ICOT Technical Memorandum: TM-0646

TM-0646

GHCによるATMS

釘宮亮二

December, 1988

ICOT

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

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

TM

GHCによるATMS

1988年12月

釘宮 亮二

目次

1.　ESPからGHCへの移植について

1. 1　概要

本研究は並列プログラミング言語GHC[古川他 87]の応用の一つとして逐次プログラミング言語で書かれた非単調推論のシステムATMS[飯島,井上 88]のGHCへの移植を試みたものである。

内容は順に,移植する際の方針,移植したシステムの仕様,システムの使用手引き,プログラムリストとなっている。　プログラムはPrologで書かれているが,シンタックスの簡単な変更(APPENDIXのBを参照)によってGHCのプログラムとして作動する。

非単調推論のシステムATMS(An Assumption-based TMS)の概要及びメカニズムの詳細については[de Kleer 86],[飯島,井上 88]等を参照して頂きたい。

1. 2　移植の方針について

GHC[古川他 87]は並列プログラミング言語である。　逐次プログラミング言語で書かれたプログラムのGHCへの移植を試みることはGHCの応用プログラミングの研究の上で意義があると思われる。　本研究ではESPで書かれた非単調推論システム[飯島,井上 88]の移植を試みたので以下にその概要を記す。

移植の際には以下の3点を基本的な方針とした。

(1) バックトラックの少ないプログラムに書き換える。

(2) データへのアクセス,データの生成は全てストリームを通じておこなう。

(3) 計算の中断を可能な限り抑える。

これらは全て元のプログラムがPrologで書かれていたとしても適用できることである。移植に当たっては,ESPのようなオブジェクト指向言語としてのインプリメントは特にせず,オブジェクトは全てストリームを流れるデータとした。　以下では上の3点について順に説明を加える。

(1)については,論理型の言語を関数型の言語にコンパイルする様な事である。　移植後のプログラムは入出力が明示的なより手続き的なプログラムへと展開されている。

(2)については,データへのアクセス,データの生成をそれぞれおこなうESPの基本的な組み込み述語get_slot,set_slotをGHC用に書き下した。　DEC-10 Prologであればそれぞれ組み込み述語assert,retractを書き下すことに相当する。　この場合,オブジェクト指向言語であるESPではデータがオブジェクト;属性(スロット),属性値(スロット値)の3つ組

に揃えられているので,統一的な扱いが可能であるという点で簡単であった。　新たに書き下した述語get_slot,set_slotは以下の通りである。

```
get_slot(Obj,Slot,V,[Data|D])
     :- Data=(Obj,Slots) |
     get_slot1(Slots,Slot,V).                % get_slot1はオブジェクトの持つ各
get_slot(Obj,Slot,V,[Data|D])                % スロットから該当するスロットの値
     :- Data¥=(Obj,    _) |                  % を取り出す
     get_slot(Obj,Slot,V,D).
get_slot(_,   _,    V,[])
     :- true | V='undef'.

set_slot(Obj,Slot,V,[Data|D0],D)
     :- Data=(Obj,Slots) |
     set_slot1(Slots,Slot,V,New_Slots),      % set_slot1はオブジェクトのスロット
     D=[(Obj,New_Slots)|D0].                 % に新しいスロット値を書き込む
set_slot(Obj,Slot,V,[Data|D0],D)
     :- Data¥=(Obj,    _) |
     D=[Data|D1],
     set_slot(Obj,Slot,V,D0,D1).
set_slot(Obj,Slot,V,[],D)
     :- true | D=[(Obj,[(Slot,V)])].
```

get_slotは参照用のストリームを1つ,set_slotはそれぞれ参照,書き込み用の2つのストリームを持っており,ストリームの上にオブジェクト毎のデータが送られている。　移植後のプログラムはset_slotのように一般に入出力の2つのストリームの変数が加えられている。　各プロセスがデータを出力するのは前段のプロセスから全てのデータが送られてくるのを待つ必要はなく,送られてくるデータが一つでもあれば,後段のプロセスへデータの出力をおこなうことができる。

(3)は単に効率の点から望ましいということである。　元のプログラムのアルゴリズムは特に変更せず,(1)(2)の点以外は素直にGHCのプログラムに書き換えた。　移植されたGHCのプログラムは計算の中断をおこさず,deterministicに処理をおこなう。

2.　GHCによるATMSの仕様

2. 1　移植したATMSの特徴

ESPはオブジェクト指向の言語であり,プログラムはメソッドとしてクラス毎に定義する。移植に際しては元のESPのプログラムにおけるクラスへのモジュール分割はほとんどそのまま保存した。　移植後のプログラムはオブジェクトをストリームを流れるデータとして生成・管理する以外に特にオブジェクト指向言語としての支援は実現していない。したがって以下に示すクラスとは主に生成・管理するデータの種別によるプログラムのモジュール分割以上のものではない。

(1)　basicクラス
　　各クラスで用いる共通的な手続きの管理
(2)　nodeクラス
　　ノード・オブジェクトの生成及び管理
(3)　justificationクラス
　　理由付け・オブジェクトの生成及び管理
(4)　assumptionクラス
　　仮説オブジェクトの生成及び管理
(5)　environmentクラス
　　環境オブジェクトの生成及び管理
(6)　nogoodクラス
　　矛盾する環境の管理
(7)　interfaceクラス
　　ユーザ(問題解決器)に対するATMSとのインターフェースの提供
(8)　initクラス
　　ATMS全体の初期化
(9)　atmsクラス
　　ユーザ(問題解決器)がATMSに与える命令の解釈,実行
(10)　printクラス
　　ATMS内部表現の表示
(11)　solverクラス
　　ATMSを用いた簡単な問題解決器

2. 2　データ構造（クラスのスロット構成）

元のESPプログラムの各オブジェクトはストリームを流れるデータとして実現し,各オブジェクトの持つ属性(スロット)はほとんど変更なしに移植した。

(1)　node クラス

クラス・スロット

1	contra_node	矛盾を示すノード・オブジェクトを記憶する。
2	nodes	生成済みノード・インスタンスを記憶する。
3	node_queue	ラベル更新を待つノード列を記憶する。
4	counter	生成インスタンス・オブジェクトの数を記憶する。

インスタンス・スロット		
1	index	何番目に生成されたオブジェクトか記憶する。
2	datum	ユーザ(問題解決器)が与えたデータを記憶する。
3	status	本ノードが信じられるか否かを示す。(IN or OUT)
4	label	本ノードのラベルを記憶する。
5	c_justs	本ノードを導く理由付けを記憶する。
6	a_justs	本ノードを前件部に持つ理由付けを記憶する。
7	assumption	本ノードに付加する仮説オブジェクトを記憶する。

(2) justification クラス

クラス・スロット		
1	justifications	生成済み理由付け・インスタンスを記憶する。
2	counter	生成インスタンス・オブジェクトの数を記憶する。

インスタンス・スロット		
1	index	何番目に生成されたオブジェクトか記憶する。
2	type	本理由付けのタイプ(ユーザ指定)を記憶する。
3	consequence	本理由付けのconsequenceを記憶する。
4	antecedents	本理由付けのantecedentsを記憶する。

(3) assumption クラス

クラス・スロット		
1	assumptions	生成済みassumption・インスタンスを記憶する。
2	counter	生成インスタンス・オブジェクトの数を記憶する。

インスタンス・スロット		
1	index	何番目に生成されたオブジェクトか記憶する。
2	node	本仮説の内容を持つノードを記憶する。
3	environments	本仮説を含む環境を記憶する。

(4) environment クラス

クラス・スロット		
1	empty_env	空の環境オブジェクトを記憶する。
2	env_table	仮説数をインデックスとして,環境をテーブル形式で記憶する。
3	environments	生成済みenvironment・オブジェクトを記憶する。
4	indices	env_tableのインデックスを記憶する。
5	counter	生成インスタンス・オブジェクトの数を記憶する。

インスタンス・スロット		
1	index	何番目に生成されたオブジェクトか記憶する。
2	ass_size	本環境が持つ仮説の数を記憶する。
3	assumptions	本環境が持つ仮説を記憶する。
4	nodes	本環境をラベルに持つノードを記憶する。
5	contradictory	本環境が矛盾するか否かを示すフラグを記憶する。

(5) nogood クラス

クラス・スロット		
1	nogood_table	仮説数をインデックスとして,矛盾する環境をテーブル形式で記憶する。
2	indices	nogood_tableのインデックスを記憶する。

% nogoodクラスにはインスタンス・オブジェクトはない。
% 上記以外のクラスにはデータとしてのオブジェクトは存在しない。

2. 3 各モジュール内の手続き（述語）

モジュールとしての各クラスに定義された述語の主なものを以下に示す。これらも元のESPプログラムのものと(basicクラス以外)ほとんど同じである。ATMSの外部仕様は(11)のatmsクラスと(9)のprintクラスが示す。

(1) node クラス

述語		
1	make_node	ノード・オブジェクトを生成する。
2	node_updating	ノード(ラベル)を更新する。
3	label_updating	ラベルを更新する。 (node_updatingから呼び出される)

(2) justification クラス

述語		
1	make_justification	理由付け(justification)オブジェクトを生成する。
2	get_envs_from_just	理由付けから環境を求める。

(3) assumption クラス

述語		
1	make_assumption	仮説オブジェクトを生成する。
2	get_env_from_asss	仮説から環境を生成する。

(4) environment クラス

述語		
1	make_environment	環境オブジェクトを生成する。
2	subsumed_check	環境(複数)がある環境に含まれるか否かを判断する。
3	env_subsumed_p	ある環境が他の環境を含むか否かを判断する。
4	get_new_envs	いくつかの環境から組み合わせた新しい環境を作成する。
5	union_envs	2つの環境を合成する。
6	push_env	環境に仮説を追加して新しい環境を作成する。
7	push_env_table	環境をenv_tableに登録する。
8	lookup_env	ある環境が存在するか捜す。
9	env_contradictory_p	環境が矛盾するか否かをチェックする。
10	remove_env_from_labels	各ノードのラベルからある環境を除く。

(5) nogood クラス

述語		
1	nogood_updating	nogoodを更新する。
2	push_nogood_table	矛盾する環境をnogood_tableへ登録する。

(6) interface クラス

述語		
1	lookup_assumption	あるノードを指示する仮説があれば,その仮説のオブジェクトを求める。
2	assume_node	あるノードを指示する仮説を立てる。
3	premise_node	あるノードがpremiseであることを宣言する。
4	nogood_nodes	矛盾するノードの組み合せを宣言する。

(7) solver クラス

述語		
1	breadth_search	横型探索で解を求める。
2	depth_search	縦型探索で解を求める。

(8) init クラス

述語		
1	init	ATMSが管理するデータを初期化する。

(9) print クラス

インタープリターの述語atms/1に与えるメッセージ・ストリーム中に項print/1(printを関数記号,以下の項を引数とする項)を加えることにより以下の機能を果たす。

('/n'という記法は,その左側に書いた述語記号,ないし関数記号がn引数のものであることを示す)

メッセージ		
1	status/1	指定するノードの状態(IN or OUT)を表示する。
2	statuses	全ノードの状態(IN or OUT)を表示する。
3	node/1	指定するノードの各スロット値を表示する。

4	nodes	全ノードの各スロット値を表示する。
5	just/1	指定する理由付けの各スロット値を表示する。
6	justs	全理由付けの各スロット値を表示する。
7	ass/1	指定する仮説の各スロット値を表示する。
8	asss	全仮説の各スロット値を表示する。
9	env/1	指定する環境の各スロット値を表示する。
10	envs	全環境の各スロット値を表示する。
11	node_justs	指定するノードの理由付けを表示する。
12	env_with_index/1	指定する仮説数を持つ環境を表示する。
13	env_datum/1	指定する環境が持つ仮説のデータを表示する。
14	env_datums	全環境に関して各環境が持つ仮説のデータを表示する。
15	table	environmentテーブルとnogoodテーブルの内容表示する。
16	context/1	指定するノードがラベルを持つ場合，その理由付けをたどる。

(10) basic クラス

述語		
1	get_slot	指定オブジェクトの指定スロットの値を得る。
2	set_slot	指定オブジェクトの指定スロットに値をセットする。
3	add_slot	指定オブジェクトの指定スロットに値を追加する。
4	new	指定した名前のインスタンス・オブジェクトを生成する。

5	class_object	指定オブジェクトが指定クラスのインスタンスであるか否か判断する。
6	add_with_index	指定オブジェクトの指定スロットにインデックスを付けて値を追加する。
7	get_with_index	指定オブジェクトの指定スロットに指定インデックスを持つ値を得る。
8	remove_with_index	指定オブジェクトの指定スロットに指定インデックスを持つ値を削除する。
9	incf	指定オブジェクトの指定スロットの値を1増加させる。
10	length	指定リストの長さを求める。
11	append	2つのリストのappendを求める。
12	member	指定要素が指定リストに入っているか
13	add	2つの整数の和を求める。
14	dolcte	指定要素を指定リストから削除する。
15	ascending_ordered_pushnew	指定要素を昇順にソートされたリストに追加する。
16	mapcan	リストを各要素とするリストの全要素からその各要素を取り出しリストを作る。
17	rest	指定リストから指定要素以降のリストを求める。
18	ordered_insert	指定要素をindexのスロット値の昇順によソートされたリストに追加する。
19	gensym	新しいアトムを生成する。
20	prolog	prologのゴールを実行する。

(11) atms クラス

ユーザ(問題解決器)がATMSに与えることができる命令の種類(述語atms/1に与えること

のできるメッセージ・ストリームの要素の種類)は以下の通りである。

メッセージ		
1	premise/1	前提データを宣言する。
2	assume/1	仮説データを宣言する。
3	justify/2	理由付け(第1引数=帰結, 第2引数=前提のリスト)を宣言する。
4	nogood/1	矛盾するデータの組み合せ(リスト)を宣言する。
5	datum_to_node/2	指定するデータ(第1引数)に対応するノード・オブジェクト(第2引数)を得る。
6	get_solution/3	可能な組み合せの環境(第3引数)を求める。(第1引数=オルタナチブな集合(リスト)を要素とするリスド, 第2引数=探索戦略(縦型(depth)か横型(breadth))
7	status/2	指定するデータ(第1引数)の状態(IN or OUT)を求める。
8	implied_p/3	指定するデータ(第1引数)が指定する環境(第2引数)に含まれるか判断する。
9	consistency_p/3	指定するデータ(第1引数)が指定する環境(第2引数)と矛盾しないか判断する。
10	get_justifications/2	指定するデータ(第1引数)の理由付けを求める。
11	get_assumption/2	指定するデータ(第1引数)に対応する仮説を求める。
12	init/0	ATMSを初期化する。('init'の前に送ったメッセージの効果は無視される)

% 上記以外にユーザが与えることができるメッセージとしてprint/1がある((9)参照)。

3. 本ATMSの使用手引き

本ATMSはPrologでインプリメントされているがシンタックスの簡単な変更(APPENDIXのBを参照) によってGHCのプログラムとしても作動させることができる。

3. 1 ATMS ファイル一覧 (2.1の各クラスに対応)

(1) basic.prl	(2) node.prl	(3) justif.prl
(4) assump.prl	(5) enviro.prl	(6) nogood.prl
(7) interf.prl	(8) init.prl	(9) atms.prl
(10) print.prl	(11) solver.prl	(12) atms.com

3. 2 ATMS 立ち上げ手順

A. Prologのプログラムとして実行

Prologのプログラムとしては DEC-10 Prolog, Quintus Prologの上でそのままで実行させることができる。

(1) コマンド・ファイルatms.comをロードする。
(述語consult('atms.com')を実行)

(2) 表示されたコマンド・メニューにしたがい,以下の述語を実行する。

c --- ATMSのコア((1)-(9)のプログラム)をロード
p --- printルーチン((10)のプログラム)をロード
s --- 問題解決器((11)のプログラム)をロード

B. GHCのプログラムとして実行

以下の手続きを前以て行えば, DEC-10 Prolog上のGHC処理系[古川他 87]では上のAの(1), (2)と同一の手続きにより立ち上げることができる。

(0) APPENDIXのBに従って(1)-(11)の各ファイルの内容を変更し, ファイル名を

'***.prl' --> '***.ghc'

の形にする。

(ロードする時, 同一のデイレクトリに変更前のファイルがあると書き換えられてしまうので注意)

3. 3 ATMSのユーザ・インターフェース

トップレベルの述語はatms/1であり,引数にメッセージ・ストリームとして命令を与える(命令の種類は2.3の(11)を参照)。

ATMSが内部で管理している情報を見るためには,メッセージ・ストリームにprint命令を与えればよい(print命令の種類は2.3の(9)を参照)。

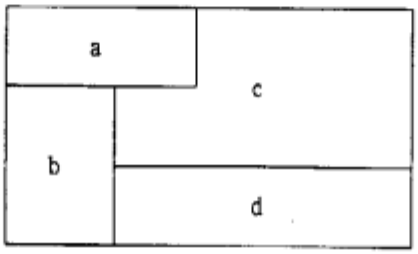
3. 4　ATMSの例題プログラム

[飯島,井上 88]に示された例題と対応するプログラムの実行例を以下に示す。

(DEC-10 Prolog)

例題

次のような領域(a,b,c,d)に色(赤,青,緑)を塗りたい。　ただし,隣合わせの領域に同じ色を塗ってはいけない。何通りの色の塗り方があるか?



プログラム

```
:- public example1/3.
example1(Strategy,Solutions,MsgS_T)
    :- true,!,
    MsgS=[assume(a(red)),assume(a(blue)),assume(a(green)),
          assume(b(red)),assume(b(blue)),assume(b(green)),
          assume(c(red)),assume(c(blue)),assume(c(green)),
          assume(d(red)),assume(d(blue)),assume(d(green)),
          nogood([a(red),b(red)]),nogood([a(blue),b(blue)]),
          nogood([a(green),b(green)]),nogood([a(red),c(red)]),
          nogood([a(blue),c(blue)]),nogood([a(green),c(green)]),
          nogood([b(red),c(red)]),nogood([b(blue),c(blue)]),
          nogood([b(green),c(green)]),nogood([b(red),d(red)]),
          nogood([b(blue),d(blue)]),nogood([b(green),d(green)]),
          nogood([c(red),d(red)]),nogood([c(blue),d(blue)]),
          nogood([c(green),d(green)]),
          get_solution(Strategy,
                       [[a(red),a(blue),a(green)],[b(red),b(blue),b(green)],
                        [c(red),c(blue),c(green)],[d(red),d(blue),d(green)]]
                       Solutions)|MsgS_T],
    atms([init|MsgS]).
```

実行例

```
| ?- example1(depth,Envs,[]).

Envs = [env68,env62,env53,env48,env39,env33]

yes
| ?- example1(depth,[Env|Envs],[print(env_datum(Env))]).

      ------------------------------
        ASSUMPTION-DATUM in ENV
      ------------------------------
       environment   : env68
       contradictory : []
       datum         : a(green)
       datum         : b(blue)
       datum         : c(red)
       datum         : d(green)

Env = env68,
Envs = [env62,env53,env48,env39,env33]

yes
| ?- example1(breadth,Envs,[]).

Envs = [[env69],[env67],[env64],[env59],[env56],[env54]]

yes
| ?- example1(breadth,[[Env]|Envs],[print(env_datum(Env))]).

      ------------------------------
        ASSUMPTION-DATUM in ENV
      ------------------------------
       environment   : env69
       contradictory : []
       datum         : a(green)
       datum         : b(blue)
       datum         : c(red)
       datum         : d(green)

Env = env69,
Envs = [[env67],[env64],[env59],[env56],[env54]]
```

［ 謝 辞 ］

本研究の機会を与えて下さったICOT第5研究室 藤井裕一室長ならびに御助力頂きましたICOT第5研究室の皆様に感謝致します。

［参考文献］

[de Kleer 86] de Kleer, J., An Assumption-based TMS, Artificial Intelligence 28 (1986) 127-162.

[飯島, 井上 88] 飯島, 井上, ESPによるATMS -第1版-, ICOT Tech. Memorandom TM-467.

[古川他 87] 古川他, 並列論理型言語GHCとその応用, 共立出版.

APPENDIX

A. プログラムリスト(Prolog version)

```
%%% (1) basic.prl

:- mode get_slot(+,+,-,+).
get_slot(Obj,Slot,V,[(Obj,Slots)|D]) :-
    true,!,
    get_slot(Slots,Slot,V).
get_slot(Obj,Slot,V,[Data|D]) :-
%%  Data\==(Obj,_),
    true,!,
    get_slot(Obj,Slot,V,D).
get_slot(_,_,V,[])
    :- true,!,V='undef'.

:- mode set_slot(+,+,+,+,-).
set_slot(Obj,Slot,V,[(Obj,Slots)|D0],D) :-
    true,!,
    set_slot(Slots,Slot,V,New_Slots),
    D=[(Obj,New_Slots)|D0].
set_slot(Obj,Slot,V,[Data|D0],D) :-
%%  Data\==(Obj,_),
    true,!,
    D=[Data|D1],
    set_slot(Obj,Slot,V,D0,D1).
set_slot(Obj,Slot,V,[],D)
    :- true,!,D=[(Obj,[(Slot,V)])].

:- mode new(+,+,-).
new(Obj,D0,D) :- true,!,D=[(Obj,[])|D0].

:- mode class_object(+,+,-).
class_object(Obj,Obj1,Result)
    :- prolog(atom(Obj1)),!,
    prolog(name(Obj,ObjL)),
    prolog(name(Obj1,Obj1L)),
    class_obj(ObjL,Obj1L,Result).

:- mode add_slot(+,+,+,+,-).
add_slot(Obj,Slot,Value,D0,D)
    :- true,!,
    get_slot(Obj,Slot,Values,D0),
    set_slot(Obj,Slot,[Value|Values],D0,D).

:- mode add_with_index(+,+,+,+,+,-).
add_with_index(Obj,Slot,Index,Value,D0,D)
    :- true,!,
    get_slot(Obj,Slot,Values,D0),
    add_with_index(Index,Value,Values,New_Values),
    set_slot(Obj,Slot,New_Values,D0,D).

:- mode get_with_index(+,+,+,-,+).
get_with_index(Obj,Slot,Index,Value,D)
    :- true,!,
    get_slot(Obj,Slot,Values,D),
    get_with_index(Index,Value,Values).

:- mode remove_with_index(+,+,+,+,+,-).
remove_with_index(Obj,Slot,Index,Value,D0,D)
    :- true,!,
    get_slot(Obj,Slot,Values,D0),
    remove_with_index(Index,Value,Values,New_Values),
    set_slot(Obj,Slot,Values,D0,D).

:- mode incf(+,+,+,-).
incf(Obj,Slot,D0,D)
    :- true,!,
```

```
    get_slot(Obj,Slot,N,D0),
    add(N,1,N1),
    set_slot(Obj,Slot,N1,D0,D).

:- mode length(+,-).
length([],L) :- true,!,L=0.
length([H|T],L)
    :- true,!,
    length(T,L1),add(L1,1,L).

:- mode append(+,+,-).
append([],B,C) :- true,!,C=B.
append([H|T],B,C)
    :- true,!,append(T,B,C1),C=[H|C1].

:- mode member(+,+,-).
member(_,[],Result) :- true,!,Result=no.
member(A,[A|_],Result) :- true,!,Result=yes.
member(A,[_|B],Result)
    :- true,!,member(A,B,Result).

:- mode add(+,+,-).
add(A,B,C)
    :- prolog(integer(A)),prolog(integer(B)),!,
    C is A + B.

:- mode delete(+,+,-).
delete(_,[],C) :- true,!,C=[].
delete(A,[A|B],C) :- true,!,C=B.
delete(A,[D|B],C)
    :- true,!,delete(A,B,C1),C=[D|C1].

:- mode ascending_ordered_pushnew(+,+,-).
ascending_ordered_pushnew(A,B,C)
    :- true,!,
    member(A,B,Result),
    ascending_ordered_pushnew(Result,A,B,C).

:- mode mapcan(+,-).
mapcan([],L) :- true,!,L=[].
mapcan([L1|L2],L3)
    :- true,!,mapcan(L1,L2,L3).

:- mode rest(+,+,-).
rest(_,[],C) :- true,!,C=[].
rest(A,[A|B],C) :- true,!,C=B.
rest(A,[_|B],C) :- true,!,rest(A,B,C).

:- mode ordered_insert(+,+,-,+).
ordered_insert(Obj,[],L2,_)
    :- true,!,L2=[Obj].
ordered_insert(Obj,L1,L2,D)
    :- true,!,
    get_slot(Obj,index,V,D),
    ordered_insert(Obj,V,L1,L2,D).

:- mode prolog(+).
prolog(Goal) :- Goal,!.

:- mode gensym(+,+,-).
gensym(A,B,C)
    :- prolog(integer(B)),!,
    prolog(name(A,AL)),prolog(name(B,BL)),
    append(AL,BL,CL),prolog(name(C,CL)).
```

```
%%% local predicates

:- mode get_slot(+,+,-).
get_slot([(Slot,V1)|_],Slot,V) :- true,!,V=V1.
get_slot([Slot1|Slots],Slot,V) :-
%%  Slot1\== (Slot,_),
    true,!,
    get_slot(Slots,Slot,V).
get_slot([],_,V) :- true,!,V='undef'.

:- mode set_slot(+,+,+,-).
set_slot([(Slot,_)|Slots],Slot,V,New_Slots) :-
    true,!, New_Slots=[(Slot,V)|Slots].
set_slot([Slot1|Slots],Slot,V,New_Slots) :-
%%  Slot1\==(Slot,_),
    true,!,
    New_Slots=[Slot1|New_Slots1],
    set_slot(Slots,Slot,V,New_Slots1).
set_slot([],Slot,V,New_Slot) :-
    true,!,New_Slot=[(Slot,V)].

:- mode ascending_ordered_pushnew(+,+,+,-).
ascending_ordered_pushnew(yes,_,B,C) :- true,!,C=B.
ascending_ordered_pushnew(no,A,B,C)
    :- true,!,
    ascending_ordered_pushnew_sub1(A,B,C).

:- mode ascending_ordered_pushnew_sub1(+,+,-).
ascending_ordered_pushnew_sub1(A,[],C) :- true,!,C=[A].
ascending_ordered_pushnew_sub1(A,[B|L1],L2)
    :- A < B,!,L2=[A,B|L1].
ascending_ordered_pushnew_sub1(A,[B|L1],L2)
    :- A >= B,!,
    L2=[B|L3],ascending_ordered_pushnew_sub1(A,L1,L3).

:- mode mapcan(+,+,-).
mapcan([],L2,L3)
    :- true,!,mapcan(L2,L3).
mapcan([H|T],L2,L3)
    :- true,!,L3=[H|L],mapcan(T,L2,L).

:- mode ordered_insert(+,+,+,-,+).
ordered_insert(Obj1,_,[],L2,_)
    :- true,!,L2=[Obj1].
ordered_insert(Obj1,V1,[Obj2|L1],L2,D)
    :- true,!,
    get_slot(Obj2,index,V2,D),
    ordered_insert(V1,V2,Obj1,Obj2,L1,L2,D).

:- mode ordered_insert(+,+,+,+,+,-,+).
ordered_insert(V1,V2,Obj1,Obj2,L1,L2,D)
    :- V1 < V2,!,L2=[Obj1,Obj2|L1].
ordered_insert(V1,V2,Obj1,Obj2,L1,L2,D)
    :- V1 >= V2,!,
    L2=[Obj2|L],
    ordered_insert(Obj1,V1,L1,L,D).

:- mode clas_obj(+,+,-).
class_obj([],_,Result)
    :- true,!,Result=yes.
class_obj([C|L],[C|L1],Result)
    :- true,!,class_obj(L,L1,Result).
class_obj(_,_,Result)
    :- true,!,Result=no.

:- mode add_with_index(+,+,+,-).
```

```
add_with_index(I,V,[],NIV) :-
    true,!,NIV=[(I,[V])].
add_with_index(I,V,[(I,Vs)|IVs],NIVs) :-
    true,!,NIVs=[(I,[V|Vs])|IVs].
add_with_index(I,V,[IV|IVs],NIVs) :-
%%  IV\=(I,_),
    true,!,
    NIVs=[IV|NIVs1],add_with_index(I,V,IVs,NIVs1).

:- mode get_with_index(+,-,+).
get_with_index(_,V,[]) :- true,!,V=[].
get_with_index(I,V,[(I,V1)|_]) :-
    true,!,V=V1.
get_with_index(I,V,[IV|IVs]) :-
%%  IV\=(I,_),
    true,!,get_with_index(I,V,IVs).

:- mode remove_with_index(+,+,+,-).
remove_with_index(_,_,[],NIV) :- true,!,NIV=[].
remove_with_index(I,V,[(I,Vs)|IVs],NIVs) :-
    true,!,
    delete(V,Vs,NVs),NIVs=[(I,NVs)|IVs].
remove_with_index(I,V,[IV|IVs],NIVs) :-
%%  IV\=(I,_),
    true,!,
    NIVs=[IV|NIVs1],
    remove_with_index(I,V,IVs,NIVs1).
```

```
%%% (2) node.prl

:- mode make_node(+,-,+,-).
make_node(Datum,Node,D0,D)
    :- true,!,
    incf(node,counter,D0,D1),
    get_slot(node,counter,Count,D1),
    gensym(node,Count,Node),
    new(Node,D1,D2),
    make_node_sub1(Datum,Node,Count,D2,D).

:- mode node_updating(+,-,+,-).
node_updating(Node,QuL,D0,D)
    :- true,!,
    get_slot(node,node_queue,Node_Queue,D0),
    Queue=[Node|Node_Queue],
    node_updating_sub1(Queue,QuL,D0,D).
:- mode label_updating(+,-,+,-).
label_updating(Node,Result,D0,D)
    :- true,!,
    get_slot(Node,label,Label1,D0),
    get_slot(environment,empty_env,Label2,D0),
    label_updating_sub1(Label1,Label2,Node,Result,D0,D).


%%% local predicates

:- mode make_node_sub1(+,+,+,+,-).
make_node_sub1(Datum,Node,Count,D0,D)
    :- true,!,
    set_slot(Node,index,Count,D0,D1),
    set_slot(Node,datum,Datum,D1,D2),
    set_slot(Node,status,in,D2,D3),
    set_slot(Node,label,[],D3,D4),
```

```
    set_slot(Node,c_justs,[],D4,D5),
    set_slot(Node,a_justs,[],D5,D6),
    set_slot(Node,assumption,[],D6,D7),
    add_slot(node,nodes,Node,D7,D).

:- mode node_updating_sub1(+,-,+,-).
node_updating_sub1([],QuL,D0,D)
    :- true,!, QuL=0,D0=D.
node_updating_sub1([Node|Q],QuL,D0,D)
    :- true,!,
    label_updating(Node,Res,D0,D1),
    node_updating_sub2(Res,Node,Q,N_Q,D1,D2),
    node_updating_sub1(N_Q,QuL1,D2,D),
    add(QuL1,1,QuL).

:- mode node_updating_sub2(+,+,+,-,+,-).
node_updating_sub2(no,_,Q,N_Q,D0,D)
    :- true,!, N_Q=Q,D0=D.
node_updating_sub2(yes,Node,Q,N_Q,D0,D)
    :- true,!,
    get_slot(Node,a_justs,Justs,D0),
    get_slot(node,contra_node,Contra,D0),
    node_updating_sub3(Justs,Nodes,Contra,D0,D),
    append(Q,Nodes,N_Q).

:- mode node_updating_sub3(+,-,+,+,-).
node_updating_sub3([],Nodes,_,D0,D)
    :- true,!, Nodes=[],D0=D.
node_updating_sub3([Jus|Justs],Nodes,Contra,D0,D)
    :- true,!,
    get_slot(Jus,consequence,Conse,D0),
    node_updating_sub4(Jus,Conse,Contra,Nodes,Nodes1,D0,D1),
    node_updating_sub3(Justs,Nodes1,Contra,D1,D).

:- mode node_updating_sub4(+,+,+,?,?,+,-).
node_updating_sub4(Jus,Conse,Contra,Nodes,Nodes1,D0,D)
    :- Conse=Contra,!,
    Nodes=Nodes1,
    nogood_updating(Jus,D0,D).
node_updating_sub4(_,Conse,Contra,Nodes,Nodes1,D0,D)
    :- Conse\==Contra,!,
    Nodes=[Conse|Nodes1],D0=D.


:- mode label_updating_sub1(+,+,+,-,+,-).
label_updating_sub1(Label1,Label2,Node,Result,D0,D)
    :- Label1=[Label2],!,
    get_slot(Label2,nodes,Nodes,D0),
    rest(Node,Nodes,Mem),
    label_updating_sub2(Label,Node,Mem,Result,D0,D).
label_updating_sub1(Label1,Label2,Node,Result,D0,D)
    :- Label1\==[Label2],!,
    label_updating_sub3(Node,Result,D0,D).

:- mode label_updating_sub2(+,+,+,-,+,-).
label_updating_sub2(Label,Node,Member,Result,D0,D)
    :- Member=[],!,
    Result=no,
    add_slot(Label,nodes,Node,D0,D).
label_updating_sub2(_,Node,Member,Result,D0,D)
    :- Member\==[],!,
    label_updating_sub3(Node,Result,D0,D).

:- mode label_updating_sub3(+,-,+,-).
label_updating_sub3(Node,Result,D0,D)
    :- true,!,
```

```
    get_slot(Node,c_justs,Justs,D0),
    label_updating_sub4(Justs,[],Nenvs,D0,D1),
    label_updating_sub5(Node,Nenvs,Result,D1,D).

:- mode label_updating_sub4(+,+,-,+,-).
label_updating_sub4([],Envs,Nenvs,D0,D)
    :- true,!, Nenvs=Envs,D0=D.
label_updating_sub4([Jus|Justs],Oenvs,Nenvs,D0,D)
    :- true,!,
    get_envs_from_just(Jus,Envs,D0,D1),
    label_updating_sub6(Envs,Oenvs,Nenvs1,D1,D2),
    label_updating_sub4(Justs,Nenvs1,Nenvs,D2,D).

:- mode label_updating_sub6(+,+,-,+,-).
label_updating_sub6([],Oenvs,Nenvs,D0,D)
    :- true,!, Nenvs=Oenvs,D0=D.
label_updating_sub6([Penv|Envs],Oenvs,Nenvs,D0,D)
    :- true,!,
    subsumed_check(Oenvs,Penv,Res,D0),
    label_updating_sub7(Res,Penv,Envs,Oenvs,Nenvs,D0,D).

:- mode label_updating_sub7(+,+,+,+,-,+,-).
label_updating_sub7(yes,_,Envs,Oenvs,Nenvs,D0,D)
    :- true,!,
    label_updating_sub6(Envs,Oenvs,Nenvs,D0,D).
label_updating_sub7(no,Penv,Envs,Oenvs,Nenvs,D0,D)
    :- true,!,
    env_contradictory_p(Penv,Res,D0,D1),
    label_updating_sub8(Res,Penv,Envs,Oenvs,Nenvs,D1,D).

:- mode label_updating_sub8(+,+,+,+,-,+,-).
label_updating_sub8(yes,_,Envs,Oenvs,Nenvs,D0,D)
    :- true,!,
    label_updating_sub6(Envs,Oenvs,Nenvs,D0,D).
label_updating_sub8(no,Penv,Envs,Oenvs,Nenvs,D0,D)
    :- true,!,
    label_updating_sub9(Oenvs,Penv,Nenvs1,D0),
    ordered_insert(Penv,Nenvs1,Nenvs2,D0),
    label_updating_sub6(Envs,Nenvs2,Nenvs,D0,D).

:- mode label_updating_sub9(+,+,-,+).
label_updating_sub9([],_,Nenvs,_)
    :- true,!, Nenvs=[].
label_updating_sub9([Oenv|Oenvs],Penv,Nenvs,D)
    :- true,!,
    env_subsumed_p(Oenv,Penv,Res,D),
    label_updating_sub10(Res,Penv,Oenv,Oenvs,Nenvs,D).

:- mode label_updating_sub10(+,+,+,+,-,+).
label_updating_sub10(yes,Penv,_,Oenvs,Nenvs,D)
    :- true,!,
    label_updating_sub9(Oenvs,Penv,Nenvs,D).
label_updating_sub10(no,Penv,Oenv,Oenvs,Nenvs,D)
    :- true,!,
    Nenvs=[Oenv|Nenvs1],
    label_updating_sub9(Oenvs,Penv,Nenvs1,D).

:- mode label_updating_sub5(+,+,-,+,-).
label_updating_sub5(Node,Nenvs,Result,D0,D)
    :- true,!,
    get_slot(Node,label,Label,D0),
    label_updating_sub11(Node,Label,Nenvs,Result,D0,D).

:- mode label_updating_sub11(+,+,+,-,+,-).
label_updating_sub11(_,Label,Nenvs,Result,D0,D)
    :- Nenvs=Label,!,Result=no,D0=D.
```

```
label_updating_sub11(Node,Label,Nenvs,Result,D0,D)
    :- Nenvs\==Label,!,
    Result=yes,
    set_slot(Node,label,Nenvs,D0,D1),
    label_updating_sub12(Node,Label,Nenvs,D1,D2),
    label_updating_sub13(Node,Nenvs,Label,D2,D3),
    label_updating_sub14(Node,Nenvs,D3,D).

:- mode label_updating_sub12(+,+,+,+,-).
label_updating_sub12(_,[],_,D0,D)
    :- true,!, D0=D.
label_updating_sub12(Node,[Oenv|Label],Nenvs,D0,D)
    :- true,!,
    member(Oenv,Nenvs,Res),
    label_updating_sub15(Res,Node,Oenv,Label,Nenvs,D0,D).

:- mode label_updating_sub15(+,+,+,+,+,+,-).
label_updating_sub15(yes,Node,Oenv,Label,Nenvs,D0,D)
    :- true,!,
    label_updating_sub12(Node,Label,Nenvs,D0,D).
label_updating_sub15(no,Node,Oenv,Label,Nenvs,D0,D)
    :- true,!,
    get_slot(Oenv,nodes,Nodes,D0),
    set_slot(Oenv,nodes,Nodes1,D0,D1),
    delete(Node,Nodes,Nodes1),
    label_updating_sub12(Node,Label,Nenvs,D1,D).

:- mode label_updating_sub13(+,+,+,+,-).
label_updating_sub13(_,[],_,D0,D)
    :- true,!, D0=D.
label_updating_sub13(Node,[Nenv|Nenvs],Label,D0,D)
    :- true,!,
    member(Nenv,Label,Res),
    label_updating_sub16(Res,Node,Nenv,Nenvs,Label,D0,D).

:- mode label_updating_sub16(+,+,+,+,+,+,-).
label_updating_sub16(yes,Node,_,Nenvs,Label,D0,D)
    :- true,!,
    label_updating_sub13(Node,Nenvs,Label,D0,D).
label_updating_sub16(no,Node,Nenv,Nenvs,Label,D0,D)
    :- true,!,
    add_slot(Nenv,nodes,Node,D0,D1),
    label_updating_sub13(Node,Nenvs,Label,D1,D).

:- mode label_updating_sub14(+,+,+,-).
label_updating_sub14(Node,Nenvs,D0,D)
    :- Nenvs=[],!,set_slot(Node,status,out,D0,D).
label_updating_sub14(Node,Nenvs,D0,D)
    :- Nenvs\==[],!,set_slot(Node,status,in,D0,D).
```

```
%%% (3) justif.prl

:- mode make_justification(+,+,+,-,+,-).
make_justification(Type,Conseq,Antecs,Jus,D0,D)
    :- true,!,
    incf(justification,counter,D0,D1),
    get_slot(justification,counter,Count,D1),
    gensym(jus,Count,Jus),
    new(Jus,D1,D2),
    make_just_sub1(Type,Conseq,Antecs,Jus,Count,D2,D3),
    make_just_sub2(Antecs,Conseq,Jus,D3,D).
```

```
:- mode get_envs_from_just(+,-,+,-).
get_envs_from_just(Jus,Envs,D0,D)
    :- true,!,
    get_slot(Jus,antecedents,Antecs,D0),
    get_slot(environment,empty_env,Eenv,D0),
    get_envs_sub1(Antecs,Iasss,Ienvs,D0),
    get_env_from_asss(Iasss,Eenv,Benv,D0,D1),
    get_envs_sub3(Benv,Ienvs,Envs,D1,D).


%%% local predicates

:- mode make_just_sub1(+,+,+,+,+,+,-).
make_just_sub1(Type,Conseq,Antecs,Jus,Count,D0,D)
    :- true,!,
    set_slot(Jus,index,Count,D0,D1),
    set_slot(Jus,type,Type,D1,D2),
    set_slot(Jus,consequence,Conseq,D2,D3),
    set_slot(Jus,antecedents,Antecs,D3,D4),
    add_slot(Conseq,c_justs,Jus,D4,D5),
    add_slot(justification,justifications,Jus,D5,D).

:- mode make_just_sub2(+,+,+,+,-).
make_just_sub2(Antecs,Conseq,Jus,D0,D)
    :- true,!,
    make_just_sub3(Antecs,Jus,D0,D1),
    make_just_sub5(Conseq,Jus,D1,D).

:- mode make_just_sub3(+,+,+,-).
make_just_sub3([],_,D0,D)
    :- true,!, D0=D.
make_just_sub3([Antec|Antecs],Jus,D0,D)
    :- true,!,
    class_object(node,Antec,Result),
    make_just_sub4(Result,Antec,Antecs,Jus,D0,D).

:- mode make_just_sub4(+,+,+,+,+,-).
make_just_sub4(yes,Antec,Antecs,Jus,D0,D)
    :- true,!,
    add_slot(Antec,a_justs,Jus,D0,D1),
    make_just_sub3(Antecs,Jus,D1,D).
make_just_sub4(no,_,Antecs,Jus,D0,D)
    :- true,!,
    make_just_sub3(Antecs,Jus,D0,D).

:- mode make_just_sub5(+,+,+,-).
make_just_sub5(Conse,Jus,D0,D)
    :- true,!,
    get_slot(node,contra_node,Contra_node,D0),
    make_just_sub6(Conse,Contra_node,Jus,D0,D).

:- mode make_just_sub6(+,+,+,+,-).
make_just_sub6(Conse,Contra_node,Jus,D0,D)
    :- Conse==Contra_node,!,
    nogood_updating(Jus,D0,D).
make_just_sub6(Conse,Contra_node,_,D0,D)
    :- Conse\==Contra_node,!,
    node_updating(Conse,_,D0,D).


:- mode get_envs_sub1(+,-,-,+).
get_envs_sub1([],Iasss,Ienvs,_)
    :- true,!, Iasss=[],Ienvs=[].
get_envs_sub1([Antec|Antecs],Iasss,Ienvs,D)
    :- true,!,
    class_object(ass,Antec,Result),
```

```
    get_envs_sub2(Result,Antec,Antecs,Iasss,Ienvs,D).

:- mode get_envs_sub2(+,+,+,-,-,+).
get_envs_sub2(yes,Antec,Antecs,Iasss,Ienvs,D)
    :- true,!,
    Iasss=[Antec|Iasssl],
    get_envs_subl(Antecs,Iasssl,Ienvs,D).
get_envs_sub2(no,Antec,Antecs,Iasss,Ienvs,D)
    :- true,!,
    get_slot(Antec,label,Label,D),
    Ienvs=[Label|Ienvsl],
    get_envs_subl(Antecs,Iasss,Ienvsl,D).

:- mode get_envs_sub3(+,+,-,+,-).
get_envs_sub3(Benv,_,Envs,D0,D)
    :- Benv=[],!,Envs=[],D=D0.
get_envs_sub3(Benv,Ienvs,Envs,D0,D)
    :- Benv\==[],!,
    get_new_envs(Benv,Ienvs,Envs,D0,D).



%%% (4) assump.prl

:- mode make_assumption(+,-,+,-).
make_assumption(Node,Ass,D0,D)
    :- true,!,
    incf(assumption,counter,D0,D1),
    get_slot(assumption,counter,Count,D1),
    gensym(ass,Count,Ass),
    new(Ass,D1,D2),
    set_slot(Ass,index,Count,D2,D3),
    set_slot(Ass,node,Node,D3,D4),
    set_slot(Ass,environments,[],D4,D5),
    add_slot(Node,assumption,Ass,D5,D6),
    add_slot(assumption,assumptions,Ass,D6,D).

:- mode get_envs_from_asss(+,+,-,+,-).
get_env_from_asss([],Benv,Nenv,D0,D)
    :- true,!,Nenv=Benv,D0=D.
get_env_from_asss([Ass|Asss],Env,Nenv,D0,D)
    :- true,!,
    push_env(Ass,Env,Tenv,D0,D1),
    get_env_subl(Tenv,Asss,Nenv,D1,D).


%%% local predicates

:- mode get_env_subl(+,+,-,+,-).
get_env_subl(Tenv,_,Nenv,D0,D)
    :- Tenv=[],!,Nenv=[],D0=D.
get_env_subl(Tenv,Asss,Nenv,D0,D)
    :- Tenv\==[],!,
    get_env_from_asss(Asss,Tenv,Nenv,D0,D).



%%% (5) enviro.prl

:- mode make_environment(+,-,+,-).
make_environment(Asss,Env,D0,D)
    :- true,!,
```

```
    incf(environment,counter,D0,D1),
    get_slot(environment,counter,Count,D1),
    gensym(env,Count,Env),
    new(Env,D1,D2),
    length(Asss,Size),
    make_env_sub1(Asss,Env,Count,Size,D2,D3),
    make_env_sub2(Asss,Env,D3,D4),
    push_env_table(Env,D4,D).

:- mode env_subsumed_p(+,+,-,+).
env_subsumed_p(E1,E2,Result,D)
    :- true,!,
    get_slot(E1,ass_size,N1,D),
    get_slot(E2,ass_size,N2,D),
    env_subsumed_p_sub1(N1,N2,E1,E2,Result,D).

:- mode subsumed_check(+,+,-,+).
subsumed_check([],_,Result,_)
    :- true,!, Result=no.
subsumed_check([E|Envs],Penv,Result,D)
    :- true,!,
    env_subsumed_p(Penv,E,Res,D),
    subsumed_check_sub1(Res,Envs,Penv,Result,D).

:- mode get_new_envs(+,+,-,+,-).
get_new_envs(Base_env,Input_envs,Envs,D0,D)
    :- true,!,
    get_new_envs_sub1(Base_env,Input_envs,[],Envs,D0,D).

:- mode union_envs(+,+,-,+,-).
union_envs(E1,E2,Env,D0,D)
    :- true,!,
    get_slot(E1,ass_size,N1,D0),
    get_slot(E2,ass_size,N2,D0),
    union_envs_sub1(N1,N2,E1,E2,Env,D0,D).

:- mode push_env(+,+,-,+,-).
push_env(Ass,Env,Nenv,D0,D)
    :- true,!,
    get_slot(Env,assumptions,Asss,D0),
    ordered_insert(Ass,Asss,Nasss,D0),
    lookup_env(Nasss,Oenv,D0),
    push_env_sub1(Oenv,Nasss,Nenv,D0,D).

:- mode push_env_table(+,+,-).
push_env_table(Env,D0,D)
    :- true,!,
    get_slot(Env,ass_size,Index,D0),
    get_slot(environment,indices,Indices,D0),
    add_with_index(environment,env_table,Index,Env,D0,D1),
    ascending_ordered_pushnew(Index,Indices,Nindices),
    set_slot(environment,indices,Nindices,D1,D).

:- mode lookup_env(+,-,+).
lookup_env(Asss,Env,D)
    :- true,!,
    length(Asss,Index),
    get_with_index(environment,env_table,Index,Entry,D),
    lookup_env_sub1(Entry,Asss,Env,D).

%% check contradiction
:- mode env_contradictory_p(+,-,+,-).
env_contradictory_p(Env,Result,D0,D)
    :- true,!,
    get_slot(Env,contradictory,Contra,D0),
    env_contradictory_p_sub1(Contra,Env,Result,D0,D).
```

```
:- mode remove_env_from_labels(+,+,-).
remove_env_from_labels(Env,D0,D)
    :- true,!,
    get_slot(Env,nodes,Nodes,D0),
    remove_env_sub1(Nodes,Env,D0,D).


%%% local predicates

:- mode make_env_sub1(+,+,+,+,+,-).
make_env_sub1(Asss,Env,Count,Size,D0,D)
    :- true,!,
    set_slot(Env,index,Count,D0,D1),
    set_slot(Env,ass_size,Size,D1,D2),
    set_slot(Env,assumptions,Asss,D2,D3),
    set_slot(Env,nodes,[],D3,D4),
    set_slot(Env,contradictory,[],D4,D5),
    add_slot(environment,environments,Env,D5,D).

:- mode make_env_sub2(+,+,+,-).
make_env_sub2([],_,D0,D)
    :- true,!, D0=D.
make_env_sub2([Ass|Asss],Env,D0,D)
    :- true,!,
    add_slot(Ass,environments,Env,D0,D1),
    make_env_sub2(Asss,Env,D1,D).

:- mode env_subsumed_p_sub1(+,+,+,+,-,+).
env_subsumed_p_sub1(N1,N2,_,_,Result,_)
    :- N1 < N2 ,!, Result=no.
env_subsumed_p_sub1(N1,N2,E1,E2,Result,D)
    :- N1 >= N2 ,!,
    get_slot(E1,assumptions,A1,D),
    get_slot(E2,assumptions,A2,D),
    env_subsumed_p_sub2(A1,A2,Result,D).

:- mode env_subsumed_p_sub2(+,+,-,+).
env_subsumed_p_sub2(_,[],Result,_)
    :- true,!, Result=yes.
env_subsumed_p_sub2([],_,Result,_)
    :- true,!, Result=no.
env_subsumed_p_sub2([Car1|Cdr1],[Car2|Cdr2],Result,D)
    :- true,!,
    get_slot(Car1,index,N1,D),
    get_slot(Car2,index,N2,D),
    env_subsumed_p_sub3(N1,N2,Car2,Cdr1,Cdr2,Result,D).

:- mode env_subsumed_p_sub3(+,+,+,+,+,-,+).
env_subsumed_p_sub3(N1,N2,_,_,_,Result,_)
    :- N1 > N2 ,!, Result=no.
env_subsumed_p_sub3(N1,N2,Car2,A1,A2,Result,D)
    :- N1 < N2 ,!,
    env_subsumed_p_sub2(A1,[Car2|A2],Result,D).
env_subsumed_p_sub3(N1,N2,_,   A1,A2,Result,D)
    :- N1 = N2 ,!,
    env_subsumed_p_sub2(A1,A2,Result,D).

:- mode subsumed_check_sub1(+,+,+,-,+).
subsumed_check_sub1(yes,_,_,Result,_)
    :- true,!, Result=yes.
subsumed_check_sub1(no,Envs,Penv,Result,D)
    :- true,!,
    subsumed_check(Envs,Penv,Result,D).

:- mode get_new_envs_sub1(+,+,+,-,+,-).
```

```
get_new_envs_sub1(Benv,Ienvs,Envs1,Envs,D0,D)
    :- Ienvs=[] ,!,
    D=D0,
    subsumed_check(Envs1,Benv,Res,D0),
    get_new_envs_sub2(Res,Benv,Envs1,Envs,D0).
get_new_envs_sub1(Benv,Ienvs,Envs1,Envs,D0,D)
    :- Ienvs=[Choices|Ienvs1] ,!,
    get_new_envs_sub5(Choices,Benv,Ienvs1,Envs1,Envs,D0,D).

:- mode get_new_envs_sub2(+,+,+,-,+).
get_new_envs_sub2(yes,_,Envs1,Envs,D)
    :- true,!, Envs=Envs1.
get_new_envs_sub2(no,Benv,Envs1,Envs,D)
    :- true,!,
    Envs=[Benv|Envs2],
    get_new_envs_sub3(Envs1,Benv,Envs2,D).

:- mode get_new_envs_sub3(+,+,-,+).
get_new_envs_sub3([],_,Envs,_)
    :- true,!, Envs=[].
get_new_envs_sub3([Env|Envs],Benv,Nenvs,D)
    :- true,!,
    env_subsumed_p(Env,Benv,Res,D),
    get_new_envs_sub4(Res,Env,Envs,Benv,Nenvs,D).

:- mode get_new_envs_sub4(+,+,+,+,-,+).
get_new_envs_su4(yes,Env,Envs,Benv,Nenvs,D)
    :- true,!,
    get_new_envs_sub3(Envs,Benv,Nenvs,D).
get_new_envs_sub4(no,Env,Envs,Benv,Nenvs,D)
    :- true,!,
    Nenvs=[Env|Nenvs1],
    get_new_envs_sub3(Envs,Benv,Nenvs1,D).

:- mode get_new_envs_sub5(+,+,+,+,-,+,-).
get_new_envs_sub5([],_,_,Envs1,Envs,D0,D)
    :- true,!, Envs=Envs1,D0=D.
get_new_envs_sub5([Env|Cenvs],Benv,Ienvs,Envs1,Envs,D0,D)
    :- true,!,
    union_envs(Env,Benv,Nenv,D0,D1),
    get_new_envs_sub6(Nenv,Cenvs,Benv,Ienvs,Envs1,Envs,D1,D).

:- mode get_new_envs_sub6(+,+,+,+,+,-,+,-).
get_new_envs_sub6(Nenv,Cenvs,Benv,Ienvs,Envs1,Envs,D0,D)
    :- Nenv=[] ,!,
    get_new_envs_sub5(Cenvs,Benv,Ienvs,Envs1,Envs,D0,D).
get_new_envs_sub6(Nenv,Cenvs,Benv,Ienvs,Envs1,Envs,D0,D)
    :- Nenv\==[],!,
    get_new_envs_sub1(Nenv,Ienvs,Envs1,Envs2,D0,D1),
    get_new_envs_sub5(Cenvs,Benv,Ienvs,Envs2,Envs,D1,D).

:- mode union_envs_sub1(+,+,+,+,-,+,-).
union_envs_sub1(N1,N2,E1,E2,Env,D0,D)
    :- N1 =< N2 ,!,
    get_slot(E1,assumptions,Asss,D0),
    get_env_from_asss(Asss,E2,Env,D0,D).
union_envs_sub1(N1,N2,E1,E2,Env,D0,D)
    :- N1 > N2  ,!,
    get_slot(E2,assumptions,Asss,D0),
    get_env_from_asss(Asss,E1,Env,D0,D).

:- mode push_env_sub1(+,+,-,+,-).
push_env_sub1(Oenv,Nasss,Nenv,D0,D)
    :- Oenv=[] ,!,
    make_environment(Nasss,Oenv1,D0,D1),
    env_contradictory_p(Oenv1,Result,D1,D),
```

```
    push_env_sub2(Result,Oenv1,Nenv).
push_env_sub1(Oenv,_,Nenv,D0,D)
    :- Oenv\==[],!,
    env_contradictory_p(Oenv,Result,D0,D),
    push_env_sub2(Result,Oenv,Nenv).

:- mode push_env_sub2(+,+,-).
push_env_sub2(yes,_,Nenv) :- true,!, Nenv=[].
push_env_sub2(no,Oenv,Nenv) :- true,!, Nenv=Oenv.

:- mode lookup_env_sub1(+,+,-,+).
lookup_env_sub1([],_,Env,_)
    :- true,!, Env=[].
lookup_env_sub1([E|Entry],Asss,Env,D)
    :- true,!,
    get_slot(E,assumptions,Asss1,D),
    lookup_env_sub2(Asss,Asss1,E,Entry,Env,D).

:- mode lookup_env_sub2(+,+,+,+,-,+).
lookup_env_sub2(Asss,Asss1,E,_,Env,_)
    :- Asss=Asss1 ,!, Env=E.
lookup_env_sub2(Asss,Asss1,_,Entry,Env,D)
    :- Asss\==Asss1,!,
    lookup_env_sub1(Entry,Asss,Env,D).

:- mode env_contradictory_p_sub1(+,+,-,+,-).
env_contradictory_p_sub1(Contra,E,Result,D0,D)
    :- Contra=[] ,!,
    get_slot(E,ass_size,Index,D0),
    get_slot(nogood,indices,Indices,D0),
    env_contradictory_p_sub2(Indices,Index,E,Result,D0,D).
env_contradictory_p_sub1(Contra,_,Result,D0,D)
    :- Contra\==[],!,Result=yes,D=D0.

:- mode env_contradictory_p_sub2(+,+,+,-,+,-).
env_contradictory_p_sub2([],_,_,Res,D0,D)
    :- true,!, Res=no,D0=D.
env_contradictory_p_sub2([Tindex|_],Index,_,Res,D0,D)
    :- Tindex > Index,!,Res=no,D0=D.
env_contradictory_p_sub2([Tindex|Indices],Index,E,Res,D0,D)
    :- Tindex =< Index ,!,
    get_with_index(nogood,nogood_table,Tindex,Entry,D0),
    env_contradictory_p_sub3(Entry,E,Res1,D0,D1),
    env_contradictory_p_sub4(Res1,Indices,Index,E,Res,D1,D).

:- mode env_contradictory_p_sub3(+,+,-,+,-).
env_contradictory_p_sub3([],_,Result,D0,D)
    :- true,!, Result=no,D0=D.
env_contradictory_p_sub3([Cenv|Entry],Env,Result,D0,D)
    :- true,!,
    env_subsumed_p(Env,Cenv,Res,D0),
    env_contradictory_p_sub5(Res,Cenv,Entry,Env,Result,D0,D).

:- mode env_contradictory_p_sub5(+,+,+,+,-,+,-).
env_contradictory_p_sub5(yes,Cenv,_,E,Result,D0,D)
    :- true,!,
    Result=yes,
    set_slot(E,contradictory,Cenv,D0,D).
env_contradictory_p_sub5(no,_,Entry,E,Result,D0,D)
    :- true,!,
    env_contradictory_p_sub3(Entry,E,Result,D0,D).

:- mode env_contradictory_p_sub4(+,+,+,+,-,+,-).
env_contradictory_p_sub4(yes,_,_,_,Result,D0,D)
    :- true,!, Result=yes,D0=D.
env_contradictory_p_sub4(no,Indices,Index,Env,Result,D0,D)
```

```
    :- true,!,
    env_contradictory_p_sub2(Indices,Index,Env,Result,D0,D).

:- mode remove_env_sub1(+,+,+,-).
remove_env_sub1([],_,D0,D)
    :- true,!, D0=D.
remove_env_sub1([Node|Nodes],Env,D0,D)
    :- true,!,
    get_slot(Node,label,Label,D0),
    delete(Env,Label,Nlabel),
    set_slot(Node,label,Nlabel,D0,D1),
    remove_env_sub2(Nlabel,Node,Nodes,Env,D1,D).

:- mode remove_env_sub2(+,+,+,+,+,-).
remove_env_sub2(Nlabel,Node,Nodes,Env,D0,D)
    :- Nlabel=[] ,!,
    set_slot(Node,status,out,D0,D1),
    remove_env_sub1(Nodes,Env,D1,D).
remove_env_sub2(Nlabel,_,Nodes,Env,D0,D)
    :- Nlabel\==[],!,
    remove_env_sub1(Nodes,Env,D0,D).



%%% (6) nogood.prl

:- mode nogood_updating(+,+,-).
nogood_updating(Jus,D0,D)
    :- true,!,
    get_envs_from_just(Jus,Envs,D0,D1),
    nogood_updating(Envs,Jus,D1,D).

:- mode push_nogood_table(+,+,-).
push_nogood_table(Env,D0,D)
    :- true,!,
    get_slot(Env,ass_size,Index,D0),
    get_slot(nogood,indices,Indices,D0),
    add_with_index(nogood,nogood_table,Index,Env,D0,D1),
    ascending_ordered_pushnew(Index,Indices,Nindices),
    set_slot(nogood,indices,Nindices,D1,D).


%%% local predicates

:- mode nogood_updating(+,+,+,-).
nogood_updating([],_,D0,D)
    :- true,!, D0=D.
nogood_updating([Cenv|Envs],Jus,D0,D)
    :- true,!,
    push_nogood_table(Cenv,D0,D1),
    remove_env_from_labels(Cenv,D1,D2),
    get_slot(environment,indices,Env_indices,D2),
    set_slot(Cenv,contradictory,Jus,D2,D3),
    get_slot(Cenv,ass_size,Index,D3),
    get_slot(nogood,indices,Nogood_indices,D3),
    nogood_updating_sub1(Nogood_indices,Index,Cenv,D3,D4),
    nogood_updating_sub3(Env_indices,Index,Cenv,D4,D5),
    nogood_updating(Envs,Jus,D5,D).

:- mode nogood_updating_sub1(+,+,+,+,-).
nogood_updating_sub1([],_,_,D0,D)
    :- true,!, D0=D.
nogood_updating_sub1([Tindex|Indices],Index,Cenv,D0,D)
    :- Tindex =< Index ,!,
```

```
    nogood_updating_sub1(Indices,Index,Cenv,D0,D).
nogood_updating_sub1([Tindex|Indices],Index,Cenv,D0,D)
    :- Tindex > Index ,!,
    get_with_index(nogood,nogood_table,Tindex,Entry,D0),
    nogood_updating_sub2(Entry,Tindex,Cenv,D0,D1),
    nogood_updating_sub1(Indices,Index,Cenv,D1,D).

:- mode nogood_updating_sub2(+,+,+,+,-).
nogood_updating_sub2([],_,_,D0,D)
    :- true,!, D0=D.
nogood_updating_sub2([Old|Entry],Tindex,Cenv,D0,D)
    :- true,!,
    env_subsumed_p(Old,Cenv,Res,D0),
    nogood_updating_sub4(Res,Old,Entry,Tindex,Cenv,D0,D).

:- mode nogood_updating_sub4(+,+,+,+,+,+,-).
nogood_updating_sub4(yes,Old,Entry,Tindex,Cenv,D0,D)
    :- true,!,
    remove_with_index(nogood,nogood_table,Tindex,Old,D0,D1),
    nogood_updating_sub2(Entry,Tindex,Cenv,D1,D).
nogood_udpating_sub4(no,_,Entry,Tindex,Cenv,D0,D)
    :- true,!,
    nogood_updating_sub2(Entry,Tindex,Cenv,D0,D).

:- mode nogood_updating_sub3(+,+,+,+,-).
nogood_updating_sub3([],_,_,D0,D)
    :- true,!, D0=D.
nogood_updating_sub3([Tindex|Indices],Index,Cenv,D0,D)
    :- Tindex =< Index ,!,
    nogood_updating_sub3(Indices,Index,Cenv,D0,D).
nogood_updating_sub3([Tindex|Indices],Index,Cenv,D0,D)
    :- Tindex > Index ,!,
    get_with_index(environment,env_table,Tindex,Entry,D0),
    nogood_updating_sub5(Entry,Cenv,D0,D),
    nogood_updating_sub3(Indices,Index,Cenv,D0,D).

:- mode nogood_updating_sub5(+,+,+,-).
nogood_updating_sub5([],_,D0,D)
    :- true,!, D0=D.
nogood_updating_sub5([Old|Entry],Cenv,D0,D)
    :- true,!,
    get_slot(Old,contradictory,Contra,D0),
    nogood_updating_sub6(Contra,Old,Entry,Cenv,D0,D).

:- mode nogood_updating_sub6(+,+,+,+,+,-).
nogood_updating_sub6(Contra,Old,Entry,Cenv,D0,D)
    :- Contra=[],!,
    env_subsumed_p(Old,Cenv,Result,D0),
    nogood_updating_sub7(Result,Old,Entry,Cenv,D0,D).
nogood_updating_sub6(Contra,_,Entry,Cenv,D0,D)
    :- Contra\==[],!,
    nogood_updating_sub5(Entry,Cenv,D0,D).

:- mode nogood_updating_sub7(+,+,+,+,+,-).
nogood_updating_sub7(yes,Old,Entry,Cenv,D0,D)
    :- true,!,
    set_slot(Old,contradictory,Cenv,D0,D1),
    remove_env_from_labels(Old,D1,D2),
    nogood_updating_sub5(Entry,Cenv,D2,D).
nogood_updating_sub7(no,_,Entry,Cenv,D0,D)
    :- true,!,
    nogood_updating_sub5(Entry,Cenv,D0,D).
```

```
%%% (7) interf.prl

:- mode assume_node(+,-,+,-).
assume_node(Node,J_obj,D0,D)
    :- true,!,
    set_slot(Node,assumption,[],D0,D1),
    make_assumption(Node,Ass_obj,D1,D2),
    make_justification(assume_node,Node,[Ass_obj],J_obj,D2,D).

:- mode premise_node(+,-,+,-).
premise_node(Node,Count,D0,D)
    :- true,!,
    get_slot(environment,empty_env,Empty_env,D0),
    set_slot(Node,label,[Empty_env],D0,D1),
    set_slot(Node,status,in,D1,D2),
    node_updating(Node,Count,D2,D).

:- mode nogood_nodes(+,-,+,-).
nogood_nodes(Nodes,J_obj,D0,D)
    :- true,!,
    get_slot(node,contra_node,Contra,D0),
    make_justification(nogood,Contra,Nodes,J_obj,D0,D).

:- mode implied_p(+,+,-,+).
implied_p(Node,Env,Result,D)
    :- true,!,
    get_slot(Node,label,Label,D),
    implied_p_sub1(Label,Env,Result,D).

:- mode consistency_p(+,+,-,+,-).
consistency_p(Node,Env,Result,D0,D)
    :- true,!,
    get_slot(Node,label,Label,D0),
    consistency_p_sub1(Label,Env,Result,D0,D).


%%% local predicates

:- mode implied_p_sub1(+,+,-,+).
implied_p_sub1([],_,Result,_)
    :- true,!, Result=no.
implied_p_sub1([Env1|Label],Env,Result,D)
    :- true,!,
    env_subsumed_p(Env,Env1,Res,D),
    implied_p_sub2(Res,Label,Env,Result,D).

:- mode implied_p_sub2(+,+,+,-,+).
implied_p_sub2(yes,_,_,Result,_)
    :- true,!, Result=yes.
implied_p_sub2(no,Label,Env,Result,D)
    :- true,!, implied_p_sub1(Label,Env,Result,D).

:- mode consistency_p(+,+,-,+,-).
consistency_p([],_,Result,D0,D)
    :- true,!, Result=no,D0=D.
consistency_p([Env1|Label],Env,Result,D0,D)
    :- true,!,
    union_envs(Env1,Env,Nenv,D0,D1),
    consistency_p_sub1(Nenv,Label,Env,Result,D1,D).

:- mode consistency_p_sub1(+,+,+,-,+,-).
consistency_p_sub1(Nenv,Label,Env,Result,D0,D)
    :- Nenv=[],!,
    consistency_p(Label,Env,Result,D0,D).
consistency_p_sub1(Nenv,Label,Env,Result,D0,D)
```

```
    :- Nenv\==[],!,
    env_contradictory_p(Nenv,Res,D0,D1),
    consistency_p_sub2(Res,Label,Env,Result,D1,D).

:- mode consistency_p_sub2(+,+,+,-,+,-).
consistency_p_sub2(yes,Label,Env,Result,D0,D)
    :- true,!,
    consistency_p(Label,Env,Result,D0,D).
consistency_p_sub2(no,_,_,Result,D0,D)
    :- true,!, Result=yes,D0=D.



%%% (8) init.prl

:- mode init(-).
init(D)
    :- true,!,
    init_node([],D0),
    init_justification(D0,D1),
    init_assumption(D1,D2),
    init_environment(D2,D3),
    init_nogood(D3,D).


%%% local predicates

%% initialize node class object
:- mode init_node(+,-).
init_node(D0,D)
    :- true,!,
    set_slot(node,nodes,[],D0,D1),
    set_slot(node,counter,0,D1,D2),
    set_slot(node,node_queue,[],D2,D3),
    make_node(contradiction,Contra_node,D3,D4),
    set_slot(node,contra_node,Contra_node,D4,D).

%% initialize justification class object
:- mode init_justification(+,-).
init_justification(D0,D)
    :- true,!,
    set_slot(justification,justifications,[],D0,D1),
    set_slot(justification,counter,0,D1,D).

%% initialize assumption class object
:- mode init_assumption(+,-).
init_assumption(D0,D)
    :- true,!,
    set_slot(assumption,assumptions,[],D0,D1),
    set_slot(assumption,counter,0,D1,D).

%% initiazlize environment class object
:- mode init_environment(+,-).
init_environment(D0,D)
    :- true,!,
    set_slot(environment,environments,[],D0,D1),
    set_slot(environment,counter,0,D1,D2),
    set_slot(environment,env_table,[],D2,D3),
    set_slot(environment,indices,[],D3,D4),
    make_environment([],Empty_env,D4,D5),
    set_slot(environment,empty_env,Empty_env,D5,D).

%% initialize nogood class object
:- mode init_nogood(+,-).
```

```
init_nogood(D0,D)
    :- true,!,
    set_slot(nogood,indices,[],D0,D1),
    set_slot(nogood,nogood_table,[],D1,D).
```

```
%%% (9) atms.prl

%% top level
:- public atms/1.
atms([Msg|MsgS])
    :- Msg=init,!,
    init(D),
    atms(MsgS,D).
atms([Msg|MsgS])
    :- Msg\==init,!,
    atms(MsgS).
atms([])
    :- true,!,true.

%% command interpreter
:- mode atms(?,+).

% termination
atms([],_)
    :- true,!,true.

% initialize ATMS
atms([init|MsgS],_)
    :- true,!,
    init(D),
    atms(MsgS,D).

% declare assumption-data
atms([assume(Datum)|MsgS],D0)
    :- true,!,
    get_node(Datum,Node,D0,D1),
    assume_node(Node,_,D1,D),
    atms(MsgS,D).

% declare premise-data
atms([premise(Datum)|MsgS],D0)
    :- true,!,
    get_node(Datum,Node,D0,D1),
    premise_node(Node,_,D1,D),
    atms(MsgS,D).

% declare contradictory data
atms([nogood(Data)|MsgS],D0)
    :- true,!,
    get_nodes(Data,Nodes,D0,D1),
    nogood_nodes(Nodes,_,D1,D),
    atms(MsgS,D).

% declare justification
atms([justify(Conse,Antes)|MsgS],D0)
    :- true,!,
    get_nodes([Conse|Antecs],[C_Node|A_Nodes],D0,D1),
    make_justification(user,C_Node,A_Nodes,_,D1,D),
    atms(MsgS,D).

% get status of specified datum
atms([status(Datum,Status)|MsgS],D)
```

```
    :- true,!,
    atms([datum_to_node(Datum,Node)],D),
    get_slot(Node,status,Status,D),
    atms(MsgS,D).

% whether specified datum is implied
% under specified environment
atms([implied_p(Datum,Env,Result)|MsgS],D)
    :- true,!,
    atms([datum_to_node(Datum,Node)],D),
    implied_p(Node,Env,Result,D),
    atms(MsgS,D).

% whether specified datum is contradictory
% to specified environment
atms([consistency_p(Datum,Env,Result)|MsgS],D0)
    :- true,!,
    atms([datum_to_node(Datum,Node)],D0),
    consistency_p(Node,Env,Result,D0,D),
    atms(MsgS,D).

% get justification of specified datum
atms([get_justifications(Datum,Justs)|MsgS],D)
    :- true,!,
    atms([datum_to_node(Datum,Node)],D),
    get_slot(Node,c_justs,Justs,D),
    atms(MsgS,D).

% get assumption of specified datum
atms([get_assumption(Datum,Ass)|MsgS],D)
    :- true,!,
    atms([datum_to_node(Datum,Node)],D),
    get_slot(Node,assumption,Asss,D),
    get_assumption(Asss,Ass),
    atms(MsgS,D).

% get node corresponding specified datum
atms([datum_to_node(Datum,Node)|MsgS],D)
    :- true,!,
    get_slot(node,nodes,Nodes,D),
    datum_to_node(Nodes,Datum,Node,D),
    atms(MsgS,D).

atms([get_solution(Strategy,Alt_Sets,Sol)|MsgS],D0)
    :- true,!,
    solution_subl(Alt_Sets,Alt_Nodes,D0,D1),
    solver(Strategy,Alt_Nodes,Sol,D1,D),
    atms(MsgS,D).

atms([print(What)|MsgS],D)
    :- true,!,
    print(What,D),
    atms(MsgS,D).


%%% local predicates

:- mode get_nodes(+,-,+,-).
get_nodes([],Nodes,D0,D)
    :- true,!, Nodes=[],D=D0.
get_nodes([Datum|Data],Nodes,D0,D)
    :- true,!,
    Nodes=[Node|Nodes1],
    get_node(Datum,Node,D0,D1),
    get_nodes(Data,Nodes1,D1,D).
```

```
:- mode get_node(+,-,+,-).
get_node(Datum,Node,D0,D)
    :- true,!,
    atms([datum_to_node(Datum,Node1)],D0),
    get_node(Datum,Node1,Node,D0,D).

:- mode get_node(+,+,-,+,-).
get_node(Datum,Node1,Node,D0,D)
    :- Node1=new,!,
    make_node(Datum,Node,D0,D).
get_node(_,Node1,Node,D0,D)
    :- Node1\==new,!,
    Node=Node1,D=D0.

:- mode datum_to_node(+,+,-,+).
datum_to_node([],_,Node,_)
    :- true,!, Node=new.
datum_to_node([Node1|Nodes],Datum,Node,D)
    :- true,!,
    get_slot(Node1,datum,Datum1,D),
    datum_to_node(Datum,Datum1,Node,Node1,Nodes,D).

:- mode datum_to_node(+,+,-,+,+,+).
datum_to_node(Datum,Datum1,Node,Node1,_,_)
    :- Datum=Datum1,!, Node=Node1.
datum_to_node(Datum,Datum1,Node,_,Nodes,D)
    :- Datum\==Datum1,!,
    datum_to_node(Nodes,Datum,Node,D).

:- mode get_assumption(+,-).
get_assumption([],Ass) :- true,!, Ass=[].
get_assumption([Ass_obj],Ass) :- true,!, Ass=Ass_obj.

:- mode solution_sub1(+,-,+,-).
solution_sub1([],N_Sets,D0,D)
    :- true,!, N_Sets=[],D=D0.
solution_sub1([Set|Sets],N_Sets,D0,D)
    :- true,!,
    get_nodes(Set,N_Set,D0,D1),
    N_Sets=[N_Set|N_Sets1],
    solution_sub1(Sets,N_Sets1,D1,D).
```

```
%%% (10) print.prl

:- mode printf(+).
printf([Msg|MsgS])
    :- true,!,
    prolog(Msg),
    printf(MsgS).
printf([])
    :- true,!,true.

:- mode print(+,+).
print(status(Node),D)
    :- true,!,
    get_slot(Node,status,Status,D),
    get_slot(Node,label,Label,D),
    printf([nl,write('      ----------------------------'),nl,
            write('              NODE - STATUS          '),nl,
            write('      ----------------------------'),nl,nl,
            write('      '),write(Node),write(' : '),write(Status),
            write(' under '),write(Label),nl,nl]).
```

```
print(statuses,D)
    :- true,!,
    get_slot(node,nodes,Nodes,D),
    statuses(Nodes,D).
print(node(Node),D)
    :- true,!,
    get_slot(Node,index,Index,D),
    get_slot(Node,datum,Datum,D),
    get_slot(Node,status,Status,D),
    get_slot(Node,label,Label,D),
    get_slot(Node,c_justs,C_justs,D),
    get_slot(Node,a_justs,A_justs,D),
    get_slot(Node,assumption,Asss,D),
    printf([nl,write('      ------------------------------'),nl,
                write('                NODE              '),nl,
                write('      ------------------------------'),nl,
                write('       node       : '),write(Node),    nl,
                write('       index      : '),write(Index),   nl,
                write('       datum      : '),write(Datum),   nl,
                write('       status     : '),write(Status),  nl,
                write('       label      : '),write(Label),   nl,
                write('       c_justs    : '),write(C_justs), nl,
                write('       a_justs    : '),write(A_justs), nl,
                write('       assumption : '),write(Asss), nl,nl]).
print(nodes,D)
    :- true,!,
    get_slot(node,nodes,Nodes,D),
    nodes(Nodes,D).
print(just(J),D)
    :- true,!,
    get_slot(J,index,Index,D),
    get_slot(J,type,Type,D),
    get_slot(J,antecedents,Antec,D),
    get_slot(J,consequence,Conse,D),
    printf([nl,write('      ------------------------------   '),nl,
                write('          JUSTIFICATION                '),nl,
                write('      ------------------------------   '),nl,
                write('       justification : '),write(J),       nl,
                write('       index         : '),write(Index),   nl,
                write('       type          : '),write(Type),    nl,
                write('       antecedents   : '),write(Antec),   nl,
                write('       consequence   : '),write(Conse),nl,nl]).
print(justs,D)
    :- true,!,
    get_slot(justification,justifications,Justs,D),
    justifications(Justs,D).
print(ass(Ass),D)
    :- true,!,
    get_slot(Ass,index,Index,D),
    get_slot(Ass,node,Node,D),
    get_slot(Ass,environments,Envs,D),
    printf([nl,write('      ------------------------------ '),nl,
                write('            ASSUMPTION               '),nl,
                write('      ------------------------------ '),nl,
                write('       assumption   : '),write(Ass),    nl,
                write('       index        : '),write(Index),  nl,
                write('       node         : '),write(Node),   nl,
                write('       environments : '),write(Envs),nl,nl]).
print(asss,D)
    :- true,!,
    get_slot(assumption,assumptions,Asss,D),
    assumptions(Asss,D).

print(env(Env),D)
    :- true,!,
    get_slot(Env,index,Index,D),
```

```
    get_slot(Env,ass_size,Size,D),
    get_slot(Env,assumptions,Asss,D),
    get_slot(Env,nodes,Nodes,D),
    get_slot(Env,contradictory,Contra,D),
    printf([nl,write('      ------------------------------  '),  nl,
            write('              ENVIRONMENT             '),  nl,
            write('      ------------------------------  '),  nl,
            write('       environment   : '),write(Env),      nl,
            write('       index         : '),write(Index),    nl,
            write('       ass_size      : '),write(Size),     nl,
            write('       assumptions   : '),write(Asss),     nl,
            write('       nodes         : '),write(Nodes),    nl,
            write('       contradictory : '),write(Contra),nl,nl]).
print(envs,D)
    :- true,!,
    get_slot(environment,environments,Envs,D),
    environments(Envs,D).
print(node_justs(Node),D)
    :- true,!,
    get_slot(Node,datum,Datum,D),
    get_slot(Node,c_justs,Justs,D),
    printf([nl,write('      ================================'),nl,
            write('          NODE - JUSTIFICATION      '),nl,
            write('      ================================'),nl]),
    justifications(Justs,D).
print(env_with_index(Index),D)
    :- true,!,
    get_with_index(environment,env_table,Index,Envs,D),
    environments(Envs,D).
print(env_datum(Env),D)
    :- true,!,
    get_slot(Env,contradictory,Contra,D),
    get_slot(Env,assumptions,Asss,D),
    printf([nl,write('      --------------------------------'), nl,
            write('        ASSUMPTION-DATUM in ENV     '), nl,
            write('      --------------------------------'), nl,
            write('       environment   : '),write(Env),    nl,
            write('       contradictory : '),write(Contra),nl]),
    env_datum(Asss,D).
print(env_data,D)
    :- true,!,
    get_slot(environment,environments,Envs,D),
    env_data(Envs,D).
print(table,D)
    :- true,!,
    get_slot(environment,indices,Env_indices,D),
    get_slot(nogood,indices,Nogood_indices,D),
    printf([nl,write('      --------------------------------'),nl,
            write('          ENVIRONMENT TABLE       '),nl,
            write('      --------------------------------'),nl]),
    table_sub1(Env_indices,D),
    printf([nl,write('      --------------------------------'),nl,
            write('            NOGOOD TABLE          '),nl,
            write('      --------------------------------'),nl]),
    table_sub2(Nogood_indices,D).
print(context(Node),D)
    :- true,!,
    context_sub1(Node,1,D),
    printf([nl]).


%%% local predicates

:- mode statuses(+,+).
statuses([],_)
    :- true,!,true.
```

```
statuses([Node|Nodes],D)
    :- true,!,
    print(status(Node),D),
    statuses(Nodes,D).

:- mode nodes(+,+).
nodes([],_)
    :- true,!,true.
nodes([Node|Nodes],D)
    :- true,!,
    print(node(Node),D),
    nodes(Nodes,D).

:- mode justifications(+,+).
justifications([],_)
    :- true,!,true.
justifications([Jus|Justs],D)
    :- true,!,
    print(just(Jus),D),
    justifications(Justs,D).

:- mode assumptions(+,+).
assumptions([],_)
    :- true,!,true.
assumptions([Ass|Asss],D)
    :- true,!,
    print(ass(Ass),D),
    assumptions(Asss,D).

:- mode environments(+,+).
environments([],_)
    :- true,!,true.
environments([Env|Envs],D)
    :- true,!,
    print(env(Env),D),
    environments(Envs,D).

:- mode env_datum(+,+).
env_datum([],_)
    :- true,!,true.
env_datum([Ass|Asss],D)
    :- true,!,
    get_slot(Ass,node,Node,D),
    get_slot(Node,datum,Datum,D),
    printf([write('      datum        : '),write(Datum),nl]),
    env_datum(Asss,D).

:- mode env_data(+,+).
env_data([],_)
    :- true,!,true.
env_data([Env|Envs],D)
    :- true,!,
    print(env_datum(Env),D),
    env_data(Envs,D).

:- mode table_sub1(+,+).
table_sub1([],_)
    :- true,!,true.
table_sub1([Index|Indices],D)
    :- true,!,
    printf([write('     env index : '),write(Index),nl]),
    get_with_index(environment,env_table,Index,Envs,D),
    printf([tab(5),write(Envs),nl]),
    table_sub1(Indices,D).

:- mode table_sub2(+,+).
```

```
table_sub2([],_)
    :- true,!,true.
table_sub2([Index|Indices],D)
    :- true,!,
    printf([write('     nogood index : '),write(Index),nl]),
    get_with_index(nogood,nogood_table,Index,Nogoods,D),
    printf([tab(5),write(Nogoods),nl]),
    table_sub2(Indices,D).

:- mode context_sub1(+,+,+).
context_sub1(Node,N,D)
    :- true,!,
    class_object(ass,Node,Result),
    context_sub2(Result,Node,N,D).


:- mode context_sub2(+,+,+,+).
context_sub2(yes,Node,N,_)
    :- true,!,
    set_cursor(N),
    printf([write(' ASSUMPTION    : '),write(Node)]).
context_sub2(no,Node,N,D)
    :- true,!,
    get_slot(Node,label,Label,D),
    get_slot(environment,empty_env,Empty_env,D),
    context_sub3(Label,Empty_env,Node,N,D).

:- mode context_sub3(+,+,+,+,+).
context_sub3([],_,Node,N,_)
    :- true,!,
    set_cursor(N),
    printf([write(' NIL           : '),write(Node)]).
context_sub3(Label,Empty_env,Node,N,D)
    :- Label=[Empty_env],!,
    set_cursor(N),
    printf([write(' PREMISE       : '),write(Node)]).
context_sub3(Label,Empty_env,Node,N,D)
    :- Label\==[Empty_env],!,
    get_slot(Node,c_justs,Justs,D),
    set_cursor(N),
    printf([write(Node),write(' : ')]),
    context_sub4(Justs,N,D).

:- mode context_sub4(+,+,+).
context_sub4([],_,_)
    :- true,!,true.
context_sub4([Jus|Justs],N,D)
    :- true,!,
    get_slot(Jus,antecedents,Antecs,D),
    set_cursor(N),
    printf([write(' JUSTIFICATION : '),write(Jus)]),
    add(N,1,N1),
    context_sub5(Antecs,N1,D),
    context_sub4(Justs,N,D).

:- mode context_sub5(+,+,+).
context_sub5([],_,_)
    :- true,!,true.
context_sub5([Antec|Antecs],N,D)
    :- true,!,
    context_sub1(Antec,N,D),
    context_sub5(Antecs,N,D).

:- mode set_cursor(+).
set_cursor(N)
    :- true,!,
```

```
    add(N,5,N1),
    printf([nl,tab(N1)]).



%%% (11) solver.prl

:- mode solver(+,+,-,+,-).
solver(breadth,Alt_Sets,Sol,D0,D)
    :- true,!,
    breadth_search(Alt_Sets,Sol,D0,D).
solver(depth,  Alt_Sets,Sol,D0,D)
    :- true,!,
    depth_search(Alt_Sets,Sol,D0,D).


%%% local predicates

%% depth search
:- mode depth_search(+,-,+,-).
depth_search(Alt_Sets,Sol,D0,D)
    :- true,!,
    depth_search_sub1(Alt_Sets,Choices,D0),
    depth_search_sub3(Choices,[],Sol,D0,D).

:- mode depth_search_sub1(+,-,+).
depth_search_sub1([],Choices,_)
    :- true,!, Choices=[].
depth_search_sub1([Alt_Set|Alt_Sets],Choices,D)
    :- true,!,
    depth_search_sub2(Alt_Set,T_labels,D),
    mapcan(T_labels,Labels),
    Choices=[Labels|Choices1],
    depth_search_sub1(Alt_Sets,Choices1,D).

:- mode depth_search_sub2(+,-,+).
depth_search_sub2([],Labels,_)
    :- true,!, Labels=[].
depth_search_sub2([Alt|Alt_Set],Labels,D)
    :- true,!,
    get_slot(Alt,label,Label,D),
    Labels=[Label|Labels1],
    depth_search_sub2(Alt_Set,Labels1,D).

:- mode depth_search_sub3(+,+,-,+,-).
depth_search_sub3([],OldS,Sol,D0,D)
    :- true,!, Sol=OldS,D=D0.
depth_search_sub3([[]|_],OldS,Sol,D0,D)
    :- true,!, Sol=OldS,D=D0.
depth_search_sub3([[Choice|Choices]|Cdr],OldS,Sol,D0,D)
    :- true,!,
    depth_search_sub4(Choice,Cdr,OldS,NewS,D0,D1),
    depth_search_sub3([Choices|Cdr],NewS,Sol,D1,D).

:- mode depth_search_sub4(+,+,+,-,+,-).
depth_search_sub4(Sol,[],OldS,NewS,D0,D)
    :- true,!, NewS=[Sol|OldS],D=D0.
depth_search_sub4(Sol,[Car|Cdr],OldS,NewS,D0,D)
    :- true,!,
    get_slot(Sol,contradictory,Contra,D0),
    depth_search_sub5(Contra,Sol,Car,Cdr,OldS,NewS,D0,D).

:- mode depth_search_sub5(+,+,+,+,+,-,+,-).
depth_search_sub5(Contra,Sol,Car,Cdr,OldS,NewS,D0,D)
```

```
    :- Contra=[],!,
    depth_search_sub6(Sol,Car,Cdr,OldS,NewS,D0,D).
depth_search_sub5(Contra,_,_,_,OldS,NewS,D0,D)
    :- Contra\==[],!,NewS=OldS,D=D0.

:- mode depth_search_sub6(+,+,+,+,-,+,-).
depth_search_sub6(_,[],_,OldS,NewS,D0,D)
    :- true,!,NewS=OldS,D=D0.
depth_search_sub6(Sol,[Choice|Choices],Cdr,OldS,NewS,D0,D)
    :- true,!,
    union_envs(Sol,Choice,Nenv,D0,D1),
    depth_search_sub7(Nenv,Sol,Choices,Cdr,OldS,NewS,D1,D).

:- mode depth_search_sub7(+,+,+,+,+,-,+,-).
depth_search_sub7(Nenv,Sol,Choices,Cdr,OldS,NewS,D0,D)
    :- Nenv=[],!,
    depth_search_sub6(Sol,Choices,Cdr,OldS,NewS,D0,D).
depth_search_sub7(Nenv,Sol,Choices,Cdr,OldS,NewS,D0,D)
    :- Nenv\==[],!,
    set_slot(Nenv,contradictory,[],D0,D1),
    depth_search_sub4(Nenv,Cdr,OldS,NewS1,D1,D2),
    depth_search_sub6(Sol,Choices,Cdr,NewS1,NewS,D2,D).

%% breadth search
:- mode breadth_search(+,-,+,-).
breadth_search(Alt_Sets,Sol,D0,D)
    :- true,!,
    get_slot(environment,empty_env,Empty_env,D0),
    breadth_search_sub1(Alt_Sets,[[Empty_env]],Sol,D0,D).

:- mode breadth_search_sub1(+,+,-,+,-).
breadth_search_sub1([],CurS,Sol,D0,D)
    :- true,!, Sol=CurS,D0=D.
breadth_search_sub1([Alt_Set|Alt_Sets],CurS,Sol,D0,D)
    :- true,!,
    breadth_search_sub2(Alt_Set,[],CurS,NewS,D0,D1),
    breadth_search_sub1(Alt_Sets,NewS,Sol,D1,D).

:- mode breadth_search_sub2(+,+,+,-,+,-).
breadth_search_sub2([],MidS,_,Sol,D0,D)
    :- true,!, Sol=MidS,D=D0.
breadth_search_sub2([Alt|Alt_Set],MidS,CurS,Sol,D0,D)
    :- true,!,
    breadth_search_sub3(CurS,Alt,MidS,NewS,D0,D1),
    breadth_search_sub2(Alt_Set,NewS,CurS,Sol,D1,D).

:- mode breadth_search_sutb3(+,+,+,-,+,-).
breadth_search_sub3([],_,MidS,NewS,D0,D)
    :- true,!, NewS=MidS,D=D0.
breadth_search_sub3([OldE|CurS],Alt,MidS,NewS,D0,D)
    :- true,!,
    get_slot(Alt,label,Label,D0),
    get_slot(environment,empty_env,Empty_env,D0),
    get_new_envs(Empty_env,[OldE,Label],NewE,D0,D1),
    breadth_search_sub4(NewE,Alt,CurS,MidS,NewS,D1,D).

:- mode breadth_search_sub4(+,+,+,+,-,+,-).
breadth_search_sub4(NewE,Alt,CurS,MidS,NewS,D0,D)
    :- NewE=[],!,
    breadth_search_sub3(CurS,Alt,MidS,NewS,D0,D).
breadth_search_sub4(NewE,Alt,CurS,MidS,NewS,D0,D)
    :- NewE\==[],!,
    append([NewE],MidS,New_MidS),
    breadth_search_sub3(CurS,Alt,New_MidS,NewS,D0,D).
```

```
%%% (12) atms.com

ghccompile :-
          ghccompile('basic.ghc', 'basic.prl'),
    ttynl,ghccompile('enviro.ghc','enviro.prl'),
    ttynl,ghccompile('node.ghc',  'node.prl'),
    ttynl,ghccompile('justif.ghc','justif.prl'),
    ttynl,ghccompile('assump.ghc','assump.prl'),
    ttynl,ghccompile('nogood.ghc','nogood.prl'),
    ttynl,ghccompile('interf.ghc','interf.prl'),
    ttynl,ghccompile('atms.ghc',  'atms.prl'),
    ttynl,ghccompile('init.ghc',  'init.prl'),
    ttynl,ghccompile('print.ghc','print.prl'),
    ttynl,ghccompile('solver.ghc','solver.prl'),
    ttynl.

'$$$menu' :-
    ttynl,
    display(' predicates to load :'),
    ttynl,ttynl,
    display(' predicate            function'),
    ttynl,
    display(' ------------------------------------------------'),
    ttynl,
    display('     c            load ATM-core'),
    ttynl,
    display('     p            load print-routine'),
    ttynl,
    display('     s            load simple problem solver'),
    ttynl.

'$$$load'(O) :-
    ttynl,
    display(O),display(' compiling'),
    ttynl,
    compile(O).

c :-
    '$$$load'('basic.prl'),
    '$$$load'('enviro.prl'),
    '$$$load'('node.prl'),
    '$$$load'('justif.prl'),
    '$$$load'('assump.prl'),
    '$$$load'('nogood.prl'),
    '$$$load'('interf.prl'),
    '$$$load'('atms.prl'),
    '$$$load'('init.prl').

p :- '$$$load'('print.prl').

s :- '$$$load'('solver.prl').


:- ttynl,ttynl,
   display(' *** ATMS (An Assumption-based Truth Maintenance System) ***'),
   ttynl,ttynl,
   (ghccompile,'$$$menu' ; '$$$menu'),
   ttynl.
```

B.　GHCへの移植の手引き

A.に掲載したプログラムはDEC-10 Prolog, Quintus Prologの上でそのままで実行できるが,以下の(1)(2)(3)の手続きにより簡単にGHCのプログラムとなる。

GHCの処理系としてはDEC-10 Prolog上のものを想定しているが,Quintus Prologの上に同様の処理系を構成しても使うことができる。

(1) シンタックスの変更

Prologのカット・オペレータをGHCのガード・オペレータに,項の比較のPrologのオペレータ'\=='を対応するGHCのオペレータ'\='に,mode宣言,public宣言(atms/1のみ)をコメント行に置き換えるだけでよい。

置き換えられる文字列	置き換える文字列
,!,	|
\==	\=
:- mode	% :- mode
:- public	% :- public

(2) 述語定義の変更

add/3(ファイルbasic.prlに定義)の中のPrologのオペレータ' is 'を対応するGHCのオペレータ' := 'に変更する。

(3) 述語定義の削除

prolog/1(ファイルbasic.prlに定義)の定義を削除する。prolog/1はGHCでは組み込み述語である。