0, vn = 0, funct = 0, rec = 0,
        allunit = 0, defs = 0, units = 0, energy;
        struct func *f;
        register struct term *t;
        register int i;

        f = Pred(tm);
        arity = f->f_arity;
        for (i = 0; i < arity; i++) {
                if (Component(f,i) == NULL) continue;
                t = Arg(tm,i);
                if (isconst(t)) con++;
                else if (isvar(t)) vn+=(occurrence(t) - 1);
                else funct++;
        }
        rec = isrecursive(f);
        allunit = isallunit(f);
        defs = f->f_defcount;
        units = f->f_unitcount;
        energy = con * 7 + arity + funct * 4 + vn * 2 - defs*2 + units*2
                - rec * 4 + allunit * 6;
        return(energy);
}

/* general predicates for transform.c */
struct clause *surface_copy_clause(cl,flag) /* make copy of cl */
struct clause *cl;
int flag;
{
        if (cl == NULL) return(NULL);
        return(Nclause(cl->c_form,
                        surface_copy_clause(cl->c_link,flag),
                        flag));
}

int is_modular_clause(cl)       /* check if clause cl is modular */
struct clause *cl;
{
        register struct clause *c;
        for (c = cl; c != NULL; c = c->c_link)
        {
                if (! is_modular_literal(c->c_form)) return(FALSE);
        }
        return(TRUE);
}

int is_modular_literal(t)       /* check if literal t is modular */
struct term *t;
{
        struct func *f;
        register int i;
        register struct term *arg;
        int has_common_label(); /* mainsub.c */
```

```
        if (! is_functor(t)) {
          wlp__stderr;
          Pterm(t,(struct pair *)NULL);
          error("Only Functors can be used as Constraints");
        }
        f = Pred(t);
        /* if (is_component_not_checked(f)) recale_f relen(f);*/
        for (i = 0; i < f->f_arity; i++)
        {
                if (Component(f,i) == NULL) continue;
                else {
                        arg = Arg(t,i);
                        if (isvar(arg) && occurrence(arg) <= 1)
                                continue;
                        if (is_pst(arg) &&
                            ! has_common_label((struct pst *)arg)->p_lists,
                                                Component(f,i)))
                                continue;
                        else
                                return(FALSE);
                }
        }
        return(TRUE);
}

int has_no_var(t)               /* check if t contains no variable */
struct term *t;
{
        register int i;

        for (i = 0; i < Pred(t)->f_arity; i++)
                if (! novar(Arg(t,i))) return(FALSE);
        return(TRUE);
}

struct eclause *eclause_cone(ec1,ec2) /* concatenate eclauses */
struct eclause *ec1,*ec2;
{
        register struct eclause *ec;

        if (ec1 == NULL) return(ec2);
        if (ec2 == NULL) return(ec1);
        if (ec1 == (struct eclause *)MFAIL||ec2 == (struct eclause *)MFAIL)
                return((struct eclause *)MFAIL);
        ec = ec1;
        while (ec->c_link != NULL)
          ec = ec->c_link;
        ec->c_link = ec2;
        return(ec1);
}

/*
* sort_clause
*   insert_clause
*       greater_term
*           greater_arg
*               arg_type
*               cmp_var,cmp_cplxt,cmp_list,cmp_flt,cmp_int,cmp_str,cmp_fp
*/
struct clause *sort_clause(cl)  /* sort clause for fold transformation */
struct clause *cl;
{
        if (cl == NULL) return(NULL);
        return(insert_clause(cl, sort_clause(cl->c_link)));
}

struct clause *insert_clause(ct, cl) /* insert ct into cl */
struct clause *ct,*cl;
{
        register struct clause *c,*cbefore;

        ct->c_link = NULL;
        if (cl == NULL) return(ct);
        if (greater_term(ct->c_form,cl->c_form)) { /* ct > top of cl? */
                ct->c_link = cl;
                return(ct);
        }
        for (c = cl; c != NULL; cbefore = c, c = c->c_link)
                if (greater_term(ct->c_form,c->c_form)) /* ct > c? */
                        break;
        cbefore->c_link = ct;
        ct->c_link = c;
        return(cl);
}

#define ARG_EQ 0
#define ARG_TRUE 1
#define ARG_FALSE 2

/*  term comparator */
int greater_term(t1,t2)         /* t1 > t2 ?? */
struct term *t1,*t2;
{
  register int i,cp;

  if (Pred(t1)->f_number != Pred(t2)->f_number)
    return(Pred(t1)->f_number > Pred(t2)->f_number);
  for(i = 0; i < Pred(t1)->f_arity; i++)
    if ((cp = greater_arg(Arg(t1,i),Arg(t2,i))) != ARG_EQ)
      return(cp == ARG_TRUE);
  return(FALSE);
}

/*
argument type:
0 variable (cmp_var)
1 complex term that has a variable (cmp_cplxt)
2 list that has a variable (cmp_list)
3 complex term without variable (cmp_cplxt)
4 list without variable (cmp_list)
```

```
5 atom, floating number (cmp_flt)
6 atom, integer number (cmp_int)
7 atom, string  (cmp_str)
8 atom, filepointer (cmp_fp)
9 clause (cmp_clause)
10 pst (cmp_pst)
*/
int arg_type(a)                 /* return argument type */
struct term *a;
{
  switch (a->type.ident) {
  case VAR_VOID_TYPE:
  case VAR_PST_TYPE:
  case VAR_GLOBAL_TYPE:
    return(0);
  case ATOMIC_TYPE:
    return(5 + a->t_arity);
  case LIST_TYPE:
    return(2);
  case CONST_LIST_TYPE:
    return(4);
  case CLAUSE_TYPE:
    return(9);
  case PST_TYPE:
    return(10);
  default:
    if (a->t_arity <= 0) return(3);
    return(1);
  }
}

int greater_arg(a1,a2)          /* a1 > a2 ? */
struct term *a1,*a2;
{
  int cp;
  int atype1 = arg_type(a1),atype2 = arg_type(a2);

  if ((cp = (atype1 - atype2)) != 0)
    if (cp > 0) return(ARG_TRUE);
    else return(ARG_FALSE);

  switch(atype1){
    case 0 : return(cmp_var(a1,a2));
    case 1 : return(cmp_cplxt(a1,a2));
    case 2 : return(cmp_list(a1,a2));
    case 3 : return(cmp_cplxt(a1,a2));
    case 4 : return(cmp_list(a1,a2));
    case 5 : return(cmp_flt(a1,a2));
    case 6 : return(cmp_int(a1,a2));
    case 7 : return(cmp_str(a1,a2));
    case 8 : return(cmp_fp(a1,a2));
    case 9 : return(cmp_clause(a1,a2));
    case 10: return(cmp_pst(a1,a2));
    }
    return(ARG_FALSE);
}

int cmp_var(a1,a2)              /* compare variable  a1 > a2?*/
struct term *a1,*a2;
{
        register int i;

        i = voccurrence(a1) - voccurrence(a2);
        if (i == 0) return(ARG_EQ);
        else if (i > 0) return(ARG_TRUE);
        else return(ARG_FALSE);
}

int cmp_cplxt(a1,a2)            /* compare complex terms */
struct term *a1,*a2;
{
        register int i,cp;

        if (Pred(a1)->f_number != Pred(a2)->f_number)
                return(Pred(a1)->f_number > Pred(a2)->f_number);
        for(i = 0; i < Pred(a1)->f_arity; i++)
                if ((cp = greater_arg(Arg(a1,i),Arg(a2,i))) != ARG_EQ)
                        return(cp);
        return(ARG_EQ);
}

int cmp_list(a1,a2)             /* compare list */
struct term *a1,*a2;
{
        int cp;

        if (a1 == NIL || a2 == NIL)
                if (a1 == a2) return(ARG_EQ);
                else if (a1 == NIL) return(ARG_TRUE);
                else return(ARG_FALSE);

        if (isvar(a1)) { /* patch 1991 03 03 */
          if (isvar(a2)) return(cmp_var(a1,a2));
          else return(ARG_FALSE);
        }

        if (cp = greater_arg(head_of_list(a1), head_of_list(a2)) != ARG_EQ)
                return(cp);
        else return(cmp_list(tail_of_list(a1),tail_of_list(a2)));
}

int cmp_clause(a1,a2)           /* compare clause */
struct term *a1,*a2;
{
        int cp;
        if (a1 == NULL || a2 == NULL)
                if (a1 == a2) return(ARG_EQ);
                else if (a1 == NULL) return(ARG_TRUE);
                else return(ARG_FALSE);
        if (cp = greater_arg(head_of_list(a1), head_of_list(a2)) != ARG_EQ)
```

```
                return(cp);
        else return(cmp_eclause(tail_of_list(a1),tail_of_list(a2)));
}

int cmp_flt(a1,a2)
struct term *a1,*a2;
{
        float cp;

        cp = num_value(a1) - num_value(a2);
        if (cp == 0) return(ARG_EQ);
        else if (cp > 0) return(ARG_TRUE);
        else return(ARG_FALSE);
}

int cmp_int(a1,a2)
struct term *a1,*a2;
{
        register int cp;
        cp = (int)num_value(a1) - (int)num_value(a2);
        if (cp == 0) return(ARG_EQ);
        else if (cp > 0) return(ARG_TRUE);
        else return(ARG_FALSE);
}

int cmp_str(a1,a2)
struct term *a1,*a2;
{
        register int cp;
        cp = strcmp(str_value(a1), str_value(a2));
        if (cp == 0) return(ARG_EQ);
        else if (cp > 0) return(ARG_TRUE);
        else return(ARG_FALSE);
}

int cmp_fp(a1,a2)
struct term *a1,*a2;
{
        register int cp;

        cp = filep_value(a1) - filep_value(a2);
        if (cp == 0) return(ARG_EQ);
        else if (cp > 0) return(ARG_TRUE);
        else return(ARG_FALSE);
}

int cmp_pst(a1,a2)
struct term *a1, *a2;
{
  register struct eclause *e1, *e2;
  int cp;

  e1 = ((struct pst *)a1)->p_lists;
  e2 = ((struct pst *)a2)->p_lists;
  while ((e1 != (struct eclause *)NULL) && (e2 != (struct eclause *)NULL)) {
    if ((cp = greater_arg(e1->e_form,e2->e_form)) != ARG_EQ)
      return(cp);
    e1 = e1->e_link;
    e2 = e2->e_link;
  }
  if (e1 == e2) return(ARG_EQ);
  else if (e2==(struct eclause *)NULL) return(ARG_TRUE);
  else return(ARG_FALSE);
}

/* c.t. step trace subroutine */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  int step_asking()

.     step_asking:
.       [apply]
.       nth_cset:
.       nth_literal:
.       quit_transformation:
.       reorder_clauses:
.       skip_cr:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int step_asking()               /* c.t. step trace --> 0(auto),1(cont) */
{
    tprint0("\n#step <h,b,q,x,u,c,n,CR>? ");
    while(1)
    {
        switch(getchar())
        {
          case '?' :
          case 'h' :
            tprint0("%%PATTERN: [<clause No>(<status>,<literal No.>)]");
            tprint0("New Predicate <=> unmodular body\n");
            tprint0("\t<STATUS> : U=untouched, M=modular_defined,");
            tprint0("I=unit_defined, D=Derived,\t\tG=registered,");
            tprint0("F=False, R=Reduced, T=Temporary\n");
            tprint0("COMMNAD: q:quit\tz:abort");
            tprint0("\tn:normal trace\tx:trace off");
            tprint0("\nb:break\tCR: continue");
            tprint0("\tu <clauseset No.> <literal No.>: manual unfolding\n");
            skip_cr();
            break;
          case 'b': {
            int  *usave = up;
            struct pair *esave = ep;
            struct ustack *usave = utop;
            struct cset *deflist_save  DEF_list, *cstrlist_save = CSTR_list,
                *initdeflist_save = INITDEF_list;
            struct eclause *constliterals_save = CONST_literals,
                *restliterals_save = REST_literals;
            int cstrnumber_save = CSTR_number;

            utop = usp;
            if (setjmp(unbreak_reset)) {
```

```
            utop = usave; hp = hsave; ep = esave;
            DEF_list = deflist_save; CSTR_list = cstrlist_save;
            INITDEF_list = initdeflist_save;
            CONST_literals = constliterals_save;
            REST_literals = restliterals_save;
            CSTR_number = cstrnumber_save;
            break;
          }
          while(1) {
            prolog_execution();
          }
        }
        case 'q' : quit_transformation(); /* quit */
          longjmp(trans_fail,1);
        case 'z' : abandon_transformation(); /* abort */
          longjmp(trans_fail,1);
        case 'u' : {int cnum,lnum; /* manual unfolding */
                   struct cset *te;
                   struct clause *tl;
                   scanf("%d %d",&cnum,&lnum);
                   skip_cr();
                   te = nth_cset(cnum);
                   if (te == NULL) {
                     tprintf("Error: no clause %d",cnum);
                     break;
                   }
                   tl = nth_literal(te->cs_clause->c_link,lnum);
                   if (tl == NULL)
                     {
                       tprint0("Error: literal out of range");
                       break;
                     }
                   te->cs_status = REMOVED;
                   if (te->cs_status != DERIVATION) /* in CSTR_list */
                         Pred(te->cs_clause->c_form)->f_setcount--;
                   reorder_clause(te->cs_clause, tl);
                   tprintf("manual_unfold [%d] ",lnum);
                   Plem(te->cs_clause->c_link->c_form,NULL);
                   if (apply(tl->c_form,te->cs_clause->c_form,
                             tl->c_link,te->cs_anumber)
                       == FALSE)
                     tprint0(" ->FAIL\n")
                   else
                     {tprint0(" ->TRUE\n")
                       return(1);
                     }
                 }
        case 'x' : CTnotrace; Notrace_mode;
          skip_cr();return(0); /* no trace */
        case 'n' : CTnormal; skip_cr();return(0);/* normal trace */
        case '\n': return(0); /* continue in autocode */
        default  : break;
      }
    }
}
```

```
struct cset *nth_cset(n)        /* cset whose cs_number = n */
int n;
{
        register struct cset *c;
        for (c = DEF_list; c != NULL; c = c->cs_link)
                if (c->cs_status == DERIVATION &&
                    c->cs_number == n) return(c);
        for (c = CSTR_list; c != NULL; c = c->cs_link)
                if (c->cs_status != REMOVED &&
                    c->cs_number == n) return(c);
        return(NULL);
}

struct clause *nth_literal(cl,n)
struct clause *cl;
int n;
{
        register struct clause *c;
        register int i;

        for (c = cl, i = 1; c != NULL; c = c->c_link, i++)
                if (i == n) return(c);
        return(NULL);
}

void skip_cr()                  /* skip user's input "...CR" */
{
        while( getchar() != '\n') ;
}

void show_newdefs()             /* show newly defined clauses */
{
        register struct cset *cs;
        register struct func *f;
        for (cs = DEF_list; cs != NULL; cs = cs->cs_link)
        {
                f = Pred(cs->cs_clause->c_form);
                if isnotreduced(f)
                        Showfunc(f);
        }
}
```

```
/*-----------------------------------------------------------------------
*               cu Prolog III (Constraint Unification Prolog)
*   Copyright: Institute for New Generation Computer Technology,Japan
*                          1989--91
-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------
*               << tr_split.c >>
*         divide constraint into equivalence classes
-----------------------------------------------------------------------*/

#include "include.h"
#define DEBUG 0
extern struct cset *DEF_list;
extern struct cset *CSTR_list;
extern struct cset *INITDEF_list;
extern int CSTR_number;
extern struct clause *CONST_literals, *REST_literals;
extern struct itrace *newisave;
extern jmp_buf trans_fail;
extern jmp_buf split_fail;

/* sub functions for transform.c (divide clause into equivalence classes)*/
/* this module uses global vars: CONST literals, REST literals
*
.   split
.     vconstraint
.     attach
.       is_modular_literal
.       attach_arg
.         novar
.         replace terms
.       has_no_var
.         novar
.     divide_consts
.       has_no_pst
.     delete_constraint
.     remove_modular_literals
.       is_modular_literal
*/

/* ---------- begin 'split' ---------- */
struct clause *CONST_PST_literals; /* used in split, divide_consts */

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   split(clist,vlist,anum)
  split a clause clist into equivalence class
     vlist = variable list of clist
     anum = # of variables + # of PSTs in clist
  global vars:
     CONST_literals : no vars and no psts
     REST_literal : modular literals
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
struct compartment *split(clist, vlist, anum)
struct clause *clist;
struct term *vlist;
int anum;
{
        register struct term *v;
        register struct compartment *cmp,*cmplast;
        struct clause *cnext,*remove_modular_literals();
        void divide_consts();

        CONST_literals = NULL; /* global vars */
        REST_literals = NULL;
        CONST_PST_literals = NULL;

        clist = remove_modular_literals(clist);
        if (clist == NULL) return(NULL);
        else if (vlist == NULL) /* no variable */
                divide_consts(clist); /* CONST or CONST_PST? */
        else
        {
                /* clear_vconstraint(vlist) */
                for (v = vlist; v != NULL; v = vlink(v))
                        vconstraint(v) = NULL;
                attach(clist, vlist,anum);
                delete_constraint(vlist);
        }

        for (cmplast = NULL; CONST_PST_literals != NULL;
             CONST_PST_literals = cnext)
        {
                cnext = CONST_PST_literals->c_link;
                CONST_PST_literals->c_link = NULL;
                MEMORY_ALLOC(cmp,compartment,TEMPORAL);
                cmp->cmp_clause = CONST_PST_literals;
                cmp->cmp_link = cmplast;
                cmplast = cmp;
        }
        for (v = vlist; v != NULL; v = vlink(v))
                if (vconstraint(v) != NULL){
                        MEMORY_ALLOC(cmp,compartment,TEMPORAL);
                        cmp->cmp_clause = vconstraint(v);
                        cmp->cmp_link = cmplast;
                        cmplast = cmp;
                }
        return(cmplast);
}

/* remove modular literals in cl into REST_literals */
struct clause *remove_modular_literals(cl)
struct clause *cl;
{
        register struct clause *cnext;

        if (cl == NULL) return(NULL);
        if (is_modular_literal(cl->c_term))
        {
                cnext = cl->c_link;
                cl->c_link = REST_literals;
```

```
                REST_literals = cl;
                return(remove_modular_literals(cnext));
        }
        else
        {
                cnext = cl->c_link;
                cl->c_link = remove_modular_literals(cnext);
                return(cl);
        }
}

void delete_constraint(vl)      /* vconstraint(v)=NULL for all vl */
struct term *vl;
{
        register struct term *vl, *v2;
        register struct clause *c;

        for (vl = vl; vl != NULL; vl = vlink(vl))
        {
          if (vconstraint(vl) == NULL) continue;
          c = vconstraint(vl);
          for (v2 = vlink(vl); v2 != NULL; v2 = vlink(v2))
            if (vconstraint(v2) == c) vconstraint(v2) = NULL;
        }
}


int has_no_pst(t)               /* check pst in arguments */
struct term *t;
{
        register int i;

        for (i = 0; i < Pred(t)->f_arity; i++)
                if (is_pst(Arg(t,i))) return(FALSE);
        return(TRUE);
}

void divide_consts(cl)          /* cl: constraint clauses */
struct clause *cl;
{
        register struct clause *c,*cnext;
        for (c = cl; c != NULL; c = cnext)
        {
                cnext = c->c_link;
                c->c_link = NULL;
                if (has_no_pst(c->c_form))
                {
                        c->c_link = CONST_literals;
                        CONST_literals = c;
                }
                else
                {
                        c->c_link = CONST_PST_literals;
                        CONST_PST_literals = c;
                }
        }
}

int Attached;                   /* used in attach, attach_arg */

void attach(c, vl,anum) /* split main */
register struct clause *c;
struct term *vl;
int anum;
{
        struct clause *cnext;
        register struct term *t;
        register int i, j;

        while (c != NULL) {
          cnext = c->c_link;
          c->c_link = NULL;

          t = c->c_form; j = Pred(t)->f_mark;
          if (has_no_var(t))
          {
                  c->c_link = CONST_literals;
                  CONST_literals = c;
          }
          else if ((j & NON_UNFOLDABLE) != 0) {
            i = satisfiable(c,anum);
            j &= STAY_IF;
            if (((i == TRUE) && (j != 0)) ||
                ((i == FALSE) && (j == 0))) {
              j = Pred(t)->f_arity;
              for (i = 0; i < j; i++)
                attach_arg(Arg(t,i),c,vl);
            }
            else if ((i == TRUE) && (j == 0))
              /* do nothing */ ;
            else /* in case of FAIL */
              longjmp(split_fail,i); /* -> modular_form() */
          }
          else if (is_modular_literal(t)) {
            c->c_link = REST_literals;
            REST_literals = c;
          }
          else {
            /* attach_term(c,vl); */
            Attached = 0;
            j = Pred(t)->f_arity;
            for (i = 0; i < j; i++)
              attach_arg(Arg(t,i),c,vl);
            if (Attached == 0) divide_consts(c);
          }
          c = cnext;
        }
}
```

```
void attach_arg(arg,c,vl)       /* sub of attach */
struct term *arg,*vl;
struct clause *c;
{
  register int i;

  if (novar(arg)) return;

/*  printf("attach_arg arg=");Pterm(arg,NULL);
  printf(" c=");Pclause(c,NULL);
  NL;
*/
  if (isvar(arg)) {
    if (arg->type.ident == VAR_GLOBAL_TYPE) {  /* 1991 03 02 */
      if (vconstraint(arg) == NULL) vconstraint(arg) = c;
      else replace_terms(vconstraint(arg),c,vl);
      Attached = 1;                 /* global var */
      return;
    }
    else return;
  }

  if (is_list(arg) || is_clause(arg)) {
    attach_arg(head_of_list(arg),c,vl);
    attach_arg(tail_of_list(arg),c,vl);
    return;
  }
  if (is_pst(arg)) {
    struct eclause *plists;
    plists = ((struct pst *)arg)->p_lists;
    while (plists != (struct eclause *)NULL) {
      attach_arg(Arg2(plists->e_form),c,vl);
      plists = plists->e_link;
    }
    return;
  }
  for (i = 0; i < Pred(arg)->f_arity; i++)
    attach_arg(Arg(arg,i),c,vl);
}


void replace_terms(c1,c2,vl)
struct clause *c1;
register struct clause *c2;
struct term *vl;
{
  register struct term *v;

  if (c1 == c2) return;
  for (v = vl; v != NULL; v = vlink(v))
    if (vconstraint(v) == c1) vconstraint(v) = c2;
  while (c2->c_link != NULL)
    c2 = c2->c_link;
  c2->c_link = c1;
}
```

```
/*        - - - - - end of 'split'  - - -  - - - - - - - - - */


/*    - -           begin 'variant' - - - - - - - - - - - - - */
struct vpair *VPAIR,            /* vpair of var.pst */
        *TermPAIR;              /* vpair of others */
int Consider_Vacuous,           /* consider vacuous or not */
        PST_level;              /* depth of pst */

void Pvariant(va)
struct variant *va;
{
        struct vpair *vp;
        Pclause(va->v_clause,NULL);
        if (va->v_pair == NULL) return;
        putchar('\n');
        for (vp=va->v_pair; vp != NULL; vp=vp->v_link)
        {
                printf("<");
                Pterm(vp->v1,NULL);putchar(',');
                Pterm(vp->v2,NULL);putchar('>');
        }
}

void Pvpair(va)
struct vpair *va;
{
        register struct vpair *vp;
        for (vp=va; vp != NULL; vp=vp->v_link)
        {
                printf("<");
                Pterm(vp->v1,NULL);putchar(':');
                Pterm(vp->v2,NULL);printf("> ");
        }
}

int vpair_length(vp)
struct vpair *vp;
{
        register int i;
        for(i = 0; vp != NULL; i++, vp=vp->v_link);
        return(i);
}

/*
[variant] - - - - - - - - - - - - - - - - - - - - - - - - - - - -
.   copy_clause
.   .   copy_term
.   .   .   NpstObj~
.   .   .   in_sheap~
.   .   .   copy_arg*
.   .   .   .   has_common_label
.   .   .   .   store_vpair
.   .   .   exist_termpair
```

```
.   .   .   exist vpair
.   .   .   copy pst1
.   .   .   .   Npst
.   .   .   .   store vpair~
.   .   .   .   copy eclause1
.   .   .   store tempair
.   .   .   var trans1
.   .   .   .   store vpair~
*/
/*****************************************************************
  variant(cl,flag)      : normal copy
  variant_v(cl,flag)    : consider vacuous arguments
  make a variant of a clause cl
*****************************************************************/
struct variant *variant(cl,flag)
struct clause *cl;
int flag;
{
        struct variant *ccopy;

        VPAIR = TermPAIR = NULL;                /* global vars */
        v_list = pv_list = NULL;
        v_number = p_number = 0;
        PST_level = 0;
        Consider_Vacuous = 0;
        MEMORY_ALLOC(ccopy,variant,TEMPORAL);
        ccopy->v_clause = copy_clause(cl,flag);
        ccopy->v_var = v_list;
        ccopy->v_anum = v_number + p_number;
        ccopy->v_pair = VPAIR;
        if (p_number != 0)
                renum_pvars((struct pstvar *)pv_list,v_number);
        return(ccopy);
}

struct variant *variant_v(cl,flag) /* variant considering vacuous args */
struct clause *cl;
int flag;
{
        struct variant *ccopy;

        VPAIR = TermPAIR = NULL;                /* global vars */
        v_list = pv_list = NULL;
        v_number = p_number = 0;
        PST_level = 0;
        Consider_Vacuous = 1;
        MEMORY_ALLOC(ccopy,variant,TEMPORAL);
        ccopy->v_clause = copy_clause(cl,flag);
        ccopy->v_var = v_list;
        ccopy->v_anum = v_number + p_number;
        ccopy->v_pair = VPAIR;
        if (p_number != 0)
                renum_pvars((struct pstvar *)pv_list,v_number);
        return(ccopy);
}

void store_vpair(told,tnew)     /* store told->tnew (var,pst)in VPAIR */
struct term *told,*tnew;
{
        struct vpair *vp;

        MEMORY_ALLOC(vp,vpair,TEMPORAL);
        vp->vl = told; vp->v2 = tnew;
        vp->v_link = VPAIR;
        VPAIR = vp;
}

struct term *exist_vpair(t)     /* check if t is in VPAIR */
struct term *t;
{
        register struct vpair *vp;
        for (vp = VPAIR; vp != NULL; vp = vp->v_link)
                if (vp->vl == t) return(vp->v2);
        return(NULL);
}

void store_tempair(told,tnew)  /* store told->tnew (term) to TermPAIR */
struct term *told,*tnew;
{
        struct vpair *vp;

        MEMORY_ALLOC(vp,vpair,TEMPORAL);
        vp->vl = told; vp->v2 = tnew;
        vp->v_link = TermPAIR;
        TermPAIR = vp;
}

struct term *exist_tempair(t)  /* check if t is in TermPAIR */
struct term *t;
{
        register struct vpair *vp;
        for (vp = TermPAIR; vp != NULL; vp = vp->v_link)
                if (vp->vl == t) return(vp->v2);
        return(NULL);
}

struct clause *copy_clause(cl,flag) /* sub. of variant */
struct clause *cl;
int flag;
{
        struct term *copy_term();

        if (cl == NULL) return(NULL);
        return(Nclause(copy_term(cl->c_form,flag),
                       copy_clause(cl->c_link,flag),
                       flag)
               );
}

struct term *copy_term(t,flag)  /* variant of t */
```

```
struct term *t;
int flag;
{
  struct term *nt, *copy_pst(), *copy_arg();
  register int i;

  if (t == NULL) return(t);
  if ((nt = exist_vpair(t)) != NPLL) return(nt); /* check history */
  if (isvar(t)) return(var_trans(t,flag)); /* t is var */
  if (is_pst(t)) {
        PST_level++;
        nt = copy_pst((struct pst *)t.flag); /* t is PST */
        PST_level--;
        return(nt);
  }
  if ((nt = exist_tempair(t)) != NULL) return(nt); /* check history */
  switch(t->type.ident) {       /* atom, complex term, eclause */
  case ATOMIC_TYPE: /* constant term */
        return(t);
  case CONST_LIST_TYPE:
        if (in_sheap(t) || flag != ETERNAL) return(t);
  case LIST_TYPE:
    nt = (struct term *)Nlist(copy_term(head_of_list(t),flag),
                (struct clause *)copy_term(tail_of_list(t),flag), flag);
    break;
  case CLAUSE_TYPE:
    nt = (struct term *)Nclause(copy_term(head_of_list(t),flag),
                (struct clause *)copy_term(tail_of_list(t),flag),flag);
    break;
  case ECLAUSE_TYPE: /* pst object */
    nt = (struct term *)Npstobj(copy_term(((struct eclause *)t)->e_form,flag),
                (struct pair *)NULL,
                (struct eclause *)
                  copy_term((struct term *)((struct eclause *)t)->e_link,
                        flag),
                flag),
    break;
  case VAR_VOID_TYPE:
    return(Anonymous_var);
  case VAR_PST_TYPE:
    error("System error occurs at 'copy_term'");
  default:
    if (isconst_functor(t))
          nt = up_const_functor(t,flag);
    else {
          nt = Nterm(t->t_arity,flag);
          Pred(nt) = Pred(t);
          if (Consider_Vacuous == 1)
                for (i = 0; i < t->t_arity; i++)
                      Arg(nt,i) = copy_arg(t,i,flag);
          else
                for (i = 0; i < t->t_arity; i++)
                      Arg(nt,i) = copy_term(Arg(t,i),flag);
    }
    store_tempair(t,nt);
    return(nt);
}

struct term *copy_pst(pt,flag)  /* copy pst */
struct pst *pt;
int flag;
{
  register struct pst *p;
  register struct term *nt;
  struct eclause *copy_eclause();

  nt = exist_tempair((struct term *)pt);
  if (nt != NPLL)
  {
        if (PST_level == 1) store_vpair((struct term *)pt,nt);
        return(nt);
  }
  p = Npst(flag);
  ((struct pstvar *)(p->p_var))->old_var = pt->p_var;
  p->p_lists=(struct eclause *)copy_term((struct term *)pt->p_lists,flag);
  if (PST_level == 1)           /* top level PST */
        store_vpair((struct term *)pt,(struct term *)p);
  else
        store_tempair((struct term *)pt,(struct term *)p);
  return((struct term *)p);
}


struct term *var_trans(v, flag) /* var that corresponds to v */
struct term *v;
int flag;
{
        register struct term *nv;
        sprintf(nbuf,"V%d",v_number);
        nv = Nvar(nbuf,flag);
        voccurrence(nv) = voccurrence(v);
        vheadoccurrence(nv) = vheadoccurrence(v);
        store_vpair(v,nv);
        return(nv);
}


struct term *copy_arg(t,i,flag) /* copy ith argument of t with vacuity check */
struct term *t;
int i,flag;
{
        register struct term *arg,*nt;

        arg = Arg(t,i);
        if ((nt = exist_vpair(arg)) != NULL) return(nt); /* check history */
        if ((nt = exist_tempair(arg)) != NULL) return(nt); /* check history */
        if (is_pst(arg) && (PST_level == 1) &&
            !has_common_label(
```

```
                    ((struct pst *)arg) )p_lists.Component(Pred(i),i)))
        [
                struct term *nv;
                sprintf(nbuf,"P%d",v_number);
                nv = Nvar(nbuf,flag);
                store_vpair(arg,nv);
                return(nv);
        ]
        else return(copy_term(arg,flag));
]

/* ---------- end of 'variant' ----------          */
```

```
# Makefile for cup

OBJECTS = main.o jpsgsub.o mainsub.o modular.o new.o print.o read.o\
          refute.o unify.o defsysp.o syspredl.o\
          syspred2.o trans.o tr_gub.o tr_split.o
CFLAGS = -g
CC = cc
FLAGS = -lm

cup: $(OBJECTS)
        $(CC) $(CFLAGS)  -o cup3 $(OBJECTS) $(FLAGS)

$(OBJECTS): include.h
```

```
%%%%%%%%%%%%%%%%%%%%%%%%% jp.pro%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%              JPSG parser ver1.0
%%%                      1991.5.29 (by H.Tsuda)
%%%     jp.pro           parser program
%%%     jconst.pro       general constraints
%%%     post.pro         pp,aux,etc. dictionary
%%%     suffix.pro       suffix dictionary
%%%     noun.dic         proper noun, common noun
%%%     verb.dic         verbs dictionary
%%%     sahen.dic        sahen-verb,noun dictionary
%%%     adject.dic       adjective, adjective-verb dictionary
%%%     etc.dic          other dictionary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%  Category :
%%%      [core/[pos/Pos, form/Form, view/View],
%%%        ajc/Adjacent, sc/Subcat, ajn/Adjoin,
%%%        psl/PSlash, slash/Slash, refl/Refl, sem/Sem (,temp/Temp)]
%%% Pos: part of speech   p,n,v,vs(sahen-dousi),adn(rentai-si),adv(fuku si)
%%% Form: when Pos=n, n,ns(sahen meisi)
%%%       when Pos=v, vv,vk,vsw,... (verb form),
%%%                   adj(adjective), na(adjective-verb)
%%% View: asp(a_begin,a_end,a_interval,a_type)
%%% Temp: Temp feature for proto-lexicon      t(s,f,r,def,drf)

%%%  left Corner Parser
p(Sentence):-
        parse0(Cat,H,Sentence,[],[idx(s,speaker)]),nl,
        tree(H),nl,
        write("category= "),write(Cat),nl,
        write("constraint = "),pcon,nl,
%%      ;Cat=[refl/TopRefl],nil_or_speaker(TopRefl).

nil_or_speaker([]).
nil_or_speaker([[sem/speaker]]).

parse0(MCat,MHist,Str,Rest,Idxlist):-
        lookup(Str,SubStr,Cat,Hist,Idxlist,Nidx),!,
        parse1(Cat,Hist,MCat,MHist,SubStr,Rest,Nidx).

parse1(Cat,H,Cat,H,Str,Str,N):-!.

parse1(LCat,LHist,GCat,GHist,[Word|SubStr],Rest,Idxlist):-
        lookup_post(LCat,Word,RCat,RHist,RuleName),
        psr_adj(LCat,RCat,MCat),
        !,
        parse1(MCat,
                t(t(MCat,RuleName,[]),LHist,RHist),
                GCat,GHist,SubStr,Rest,Idxlist).

parse1(LCat,LHist,GCat,GHist,Str,Rest,Idxlist):-
        psr(LCat,RCat,MCat,RN),
        parse0(RCat,RHist,Str,SubStr,Idxlist),
        parse1(MCat,t(t(MCat,RN,[]),LHist,RHist),
                GCat,GHist,SubStr,Rest,Idxlist).

%%%  phrase structure rules
%%%      psr(LeftCat,RightCat,MotherCat)

%%% 1. Adjacent Structure:  psr_adj(Left,Head,Mother)
psr_adj([core/Hc,sc/Csc,refl/Cref,slash/Csl,psl/Cpsl,sem/SEM0,ajn/[]],
    [core/Hc,ajc/[core/Cc,sc/Asc,refl/Refl,NC,sem/SEM0]],
      ajn/Adj, sc/Hsc, refl/Href, slash/Hsl, sem/SEM],
    [core/Hc,ajc/[],ajn/Adj,sc/Msc,refl/Mref,slash/Msl,psl/Cpsl,
        sem/SEM]),
        adjacent_sc_p(Csc,Asc,Hsc,Msc),
        slash_p(Csl,Hsl,Msl),
        refl_cond(Cref,Href,Mref,Hsc).

%%% slash feature principle:
%%%      slash_p(LeftS,RightS,MotherS)
%%%      LeftS:C.slash, RightS:H.slash, MotherS:M.slash
slash_p([],[],[]).
slash_p([S],[],[S]).
slash_p([],[S],[S]).
slash_p([S],[RS],[RS]):-sem_unify(S,RS).
sem_unify([sem/X],[sem/X]).

%%  adjacent-subcat principle for adjacent structure
%% adjacent_sc_p(CSC,ASC,HSC,MSC):
%%      CSC=C.sc, ASC=H.ajc.sc, HSC=H.sc, MSC=M.sc
%% adjacent_sc_p(CSC,[],HSC,MSC):-merge(CSC,HSC,MSC).
adjacent_sc_p([],[],SC,SC).
adjacent_sc_p([SC|Rj,[],[],[SC|R]).
adjacent_sc_p(CSC,[AS],HSC,SC):-one_of(CSC,AS,Rest),append(HSC,Rest,SC).
adjacent_sc_p([A1,A2],[A1,A2],SC,SC).  /* for ukemi */

%%%      2. relative clause structure : psr(R,H,M)
psr([core/[form/rel],sc/Rsc,slash/Rsl,psl/Rps,sem/Rs,ajc/[],ajn/[]],
    [core/Hc,sc/Ha,slash/Hsl,sem/Hs],
    [core/Mc,sc/[],slash/Msl,psl/[],sem/[Hs,Rs]],[relative_s]):-
        Hc=[pos/n],sc_sl(Rsc,Rsl,Rps,Ha,Hsl,Msl,Hs).

sc_sl([],[[sem/Hsm]],Rps,[],Msl,Hsm):-slash_p(Rps,Hsl,Msl).
sc_sl([],Rsl,Rps,[[core/[form/rel]]],Hsl,Msl,Hsm):-sl_psl_p(Rsl,Hsl,Msl,Rps).
sc_sl([[sem/Hsm]],[Rs]],Rps,[],Hsl,Msl,Hsm):-sl_psl_p([Rs]],Hsl,Msl,Rps).

%% sl_psl_p(Csl,Hsl,Msl,Psl)
sl_psl_p([],[],[P],[P]).
sl_psl_p(Csl,Hsl,Msl,[]):-slash_p(Csl,Hsl,Msl).
sl_psl_p(Csl,Hsl,Msl,[[sem/X]]):-slash_p(Csl,Hsl,[[sem/X]]).

%%%      3. Subcategorization Structure : psr(C,H,M)
psr(Comp,
    [core/Hc, ajn/Hn, ajc/Hac, sc/HC, refl/Hr, slash/Hsl, sem/Hs],
    [core/Mc, ajn/Mn, ajc/Mac, sc/Rest, refl/Mr, slash/Msl,psl/Cps,sem/Ms],
        [subcat_p] ):-
        Comp=[core/Cc, ajc/[], refl/Cr,slash/Csl,psl/Cps, ajn/[]],
        one_of(HC,Comp,Rest),
        slash_p(Csl,Hsl,Msl),
```

```
        refl_cond(Ct,Ah,Mr,Rest),
        se_cond(Ce,He).

se_cond([pos/p],[pos/v]).
se_cond([pos/adn],[pos/n]).

refl_cond([],[],[],Rt).
refl_cond([],[Cat],[Cat],Rt).
refl_cond([Cat],[],[Cat],Rt).
refl_cond([Cat],[],[],SC) :- member(Cat,SC).

%%%     4. Adjunction Structure: psr(An,H,M)
psr([core/Ac,ajn/[Head], slash/Asl,refl/RefIA,psr/Apsl,sem/Asl],
    Head,
    [core/C,ajn/A,aje/Ma,sc/Bsc,slash/Msl,refl/RefIM,psr/Apsr,sem/Asl],
        [adjunct_pl] );
        Head=[core/C,ajn/A,aje/Ma,sc/Bsc,slash/Hsl,refl/RefIH,sem/Hs],
        refl_cond(RefIA,RefIH,RefIM,Bsc),
        slash_p_ajn(Asl,Hsl,Msl,[[core/Ac,sem/As]],[[core/C,sem/Hs]]).

%%slash_p_ajn(Asl,Hsl,Msl,Ajn,Head)
slash_p_aju([Sl],Y,Y,T,[St]).
slash_p_aju(F,[Sl],Y,[Sl],Y).
slash_p_aju(Asl,Hsl,Msl,Fl,F2):-slash_p(Asl,Hsl,Msl).

%%%%%   lexical rule (for general dictionary)
%%% lex_rule(OrigCat,NewCat)
lex_rule(C,Cat):-
        same_feature(C,Cat),
        aje_default(C,Cat),ajn_default(C,Cat),refl_default(C,Cat),
        se_to_sl(C,Cat),refl_default(C,Cat),temp_to_asp(C,Cat).

%%%%%%% use default() predicate!
%% core,sem
same_feature([core/C,sem/S],[core/C,sem/S]).

%% adjacent default(defaul value is [], if it is not specified)
aje_default([aje/[]],[aje/[]]) :- !.
aje_default([aje/X],[aje/X]).

%% adjoin default(defaul value is [], if it is not specified)
ajn_default([ajn/[]],[ajn/[]]) :- !.
ajn_default([ajn/X],[ajn/X]).

%% refl default(defaul value is [], if it is not specified)
refl_default([refl/[]],[refl/[]]) :- !.
refl_default([refl/X],[refl/X]).

%% se,slash (subcat to slash movement, se,slash,psl default)
se_to_sl([sc/[]],[sc/[]],slash/[],psl/[]) :- !.
se_to_sl([sc/S],[sc/Nsc],slash/Nsl,psl/[]):-se_sl_move(S,Nsc,Nsl).
se_sl_move(Se,Se,[]).
se_sl_move(Se,Nse,[[sem/S]]) :- one_of(Se,[sem/S],Nse).

%% temp,asp (temp to aspect conversion, asp default)

temp_to_asp([temp/[]],[core/[view/[]]]) :- !.
temp_to_asp([temp/I(S,F,R,DSF,DEF)],[core/[view/asp(AB,AE,AD,AT)]]):-
        temp_estr(S,F,R,DSF,DEF,AB,AE,AD,AT).
temp_estr(S,F,TO,DSF,T1,S,F,DSF,basic).
temp_estr(TO,F,R,T1,f,F,R,f,result).
temp_estr(TO,F,T1,T2,T),F,i,i,exp).
temp_estr(TO,T1,R,T2,f,R,i,i,exp).

%%%%%   general dictionary
%%%%%  lookup(Str,RestStr,Cat,Hist,OldIdx,NewIdx)
%%%%%  'lookup'[nil'X],X, , ):-fail,!.
%%lookup([Idx([Sent|Cont],I)|X],X,Cat,Hist):-
%%        parse0(Cat,Hist,[Sent|Cont],[]).
lookup([Idx(Word,I)|X],X,Cat,t(Cat,[Word,I],[]),OldIdx,OldIdx)
        :- member(idx(I,Sem),OldIdx),!,
        dict([Word,C),lex_rule(C,Cat),Cat=[sem/Sem].
lookup([Idx(Word,I)|X],X,Cat,t(Cat,[Word,I],[]),OldIdx,[Idx(I,Sem)|OldIdx])
        :- !,dict([Word,C),lex_rule(C,Cat),C=[sem/Sem].
lookup([[Sent|Cont]|X],X,Cat,Hist,Idx,Idx)
        :-parse0(Cat,Hist,[Sent|Cont],[],Idx).
lookup([Word|X],X,Cat,t(Cat,[Word],[]),Idx,Idx)
        :- dict([Word,C),lex_rule(C,Cat).

%%%%%   pp, suffix, auxiliary verb dictionary
%%%%%   lookup_post(PreCat, Word, PostCat, PostHist,RuleName)
lookup_post([core/[pos/v,form/Form]],Word, Cat,t(Cat,[Word],[]),[suff_pl])
        :- search_suffix(Form,Word,Cat);
        Cat=[slash/[],psl/[],refl/[]].
lookup_post(_,Word,Cat,t(Cat,[Word],[]),[adjacent_pl])
        :-dict_pos(Word,Cat);
        Cat=[slash/[],psl/[],refl/[]].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%        jconst.pro
%%%%%%%%%  constraints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%     general constraint          %%%
%%%  constraint  (finite predicate)

%%% one_of(List, One, Rest)     !List!<-3
one_of([X|Y],X,Y).
one_of([X,Y|Z],Y,[X|Z]).
one_of([X,Y,Z],Z,[X,Y]).

%%% member(X,List)       !List <-3
member1(X,[X|Y]).
member1(X,[Y,X|Z]).
member1(X,[Y,Z,X]).

%%% Write permutation
perm2([A,B],[A,B]).
perm2([A,B],[B,A]).
perm3([A,B,C],[A|X]): perm2([B,C],X).
perm3([A,B,C],[B|X]):-perm2([A,C],X).
```

```
perm3([A,B,C],[C|X]): perm2([A,B],X).

%%% tel_clause(Form,HSC,CSC,HAdjoin,Csem,Hsem)
syu_ten(inf).
syu_ten(rel).
adj_syu_ten(a_inf).
adj_syu_ten(rel).

%%%%%%%  constraint (recursive predicate)
merge([],[],[]).
merge([],[A|X],[A|X]).
merge([A|X],[],[A,X]).
merge([A|X],[B|Y],[A|Z]): merge(X,[B|Y],Z).
merge([B|X],[A|Y],[A|Z]): merge([B|X],Y,Z).

append([],X,X).
append([A|X],Y,[A|Z]): append(X,Y,Z).

member(X,[X|Y]).
member(X,[Y,Z]): member(X,Z).

%%%%%%%%%%%%% saiun constraint (need debug) %%%%%%%%%%%%%%%%%%
su_handler(Adjacent,v_su,[],SC,Sem) :- suru(Adjacent,SC,Sem).
su_handler([[core/[pos/v,form/vsc],ajc/[],ajn/[],sc/[],sem/Sem]],
                Form,[],[],Sem).

suru([[core/[pos/n,form/ns],sc/Subc,sem/[Pred|SRest]]],SC,[Pred|SRest])
        :- suru_correspond(SRest,Subc,SC).
suru([],[[core/[pos/p,form/wo],
                sc/[[core/[pos/adn,form/qa],refl/RF,sem/Sbj]|Sc],
        refl/RF,sem/[Pred,Sbj,Obj|SRest]]],SC),[Pred,Sbj,Obj|SRest])
        :-suru_correspond([Sbj|SRest],
        [[core/[pos/adn,form/qa],refl/RF,sem/Sbj]|Sc],SC).
suru([],[[core/[pos/p,form/qa],refl/RF,sem/Sbj],
         [core/[pos/p,form/wo],
                sc/[[core/[pos/adn,form/qa],refl/RF,sem/Sbj]'SC],
                sem/[Pred,Sbj,Obj|SRest]]'SC],
     RR,RF,[Pred,Sbj,Obj|SRest]) :- suru_correspond(SRest,Sc,SC).

suru_correspond([],[],[]).
suru_correspond([Sbj|_],[[core/[form/F],sem/Sbj]]],
        [[core/[pos/p,form/F],sem/Sbj]]).
suru_correspond([Sbj,Obj],
        [[core/[form/FO],sem/Obj],[core/[form/FS],sem/Sbj]],
        [[core/[pos/p,form/FS],sem/Sbj],[core/[pos/p,form/FO],sem/Obj]]).
suru_correspond([Sbj,Obj,Iob],
        [[core/[form/FO],sem/Obj],[core/[form/FI],sem/Iob],
                [core/[form/FS],sem/Sbj]],
        [[core/[pos/p,form/FS],sem/Sbj],[core/[pos/p,form/FI],sem/Iob],
                [core/[pos/p,form/FO],sem/Obj]]).

adn_2([[core/[pos/adn,form/First],sem/Obj],
        [core/[pos/adn,form/Second],sem/Sbj]],
        First,Second,Obj,Sbj).
adn_2([[core/[pos/adn,form/First],sem/Obj],
        [core/[pos/adn,form/Second],sem/Sbj]],
        Second,First,Obj,Sbj).

adn_wo_ga(SC,Obj,Sbj):-adn_2(SC,wo,ga,Obj,Sbj).

adn_1([[core/[pos/adn,form/First],sem/Sem]],Form,Sem).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%              post.pro
%%%     PP, aux , etc.   (have MUJENT feature)
%%%     dict_pos( [word, Cat)            : pp, aux, etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%    Post Positions
dict_pos(nado,
        [core/[pos/n,form/Form],ajn/[],sc/[],
        ajc/[[core/[pos/n,form/Form],sc/[],sem/SEM]],sem/etc(SEM)] ).

%%%%%   Postpositional particles (joshi)        %%%%%%%%%%%%%%%%%%%%
%%%  --wo,ga,ni,de,to,no,ha,ba
general_pp(Form,
        [core/[pos/p,form/Form],ajn/[],sc/[],
        ajc/[[core/[pos/n],sc/[],sem/SEM]],sem/SEM] ).
dict_pos(wo, Cat):-general_pp(wo, Cat).
dict_pos(ga, Cat):-general_pp(ga, Cat).
dict_pos(ni, Cat):-general_pp(ni, Cat).

dict_pos(de,
        [core/[pos/adn,form/de],sc/[],
        ajc/[[core/[pos/n],sc/[],sem/SEM1]],
        ajn/[[core/[pos/v],sem/SEM2]],sem/de(SEM1,SEM2)] ).

dict_pos(to,
        [core/[pos/Pos,form/to],sc/[],
        ajc/[[core/[pos/Cat1,form/Form],sc/[],sem/SEM1]],
        ajn/[[core/[pos/Cat2],sem/SEM2]], sem/SEM] ):
        to_compl(Pos,Cat1,Form,SEM1,Cat2,SEM2,SEM).

to_compl(adj,n,_,S1,v,S2,[with(S1)|S2]).
to_compl(adn,n,_,S1,n,S2,[S1,and S2]).
to_compl(adj,v,Inf,S1,v,S2, [with,S1,S2] ):-sentence_end(Inf).
sentence_end(inf).
sentence_end(a_inf).
to_compl(adj,v,imp,S1,nil,S2,S1).

dict_pos(no,
       [core/[pos/adn,form/Form],sc/[],
        ajc/[[core/[pos/CP,form/CF],sc/[],sem/CS]],ajn/Adjoin,sem/SEM] );
        no_handler(CP,CF,Form,Adjoin,CS,SEM).

no_handler(p,Form,Form,[],Sem,Sem):-
    with_case(Form).
no_handler(n,n,no,
        [[core/[pos/n,form/nl],ajc/[],ajn/[],sc/[],sem/inst(Obj,Type)]],
          Atr,inst(Obj,[rel,Atr,Type])).
```

```
%%no_handler(n,n,Form,[],Sem,Sem):
%%    without_case(Form).

dict_pos(ha,
        [core/[pos/p,form/Form],ajn/[],sc/[],
         aje/[[core/[pos/Cat,form/F],sc/[],sem/SEM]],sem/SEM] );
        wa_compl[Cat,F,Form].

wa_compl(p,Form,Form): with_case(Form).
wa_compl(n,_,Form) :  without_case(Form).

with_case(to).
with_case(he).
with_case(ni).
with_case(no).

without_case(ga).
without_case(wo).

dict_pos(no,
        [core/[pos/p,form/Form],ajn/[],sc/[],
         aje/[[core/[pos/Cat,form/F],sc/[],sem/SEM]],sem/SEM] );
                no_compl(Cat,F,Form).

no_compl(p,wo,wo).
no_compl(n,F,ga).
no_compl(p,Form,Form):-with_case(Form).

%%%%%%% general constraint
v_renyou(conj).
v_renyou(vv).
v_renyou(v_y).
v_renyou(v_si).
inf_form(inf).
inf_form(a_inf).

%%%%%%%%%%%%%%% fuku-joshi
%%%%%%%---ba,temo,demo,te,de,tari,dari,shi

dict_pos(ba,
        [core/[pos/adv,form/katei],sc/[],
        aje/[[core/[pos/v,form/katei],sc/[],sem/SEM1]],
        ajn/[[core/[pos/v],aje/[],ajn/[],sem/SEM2]],
        sem/if(SEM1,SEM2) ] ).

%%%%    --temo,demo  (even if)
temo_demo(Form,
        [core/[pos/adj,form/temo],sc/[],
        aje/[[core/[pos/v,form/Form],sc/[],sem/SEM1]],
        ajn/[[core/[pos/v],aje/[],ajn/[],sem/SEM2]],
        sem/even_if(SEM1,SEM2) ] ).
dict_pos(temo,Cat):-temo_demo(Form,Cat),te_form(Form).
dict_pos(demo,Cat):-temo_demo(Form,Cat),demo_form(Form).
demo_form(conj_de).
demo_form(na).

%%%%    --te, de      ( iru,miru, etc.
te_de(Form,
        [core/[pos/v,form/con]2],sc/[],
        aje/[[core/[pos/v,form/Form],sc/[],sem/SEM]],sem/SEM] ).
dict_pos(te,Cat): te_de(Form,Cat),te_form(Form).
dict_pos(de,Cat): te_de(con] de,Cat).
te_form(vv).
te_form(v_y).
te_form(conj_te)
te_form(v_si).

%%%%    --tari,dari
tari_dari(Form,
        [core/[pos/adj],form/tari],sc/[],
        aje/[[core/[pos/v,form/Form],sc/[],sem/SEM1]],
        aju/[[core/[pos/vs],aje/[],ajn/[],sem/SEM2]],
        sem/[SEM1;SEM2]] ).

dict_pos(tari,Cat):-tari_dari(Form,Cat),te_form(Form).
dict_pos(dari,Cat):-tari_dari(con]_de,Cat).

%%%%    --shi     (--shi,--shi)
dict_pos(shi,
       [core/[pos/adj],form/and],sc/[],
        aje/[[core/[pos/v,form/Form],sc/[],sem/SEM1]],
        aju/[[core/[pos/v,form/Form],aje/[],ajn/[],sc/[],sem/SEM2]],
        sem/[SEM1|SEM2] ] );
        inf_form(Form).

%%%%%%%%%% weak verbs
%%%%%    --(te/de)iru,iku,kuru,miru,shimawu: stative verbs
stative_verb(F,
        [core/[pos/v,form/F],ajn/[],sc/[],
        aje/[[core/[pos/v,form/con]2],sc/[],sem/Sem]],sem/[stative|Sem]]).
dict_pos(i,Cat): stative_verb(vv,Cat).
dict_pos(ik,Cat):-stative_verb(vk,Cat).
dict_pos(k,Cat):-stative_verb(vk,Cat).
dict_pos(mi,Cat):-stative_verb(vv,Cat).
dict_pos(shimaw,Cat): stative_verb(vsw,Cat).

%%%%    --dasu,kakeru,hajimeru,owaru,tuzukeru :v(renyou) + v  sub-verb
sub_verb(F,A,
        [core/[pos/v,form/F],ajn/[],sc/[],
        aje/[[core/[pos/v,form/Form],sc/[],sem/Sem]],sem/[A|Sem]] );
                tai_set[Form).
dict_pos(das,Cat):-sub_verb(vs,begin,Cat).
dict_pos(kake,Cat): sub_verb(vv,nearly,Cat).
dict_pos(hajime,Cat):-sub_verb(vv,begin,Cat).
dict_pos(owar,Cat):-sub_verb(ver,end,Cat).
dict_pos(tuzuke,Cat):-sub_verb(vv,continue,Cat).

%%%%%%%%%%%%   Auxiliary verbs (jyo-doushi)    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%    --seru, --saseru : Causative verb (shieki)
shieki_verb(Form,
```

```
        [core/[pos/v,form/vv],ajn/[],
        ajc/[[core/[pos/v,form/Form],sc/[[core/[pos/p,form/ga],sem/Ob]],
                sem/Act]],
        sc/[[core/[pos/p,form/ga],sem/Sbj],
            [core/[pos/p,form/ni],sem/Ob]],
        sem/[cause,Sbj,Ob,Act]] ).

dict_pos(sase,Cat):-shieki_verb(Form,Cat);sara_set(Form).
dict_pos(se,Cat):-shieki_verb(Form,Cat);sere_set(Form).
sara_set(vv).
sara_set(vk).
sere_set(vc_m).
sere_set(v_sa).

%%%%%   --re(ru), --rare(ru) : Passive verb (ukemi)
dict_pos(rare,Cat):-ukemi_verb(Form,Cat);sara_set(Form).
dict_pos(re,Cat):-ukemi_verb(Form,Cat);sere_set(Form).

ukemi_verb(Form,
       [core/[pos/v,form/vv],ajn/[],
        ajc/[[core/[pos/v,form/Form],
                sc/[[core/[pos/p,form/ga],sem/Agt],
                    [core/[pos/p,form/F],sem/Pat]],
                sem/Act]],
        sc/[[core/[pos/p,form/ga],sem/Pat],[core/[pos/p,form/ni],sem/Agt]],
        sem/[passive,Pat,Agt,Act]] );
        obj_case(F).

control_passive([],Sbj,[]).
control_passive(
        [[core/[pos/p,form/Form],ajc/[],ajn/[],sc/[],sem/Sbj]|Rest],
        Sbj,Rest): obj_case(Form).

obj_case(ni).
obj_case(wo).

%%% general auxirialy verb (syusi,rentai form & others)
%%%   nu, ta,u,you,mai
aux_syu_ren(Form,A,
       [core/[pos/v,form/Fm],ajn/[],sc/[],
        ajc/[[core/[pos/v,form/Form],sc/[],sem/Sem]],sem/[A|Sem]] );
                syu_ren(Fm).

aux_verb(Fm,Form,A,
        [core/[pos/v,form/Fm],ajn/[],sc/[],
        ajc/[[core/[pos/v,form/Form],sc/[],sem/Sem]],sem/[A|Sem]]).

%%%%%   --na(i), --nu  : Not
dict_pos(na,Cat):-aux_verb(adj,Form,no,Cat); nai_set(Form).
nai_set(vc_m).
nai_set(vv).
nai_set(vk).
nai_set(v_si).
nai_set(mizen).

dict_pos(nu,Cat):-aux_syu_ren(Form,no,Cat);nu_set(Form).
dict_pos(n,Cat):-aux_syu_ren(Form,no,Cat);nu_set(Form).
dict_pos(zu,Cat):-aux_verb(renyou,Form,no,Cat);nu_set(Form).
dict_pos(ne,Cat):-aux_verb(katei,Form,no,Cat);nu_set(Form).
nu_set(vc_m).
nu_set(vv).
nu_set(vk).
nu_set(v_se).

%%%%%   --ta,da : Past
dict_pos(ta,Cat):-aux_syu_ren(Form,past,Cat);ta_form(Form).
dict_pos(da,Cat):-aux_syu_ren(conj_de,past,Cat).
dict_pos(tara,Cat):-aux_verb(katei,Form,past,Cat),ta_form(Form).
dict_pos(dara,Cat):-aux_verb(katei,conj_de,past,Cat).

ta_form(adj_tt).
ta_form(na_tt).
ta_form(X):-te_form(X).

%%%%%   --u, --you  : suiryou (guess) or ishi(will)
dict_pos(u,Cat):-aux_syu_ren(Form,may,Cat);u_set(Form).
u_set(vc_o).
u_set(mizen).

dict_pos(you,Cat):-aux_syu_ren(Form,may,Cat);you_set(Form).
you_set(vv).
you_set(vk).
you_set(v_si).

%%%%% --rashii : suitei
dict_pos(rashi,Cat):-aux_verb(adj,Form,polite,Cat);rashii_set(Form).
rashii_set(na).
rashii_set(F):-inf_form(F).

%%%%%%  --mai  : not(guess, not(will)
dict_pos(mai,Cat):-aux_syu_ren(Form,no,Cat);mai_set(Form).
mai_set(inf).
mai_set(vv).
mai_set(vk).
mai_set(v_si).

%%%%%%  --ta(i)  : hope
dict_pos(ta,Cat):-aux_verb(adj,Form,hope,Cat),v_renyou(Form).

%%%%%  --sou(da)  : may & I hear
dict_pos(sou,Cat):-aux_verb(na,Form,A,Cat); souda_set(Form,A).
souda_set(F,may):-souda_may_set(F).
souda_may_set(F):-v_renyou(F).
souda_may_set(adj).
souda_may_set(na).
souda_set(inf,hear).
souda_set(a_inf,hear).

%%%%%  --desu, masu.    : teinei
desu(Inf,Cat):-aux_verb(Inf,Form,polite,Cat);desu_set(Form).
```

```
desu_set(rel).
desu_set(adj).
desu_set(na).
dict_pos(desho,Cat):-desu(vc_o,Cat).
dict_pos(deshi,Cat):-desu(v_y,Cat).
dict_pos(desu,Cat):-aux_syu_ren(Form,polite,Cat);desu_set(Form).
masu(Inf,Cat):-aux_verb(Inf,Form,polite,Cat);v_renyou(Form).
dict_pos(mase,Cat):-masu(v_se,Cat).
dict_pos(masho,Cat):-masu(vc_o,Cat).
dict_pos(mashi,Cat):-masu(v_y,Cat).
dict_pos(masu,Cat):-aux_syu_ren(Form,polite,Cat);v_renyou(Form).
dict_pos(masure,Cat):-masu(katei,Cat).
dict_pos(mase,Cat):-masu(imp,Cat).

%%%%%  --nagara,  -tsutsu : root, adverb
dict_pos(nagara,
        [core/[pos/adv,form/adv],sc/[],
         ajc/[[core/[pos/v,form/Form],sc/[],sem/SEM]],
         ajn/[[core/[pos/vl],ajc/[],ajn/[],sem/SEM]],
         sem/[with,SEM,SEM]] );
        nagara_set(Form).
nagara_set(a_inf).
nagara_set(F):-v_renyou(F).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%        suffix.pro       %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%        verb suffix      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%   search_suffix(Form,Word,Cat)    %%%%%%%%%%%%%%%%
%%      << Example >>
%%                        mizen/   renyou/ syusi/  rentai/ katei/   imp/
%%
%% vc?  yom-mu            ma       mi      mu      mu      me       me
%%                        mo(fu)   n
%% vv   iki-ru                     _       ru      ru      re       ro
%% vk   kuru              ko       ki      kuru    kuru    kure     koi
%% vsl  tanjyou suru      si       si      suru    suru    sure     seyo
%%                        sa                                        siro
%%                        se
%% vs2  ai-suru           sa       si      suru    suru    sure     seyo
%% adj  haya i            karo     katt    i       i       kere
%%                                 ku
%% na   kirei-da          daro     datt    da      na      nara
%%                                 de
%%                                 ni
%%      << Form table >>
%% vc?:                   vc_m     conj    inf     rel     katei    imp
%% (vck,vcs,vcl,vcm,      vc_o     conj te (k,t,r,w,ik)
%%  vcr,vcw,vcg,vcb,                conj de (g,n,m,b)
%%  vcik)
%% vv:                    vv       vv      inf     rel     katei    imp
%% vk:                    mizen    v_y     inf     rel     katei    imp
%% vsl:                   v_si     v_si    inf     rel     katei    imp
%%                        v_se
%%                        v_sa
%% vs2:                   vc_m     v_y     inf     rel     katei    imp
%% adj:                   mizen    adj_tt  a_inf   rel     katei    __
%%                                 adj_ku
%% na:                    mizen    na_tt   a_inf   rel     katei    __
%%                                 na_de
%%                                 na_ni

%%%     exception
search_suffix(adj,i,
      [core/[pos/v,form/Form],ajn/[],sc/[],
        ajc/[[core/[pos/v,form/adj],sc/[],sem/Sem]],sem/Sem] );
                adj_syu_ren(Form).

search_suffix(na,na,
      [core/[pos/v,form/rel],sc/[],
        ajc/[[core/[pos/v,form/na],
                sc/[[core/[pos/p],ajc/[],ajn/[],sc/[],sem/Obj]],
                sem/Sem]],
        ajn/[[core/[pos/n,form/n],ajc/[],ajn/[],sc/[],sem/inst(Obj,Type)]],
        sem/inst(Obj,[and,Type,Sem])] ).

%%%     suffix search
search_suffix(Fm,Word,
        [core/[pos/v,form/Form],ajn/[],sc/[],
        ajc/[[core/[pos/v,form/Fm],sc/[],sem/Sem]],sem/Sem) )
        :-suff_s(Word,Fm),syu_ren(Form).

search_suffix(Form2,Word,
        [core/[pos/v,form/Form]],ajn/[],sc/[],
        ajc/[[core/[pos/v,form/Form2],sc/[],sem/Sem]],sem/Sem] )
        :-suff(Word,Form1,Form2).

%%%%%   suffix     syusi, rentai
suff_s(su,ven).
suff_s(mu,ven).
suff_s(mu,vem).
suff_s(bu,veb).
suff_s(ku,vek).
suff_s(ku,veik).
suff_s(tu,vet).
suff_s(u,vew).
suff_s(ru,ver).
suff_s(ru,vv).
suff_s(kuru,vk).
suff_s(suru,vsl).
suff_s(suru,vs2).

%%%%%    Suffix ( mizen, renyou, meirei )

%%%     adj, na
suff(karo,mizen,adj).
suff(katt,adj_tt,adj).
suff(ku,adj_ku,adj).
suff(daro,mizen,na).
```

```
suff(datt,na_tt,na).
suff(de,na_de,na).
suff(ni,na_ni,na).
suff(da,a_inf,na).

%%%     vs, vk    mizen, renyou
suff(se,v_se,vs1).
suff(si,v_si,vs1).
suff(sa,v_sa,vs1).
suff(si,v_y,vs2).
suff(ko,mizen,vk).
suff(ki,v_y,vk).

%%%     v5 mizen
suff(Suf,ve_m,Inf):-ve_m_suff(Suf,Inf).
ve_m_suff(sa,ves).
ve_m_suff(sa,vs2).
ve_m_suff(na,ven).
ve_m_suff(ma,vem).
ve_m_suff(ba,veb).
ve_m_suff(ka,vek).
ve_m_suff(ka,veik).
ve_m_suff(ta,vet).
ve_m_suff(wa,vew).
ve_m_suff(ra,ver).

suff(Suf,ve_o,Inf):-ve_o_suff(Suf,Inf).
ve_o_suff(so,ves).
ve_o_suff(so,vs2).
ve_o_suff(no,ven).
ve_o_suff(mo,vem).
ve_o_suff(bo,veb).
ve_o_suff(ko,vek).
ve_o_suff(ko,veik).
ve_o_suff(to,vet).
ve_o_suff(wo,vew).
ve_o_suff(ro,ver).

%%%     v5 renyou
suff(si,v_y,ves).

suff(Suf,conj,Inf):-ve_conj_suff(Suf,Inf).
ve_conj_suff(ni,ven).
ve_conj_suff(mi,vem).
ve_conj_suff(bi,veb).
ve_conj_suff(wi,vew)
ve_conj_suff(gi,veg).
ve_conj_suff(ki,vek).
ve_conj_suff(ki,veik).
ve_conj_suff(ti,vet).
ve_conj_suff(ri,ver).

%%%     v5 renyou   onbin
suff(i,conj_te,vek).
suff(i,conj_de,veg).
suff(t,conj_te,vet).
suff(t,conj_te,vew).
suff(n,conj_de,veb).
suff(n,conj_de,vem).
suff(t,conj_te,ver).
suff(n,conj_te,ven).
suff(t,conj_te,veik).

%%%     katei  ( only   ba)
suff(Suf,katei,Inf):-suff_ba(Suf,Inf).
suff_ba(se,ves).
suff_ba(ne,ven).
suff_ba(me,vem).
suff_ba(be,veb).
suff_ba(ke,vek).
suff_ba(ke,veik).
suff_ba(te,vet).
suff_ba(we,vew).
suff_ba(re,ver).
suff_ba(re,vv).
suff_ba(kure,vk).
suff_ba(sure,vs1).
suff_ba(sure,vs2).
suff_ba(kere,adj).
suff_ba(nara,na).


%%%     meirei
suff(Suf,imp,Inf):-imp_suff(Suf,Inf).
imp_suff(se,ves).
imp_suff(ne,ven).
imp_suff(me,vem).
imp_suff(be,veb).
imp_suff(ke,vek).
imp_suff(ke,veik).
imp_suff(te,vet).
imp_suff(we,vew).
imp_suff(re,ver).
imp_suff(ro,vv).
imp_suff(koi,vk).
imp_suff(siro,vs1).
imp_suff(sero,vs1).
imp_suff(seyo,vs2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% noun.dic %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% common noun, proper nouns %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%% common nouns.
c_noun(Sem,[core/[pos/n,form/n],sem/inst(Obj,Sem)]).

dictl(ari,Cat):-c_noun(ant,Cat).
dictl(esa,Cat):-c_noun(food,Cat).
dictl(gakusha,Cat):-c_noun(scholar,Cat).
dictl(gyouretsu,Cat):-c_noun(row,Cat).
```

```
dictl(hana,Cat):-e_noun(flower,Cat).
dictl(hatarakiari,Cat):-e_noun(ari0,Cat).
dictl(hito,Cat):-e_noun(people,Cat).
dictl(hon,Cat):-e_noun(book,Cat).
dictl(ishi,Cat):-e_noun(stone,Cat).
dictl(jimen,Cat):-e_noun(ground,Cat).
dictl(katamari,Cat):-e_noun(block,Cat).
dictl(michi,Cat):-e_noun(road,Cat).
dictl(michishirube,Cat):-e_noun(row,Cat).
dictl(michisuji,Cat):-e_noun(road,Cat).
dictl(mokutekichi,Cat):-e_noun(goal,Cat).
dictl(natsu,Cat):-e_noun(summer,Cat).
dictl(niwa,Cat):-e_noun(garden,Cat).
dictl(satou,Cat):-e_noun(sugar,Cat).
dictl(solo,Cat):-e_noun(out,Cat).
dictl(sumi,Cat):-e_noun(corner,Cat).
dictl(tsubu,Cat):-e_noun(grain,Cat).
dictl(yousu,Cat):-e_noun(yousu,Cat).
dictl(yukute,Cat):-e_noun(way,Cat).

%%%%%% proper nouns.
p_noun(Sem,[core/[pos/n,form/n],sem/Sem]).

dictl(america,Cat):-p_noun(america,Cat).
dictl(hiroshi,Cat):-p_noun(hiroshi,Cat).
dictl(ken,Cat):-p_noun(ken,Cat).
dictl(naomi,Cat):-p_noun(naomi,Cat).
dictl(wilson,Cat):-p_noun(wilson,Cat).

%%%%% jibun (self)
dictl(jibun, [core/[pos/n,form/n],
        refl/[[core/[pos/p,form/ga],sem/Sem]],
        sem/Sem]).
dictl(jken, [core/[pos/n,form/n],
        refl/[[core/[pos/p,form/ga],sem/ken]],
        sem/ken]).
%%%%% nioi, koto, tokoro, etc. (pseudo relative clause)
dictl(nioi, [core/[pos/n,form/n],sc/[[core/[pos/v,form/rel]]],sem/smell]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%  verb.dic                          %%%%%%%
%%%%%%  Verbs except saheni v     %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% vi (!subcat==1:  - ga)
ga_verb(F,Act,
        [core/[pos/v,form/F],
        sc/[[core/[pos/p,form/ga],sem/Sbj]]],
        sem/[Act,Sbj]]).

%%%%% vt (!subcat==2:)
%%   - ga  - wo
ga_wo_verb(F,Act,
        [core/[pos/v,form/F],
        sc/[[core/[pos/p,form/ga],sem/Sbj],
            [core/[pos/p,form/wo],sem/Obj]]],
        sem/[Act,Sbj,Obj]]).
%%   --ga  --ni
ga_ni_verb(F,Act,
        [core/[pos/v,form/F],
        sc/[[core/[pos/p,form/ga],sem/Sbj],
            [core/[pos/p,form/ni],sem/Obj]]],
        sem/[Act,Sbj,Obj]]).

%%   --ga  -wo --ni
ga_wo_ni_verb(F,Act,
        [core/[pos/v,form/F],
        sc/[[core/[pos/p,form/ga],sem/Sbj],
            [core/[pos/p,form/wo],sem/Iob],
            [core/[pos/p,form/ni],sem/Iob]],
        sem/[Act,Sbj,Iob,Iob]]).

%%% temp feature
%% kiru,akeru
temp1([temp/t(1,2,2,f,f)]).
%% anki suru
temp2([temp/t(1,2,0,f,n)]).
%% aruku,yomu
temp3([temp/t(1,2,2,f,0)]).
%% matu,damaru
temp4([temp/t(1,1,1,f,0)]).
%% suwaru,kekkon suru
temp5([temp/t(1,2,2,0,f)])
%% suwaru,sinu
temp6([temp/t(1,2,0,0,f)]).
%% niru,tadayou
temp7([temp/t(0,0,0,0,u)]).
%% odoroku,tumaduku
temp8([temp/t(1,2,2,0,0)])

%%% lexical entry
dictl(age,Cat):-ga_wo_ni_verb(vv,give,Cat).
dictl(ai,Cat):-ga_wo_verb(vs2,love,Cat).
dictl(ake,Cat):-ga_wo_verb(vv,open,Cat),temp1(Cat).
dictl(aruku,Cat):-ga_verb(vek,walk,Cat).
dictl(chigaw,Cat):-ga_verb(vew,differ,Cat).
dictl(chirijirininar,Cat):-ga_verb(ver,scatter,Cat).
dictl(deki,Cat):-ga_verb(vv,can,Cat).
dictl(der,Cat):-ga_ni_verb(vv,out,Cat).
dictl(hashi,Cat):-ga_verb(ver,run,Cat).
dictl(hazure,Cat):-ga_wo_verb(vv,off,Cat).
dictl(i,Cat):-ga_verb(vv,be,Cat).
dictl(ik,Cat):-ga_ni_verb(vek,go,Cat).
dictl(isog,Cat):-ga_verb(veg,hurry,Cat).
dictl(kaer,Cat):-ga_ni_verb(ver,return,Cat).
dictl(kag,Cat):-ga_wo_verb(veg,smell,Cat).
dictl(kak,Cat):-ga_wo_verb(vek,write,Cat).
dictl(kaw,Cat):-ga_wo_verb(vew,buy,Cat).
dictl(kawar,Cat):-ga_verb(ver,lose,Cat).
dictl(kat,Cat):-ga_ni_verb(vet,win,Cat).
```

```
dictl(ker,Cat):-ga_wo_verb(vv,kick,Cat).
dictl(ki,Cat):-ga_wo(vv,wear,Cat),templ(Cat).
dictl(majiwar,Cat):-ga_verb(ver,cross,Cat).
dictl(manab,Cat):-ga_wo_verb(veb,learn,Cat).
dictl(mayow,Cat):-ga_verb(vew,maze,Cat).
dictl(mi,Cat):-ga_wo_verb(vv,look_at,Cat).
dictl(midare,Cat):-ga_verb(vv,confuse,Cat).
dictl(mitsuke,Cat):-ga_wo_verb(vv,find,Cat).
dictl(mot,Cat):-ga_wo_verb(vet,have,Cat).
dictl(nar,Cat):-ga_ni_verb(vv,become,Cat).
dictl(not,Cat):-ga_ni_verb(ver,on,Cat).
dictl(ok,Cat):-ga_wo_ni_verb(vek,put,Cat).
dictl(omow,Cat):-ga_wo_verb(vew,think,Cat).
dictl(os,Cat):-ga_wo_verb(ves,push,Cat).
dictl(saeqi,Cat):-ga_wo_verb(ver,interrupt,Cat).
dictl(sagas,Cat):-ga_wo_verb(ves,search,Cat).
dictl(shir,Cat):-ga_wo_verb(ver,know,Cat).
dictl(shin,Cat):-ga_verb(ven,die,Cat).
dictl(susum,Cat):-ga_ni_verb(vem,go,Cat).
dictl(tador,Cat):-ga_wo_verb(ver,follow,Cat).
dictl(tasuker,Cat):-ga_wo_verb(vv,help,Cat).
dictl(toor,Cat):-ga_ni_verb(ver,pass,Cat).
dictl(tsuger,Cat):-ga_wo_ni_verb(vv,tell,Cat).
dictl(tsuk,Cat):-ga_ni_verb(vek,reach,Cat).
dictl(tsuzuku,Cat):-ga_verb(vek,continue,Cat).
dictl(wakar,Cat):-ga_wo_verb(ver,understand,Cat).
dictl(yom,Cat):-ga_wo_verb(vem,read,Cat).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%    sahen.dic
%%%%%    sahen-n,v dictionary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%    verb : SFRU
suru_verb(F,
        [core/[pos/vs,form/F],adc/Adj,sc/SC,sem/Sem]):-
        suru(Adj,SC,Sem).

dictl(shi,Cat):-suru_verb(v_si,Cat).
dictl(se,Cat):-suru_verb(v_se,Cat).
dictl(sa,Cat):-suru_verb(v_sa,Cat).
dictl(sure,Cat):-suru_verb(katei,Cat).
dictl(shito,Cat):-suru_verb(imp,Cat).
dictl(sero,Cat):-suru_verb(imp,Cat).

%%%%%    sa-hen verbs (do)
sahen_verb(F,Act,
        [core/[pos/v,form/F],sc/[[core/[pos/p,form/ga],sem/Sbj]],
        sem/[Act,Sbj]] ).
dictl(tanjou,Cat):-sahen_verb(vsl,birth,Cat).

%%%%%    sa-hen nouns.
%% dictl(chousa,
%%         [pos/n,NFORM,[],[],SC,[],[],Sem] ):
%%       sahen_noun(NFORM,examine,SC,SEM).
%% sahen_noun(n,SEM,[],SEM).

%% sahen_noun(ns,SEM1,SC,[SEM1,Sbj,Obj]):- adn_wo_ga(SC,Obj,Sbj).
dictl(chousa,
        [core/[pos/n,form/ns],sc/SC,sem/[examine,Sbj,Obj]] ),
        adn_wo_ga(SC,Obj,Sbj).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%    adject.dic
%%%%%    adjective, adjective-verb
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%    Adjectives       %%%%%%%%%%%%%%
Adjective(A,
        [core/[pos/v,form/adj],
        sc/[[core/[pos/p,form/ga],sem/Obj]],
        sem/[A,Obj]] ).
dictl(aka,Cat):-adjective(red,Cat).
dictl(siro,Cat):-adjective(white,Cat).
dictl(kuro,Cat):-adjective(black,Cat).
dictl(ooki,Cat):-adjective(big,Cat).
dictl(yo,Cat):-adjective(good,Cat).

%%%%%    na (adjective verb)
ajverb(A,
        [core/[pos/v,form/na],
        sc/[[core/[pos/p,form/ga],sem/Obj]],
        sem/[A,Obj]] ).
dictl(kirei,Cat):-ajverb(beautiful,Cat).
dictl(kaiteki,Cat):-ajverb(confortable,Cat).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%          etc.dic
%%%%%%%          dictionary of other words
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%% rentai-shi
rentaishi(A,
        [core/[pos/adn, form/adn],
        ajn/[[core/[pos/n,form/ni],ajc/[],ajn/[],sc/[],sem/SEM]],
        sem/[A|SEM]]
).
dictl(sono,Cat):-rentaishi(the,Cat).
dictl(kono,Cat):-rentaishi(this,Cat).
dictl(ano,Cat):-rentaishi(that,Cat).
dictl(ippikino,Cat):-rentaishi(one,Cat).

%%%%%%% fuku-shi (adverb)

adverb(MODIFY,
        [core/[pos/adv, form/adv], ajn/[[core/[pos/v],sem/SEM]],
        sem/[MODIFY|SEM]]).
dictl(yoku,Cat):-adverb(often,Cat).
dictl(zutto,Cat):-adverb(continue,Cat).
dictl(hajimeni,Cat):-adverb(first,Cat).
dictl(sukoshi,Cat):-adverb(slightly,Cat).
dictl(yagate,Cat):-adverb(in_the_end,Cat).
dictl(tsugitsugito,Cat):-adverb(continue,Cat).
```

```
dict1(dandanni,Cat):-adverb(gradually,Cat).
dict1(komakani,Cat):-adverb(minutely,Cat).
dict1(kesshite,
        [core/[pos/adv,form/adv],ajn/[[core/[pos/v],sem/[not[SEM]]],
        sem/[never[SEM]]]).
```