北見工業大学研究報告　　Memoirs of the Kitami Institute of Technology
第22巻　第1号（平成2年）　　Vol. 22 No. 1 (1990)

フレーム型知識とワーキングメモリ型知識間の知識表現形式自動変換プログラムの設計

藤原祥隆*　松西年春*
野尻武文**　今村晴尚***

（平成2年5月2日受理）

Design of a Program for Automatic Format Transformation between Frame-based and Working-memory-based knowledges

by Yoshitaka FUJIWARA, Toshiharu MATSUNISHI, Takefumi NOJIRI and Haruhisa IMAMURA

A variety of software systems called "shalls" or "knowledge base design aids" have been proposed in order to reduce the human labour required to develop expert systems. In those software systems, several knowledge representing methods such as "if-then rule", "frame" and "semantic-network" are provided to expert system developers.

However, all the methods are not necessarily supported in every operating system (OS) or every cpu model. Therefore, a change in the software developing environment, which is characterized by a specific combination of cpu models and OS, frequently follows the transformation of the knowledge bases from one method to another. This kind of transformation usually requires a lot of human labour.

With this background, a program has been developed to automatically transform the knowledge bases between two types of knowledge representing methods of the KBMS.

Because each of the two methods employs the list structured format to represent fact-type knowledge as well as rule type knowledge, this program is produced by the LISP (Common LISP), which is considered to be one of the most adequate programming languages to process list structured data.

This paper presents the basic ideas and the design results of the program.

1. まえがき

医者，技術者，証券アナリストなど，種々の分野の専門家の仕事をコンピュータに代替させようとする"エキスパートシステム"の研究が近年，盛んに行われている。効率良いエキス

*　北見工業大学電子工学科
**　札幌テレビ放送（元北見工業大学電子工学科）
***　富士写真フイルム（元北見工業大学電子工学科）

パートシステムの設計を狙いに，推論エンジン，知識獲得機能，マンマシンインタフェース機能などを標準化しレディメイドのソフトウエアモジュールとしてあらかじめ組み込み，ユーザは対象とする問題の解決に必要な知識だけをコンピュータに投入すればエキスパートシステムを完成することができる "シェル" 或は "知識ベース構築支援ツール" と呼ばれる便利なソフトウエアツールが幾つか商用化されている[1]。このようなシェル，或はツールの多くは多様な応用に適用できるように，if-then ルール，フレーム，意味ネットワークなど複数の知識表現方法を提供している。しかし，コンピュータの機種や使用するシステムソフトウエアの条件によって使用可能な知識表現法が異なることがある。そのためコンピュータ機種やシステムソフトウエアの環境の変化に伴い，ある知識表現法から他の知識表現法へ知識ベースの表現法を変換しなければならないことが起こる。このような状況においては，ある知識表現方法から別の表現方法に自動的に変換する手段があれば変換作業が効率よく行える。しかし，異なる知識表現法の間の変換は一種の翻訳処理であり，相互に明確に対応づけできる機能がどれだけあるかまた規則性がどれだけあるかにより実現の困難度が大きく左右される。一般に知られているルール，フレーム，意味ネットワークなどの知識表現方法は相互の対応付けが明確でない部分が多いと考えられるため，相互変換の処理は困難になると推測される。

以上の背景のもとで，対応する知識表現間の対応づけが明確であるため実現が比較的容易であること，また大型機種で開発した知識ベースをパーソナルコンピュータ上に移植するというケースが想定されるために自動変換のメリットも高いと考えられること，の二つの理由により KBMS[2] の二つの知識表現方法（フレーム型知識表現方法とワーキングメモリ型知識表現方法）間の自動形式変換プログラムを設計製作することとした。KBMS のフレーム型知識表現法は if-then ルールとフレームの二つの表現法の併用を許容する混合型の知識表現パラダイムである。ワーキングメモリ型知識表現法は，フレーム型知識表現法と基本的考え方は同じであるが，本来高性能マシンを想定して開発されたフレーム型知識表現法を性能の低いパーソナルコンピュータ上でも実用上，使用に耐えられる性能を発揮できるようにと，本来のフレームの有する機能中，基本的な機能のみに着目しフレームの物理構造を簡単化して高速化を図ったものである。

本変換プログラムは，これら二つの知識表現法がリスト構造をとっていることに着目し，リスト処理に最も適した LISP 言語を用いて，ソースファイルのレベルで一方の表現を入力としてこれに等価な他方の表現を出力する変換メカニズムを実現している。以下，2 章で本変換プログラムの基本的な考え方を，3 章で当該変換プログラムの構成・動作を，4 章で具体例を対象とした本プログラムの評価例を示す。

2. 変換プログラムの基本的考え方

2.1 フレーム型およびワーキングメモリ型知識表現法の概要

簡単な例題を用いて対象とするKBMSの二つの知識表現法の概要を説明する。まず知識を事実知識と規則知識に分けて考える。事実知識は例えば，"赤いバラの花"，"白いバラの花"のように"植物"，"バラ科"のような共通の属性を持つ個々の具体的な事実が集まったものとしてとらえる。これらの事実知識はそのグループを特徴づける共通的属性を記述したラベルに相当する部分（クラスと呼ぶ）と個々の具体的な事実を記述する部分（インスタンスと呼ぶ）の階層構造により表現する。図1に，"バラ"，"症状"，"結論"，の三つの異なるタイプの事実知識のグループが上記の考え方でどのように表現されるかを示す。またこの事実知識におけるクラス定義表現をフレーム型表現法と，ワーキングメモリ型表現法で記述するとソーステキストのレベルで，それぞれ図2，図3のようになる。一方，規則知識はif-thenルールの形式で表現する。一つのルールは条件部とアクション部により構成され，条件部は当該ルールのアクション部を実行するために必要な条件をAND条件で列挙する。図4に図1の事実知識に関係する規則知識の例を示す。図4の規則知識をフレーム型表現法，ワーキングメモリ型表現法で表現

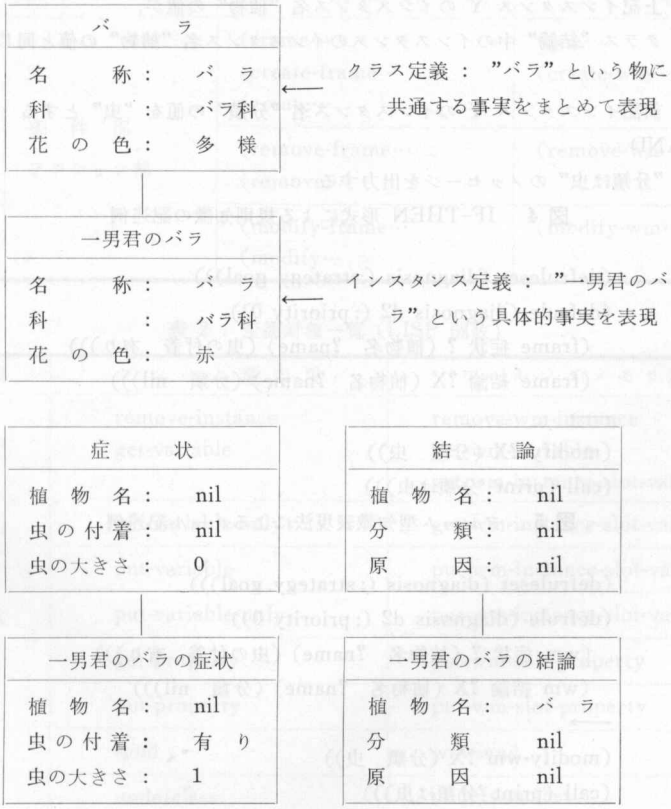


図1　フレームの例

```
(defclass バラ
        :IV ((名称　バラ) (科　バラ科) (花の色　多様)))
(defclass 症状
        :IV ((植物名　nil) (虫の付着　nil) (虫の大きさ　0)))
(defclass 結論
        : IV ((植物名　nil) (分類　nil) (原因　nil)))
```

図 2　フレーム型知識表現法によるクラス定義の記述例

```
(defwmclass バラ
        (名称　バラ) (科　バラ科) (花の色　多様))
(defwmclass 症状
        (植物名　nil) (虫の付着　nil) (虫の大きさ　0))
(defwmclass 結論
        (植物名　nil) (分類　nil) (原因　nil))
```

図 3　ワーキングメモリ型知識表現法によるクラス定義の記述例

IF
・クラス "症状" のインスタンス中，インスタンス名 "虫の症状" の値が
"有り" であるインスタンス Y が存在する
AND
・上記インスタンス Y のインスタンス名 "植物" の値が，
クラス "結論" 中のインスタンスのインスタンス名 "植物" の値と同じである
THEN
・前記インスタンス Y のインスタンス名 "分類" の値を "虫" とする
AND
・"分類は虫" のメッセージを出力する

図 4　IF–THEN 形式による規則知識の記述例

```
(defruleset (diagnosis (:strategy goal)))
(defrule (diagnosis d2 (: priority 0))
  (frame 症状 ? (植物名　?name) (虫の付着　有り)))
  (frame 結論 ?X (植物名　?name) (分類　nil)))
⟶
(modify ?X (分類　虫))
(call (print '分類は虫)))
```

図 5　フレーム型知識表現法によるルール記述例

```
(defruleset (diagnosis (: strategy goal)))
(defrule (diagnosis d2 (: priority 0))
  (wm 症状 ? (植物名　?name) (虫の付着　有り)))
  (wm 結論 ?X (植物名　?name) (分類　nil)))
⟶
(modify-wm ?X (分類　虫))
(call (print '分類は虫)))
```

図 6　ワーキングメモリ型知識表現法によるルール記述例

すると，ソーステキストのレベルでそれぞれ，図5，図6のようになる。

2.2 変換に際しての前提条件

ワーキングメモリ型表現は前述したように，フレーム型表現の部分集合であるため，フレーム型にはあるがワーキングメモリ型には存在しない機能(換言すると変換できない機能)がある。例えば，ワーキングメモリ型では，図3のインスタンスはルールのアクション部でしか生成できないが，フレーム型ではルールのアクション部とは独立にエデイタでも生成できる。このため双方向の変換を可能とするため，ワーキングメモリ型表現法に存在する機能を基準にする。

その結果，二つの知識表現法間で対応関係があるが表現形式が異なる表現（変換対象）は，クラス定義，ルールの条件部，アクション部，およびアクションで呼び出すLISP関数，の各カテゴリで表1および表2に示す種類になる。

表1　変換対象一覧（フレーム，ルール）

分　類	項　目	フレーム型表現	ワーキングメモリ型表現
フレーム	クラス定義	(delclass 木 :IV (科 nil))	(defwmclass 木 (科 nil))
	インスタンス生成	(create-instance '木 nil '科 'バラ)	(create-wm-instance '木: 科 'バラ)
ルール	条件部 アクション部	(frame… (create-frame… (create…	(wm… (create-wm…
		(remove-frame… (remove…	(remove-wm…
		(modify-frame… (modify…	(modify-wm…

表2　変換対象一覧（LISP 関数）

分　類	フレーム型表現	ワーキングメモリ型表現
LISP 関数	remove-instance get-variable	remove-wm-instance get-wm-variable get-wm-instance-slot-value
	get-variable-only	get-wm-instance-slot-value-only
	put-variable	put-wm-instance-slot-value
	put-variable-only	put-wm-instance-slot-value-only
	get-property	get-wm-solt-property
	put-property	put-wm-slot-property
	send	wm-send
	undefclass	undefwmclass delete-wmclass

3. 変換プログラムの構成

3.1 フレーム型からワーキングメモリ型への変換プログラム

3.1.1 プログラムの構成

(1) ルール変換部

ルール変換部を構成する関数 (C 言語等におけるサブルーチンあるいはプロシジャーと等価) とその機能の概要を表 3 に示す。

(2) クラス変換部

クラス変換部を構成する関数とその機能の概要を表 4 に示す。

表 3 ルール変換を構成する関数とその機能の概要

関数名	機能の概要
rconv	ルール変換プログラムの全体の流れを制御する。
fileopen1	「フレーム型のルールを WM 型に変換します」というタイトルを表示してから，フレーム型のルールが入っているファイル名の入力を要求する。また，変換後に格納するファイル名の入力も要求する。そして入力されたファイル名を持つファイルを開く。
datain1	開かれた入力ファイルからストリームを読み込み，そのストリームから一つのリストを取り出してそれを関数の値にする。
search	読み込まれたルールの中にある要素を取り出す。深いところに要素があっても取り出すことができる。
lldl	関数 datainl で取り出されたリスト二つを引数にして，それらを結合させる。
lddd	リストとデータを引数として，それらを結合させる。
change1	引数に対して，フレーム型の関数に該当するものがあれば，それをワーキングメモリ型の関数に変換し，なければ引数そのものを関数値とする。
out1	型式変換されたルールを画面に表示し，出力用ファイルに書き込む。
fileclose1	出力ファイルと入力ファイルを閉じる。

表 4 クラス変換を構成する関数とその機能の概要

関数名	機能の概要
cconv	クラス変換プログラムの全体の流れを制御する。
fileopen2	「フレーム型のクラス定義文を WM 型に変換します」というタイトルを表示してから，フレーム型のクラス定義文が入っているファイル名の入力を要求する。また，変換後に格納するファイル名の入力も要求する。そして，入力されたファイル名を持つファイルを開く。
datain2	開かれた入力ファイルからストリームを読み込み，そのストリームから一つのリストを取り出してそれを関数の値にする。
change2	読み込まれたクラス定義文の中にある要素からリストが一つ与えられ，その第 1 番目の要素が defclass なら defwmclass に変換し，definstance ならスキップしてそのリストを削除する。
lldd	関数 datain1 で取り出されたリスト二つを引数にして，それらを結合させる。
lddd	リストとデータを引数として，それらを結合させる。
out2	型式変換されたクラス定義文を画面に表示し，出力用ファイルに書き込む。
fileclose2	出力ファイルと入力ファイルを閉じる。

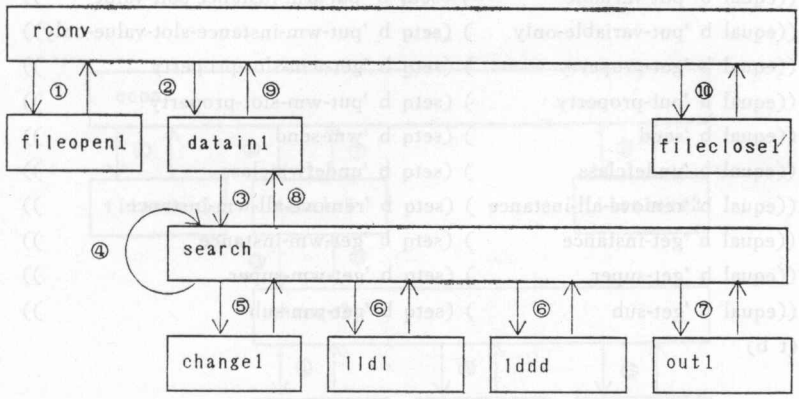
3.2.1　プログラムの動作

(1)　ルール変換部

3.1.1の表3で示した関数を用いたルール変換の手順を図7に示す。また上記関数中，ルール変換の主要関数である，changel, search, rconv, の COMMON LISP による記述例をそれぞれ，図8, 図9, 図10に示す。

(2)　クラス変換部

3.1.1の表4で示した関数を用いたクラス変換の手順を図11に示す。また上記関数中，クラス変換の主要関数である，change2, cconv, の COMMON LISP による記述例をそれぞれ，図12, 図13, に示す。



【動作の説明】

①：タイトルを書いて入出力用ファイル名の入力要求をする。

②：ストリームからリストを一つ取り出す。

③：取り出されたリストから要素を取り出す。

④：深いところにある要素を取り出すため，繰り返す。

⑤：型式変換に該当する関数があったら変換し，無ければそのままを返す。

⑥：変換された関数と，そうでない要素を結び付ける。

⑦：結果を画面に表示し，出力ファイルに書き込む。

⑧：新たなリストを取りに行く。

(②から⑧を繰り返す。)

⑨：リストがなくなったら帰る。

⑩：すべてが終了したら入出力ファイルを閉じておしまい。

図7　ルール変換の手順

```
(defun change1 (b)
 (cond
   ((equal b 'frame                ) (setq b 'wm                                  ))
   ((equal b 'create               ) (setq b 'create-wm                           ))
   ((equal b 'modify               ) (setq b 'modify-wm                           ))
   ((equal b 'remove               ) (setq b 'remove-wm                           ))
   ((equal b 'create-frame         ) (setq b 'create-wm                           ))
   ((equal b 'modify-frame         ) (setq b 'modify-wm                           ))
   ((equal b 'remove-frame         ) (setq b 'remove-wm                           ))
   ((equal b 'remove-instance      ) (setq b 'remove-wm-instance                  ))
   ((equal b 'get-variable         ) (setq b 'get-wm-variable                     ))
   ((equal b 'get-variable-only    ) (setq b 'get-wm-instance-slot-value-only))
   ((equal b 'put-variable         ) (setq b 'put-wm-instance-solt-value          ))
   ((equal b 'put-variable-only    ) (setq b 'put-wm-instance-slot-value-only))
   ((equal b 'get-property         ) (setq b 'get-wm-slot-property                ))
   ((equal b 'put-property         ) (setq b 'put-wm-slot-property                ))
   ((equal b 'send                 ) (setq b 'wm-send                             ))
   ((equal b 'undefclass           ) (setq b 'undefwmclass                        ))
   ((equal b 'remove-all-instance  ) (setq b 'remove-all-wm-instance               ))
   ((equal b 'get-instance         ) (setq b 'get-wm-instance                     ))
   ((equal b 'get-super            ) (setq b 'get-wm-super                        ))
   ((equal b 'get-sub              ) (setq b 'get-wm-sub                          ))
   (t b)
 )
)
```

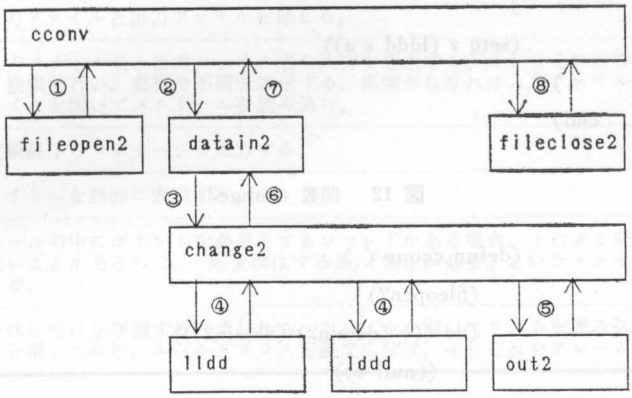
図 8　関数 change1

```
(defun search (x)
   (do ((f 0) (z nil) (a nil))
       ((null x) z)
    (tagbody
       (setq a (car x)) (setq x (cdr x))
       (cond ((and (equal f 2) (not (listp a))) (go skip))
             ((and (equal f 1) (not (listp a))) (setq f 2))
             (t (setq f 0))
       )
       (cond ((listp a) (setq z (lldl z (search a))))
             (t         (setq a (changel a))
                        (if (equal a 'create-wm) (setq f 1))
                        (setq z (lddd z a))
             )
       )
    skip)
   )
)
```

図 9　関数 search

```
(defun rconv ( )
    (fileopen1)
    (do ((v (datain1) (datain1)))
        ((null v))
        (out1 (search v))
    )
    (fileclosel)
    (terpri)
    '終了しました。
)
```

図 10　関数 rconv



【動作の説明】

①：タイトルを書いて入出力用ファイル名の入力要求をする。

②：ストリームからリストを一つ取り出す。

③：取り出されたリストを変換する。

④：変換された関数と，そうでない要素を結び付ける。

⑤：結果を画面に表示し，出力ファイルに書き込む。

⑥：新たなリストを取りに行く。
（②から⑥を繰り返す。）

⑦：リストがなくなったら帰る。

⑧：すべて終了したら入出力ファイルを閉じておしまい。

図 11　クラス変換の手順

```
(defun change2 (x)
  (tagbody
    (do ((z nil) (a nil))
        (null x))
        (setq a (car x)) (setq x (cdr x))
        (cond ((equal a 'defclass) (setq a 'defwmclass))
              ((equal a 'definstance) (go end))
              ((equal a ': iv       ) (progn
                                        (setq z (lldd z (car x)))
                                        (out2 z)
                                        (go end)
                                        )
              )
        )
        (setq z (lddd z a))
    )
  end)
)
```

図 12 関数 change2

```
(defun cconv ( )
  (fileopen2)
  (do ((v (datain2) (datain2)))
      ((null v))
      (change2 v)
  )
  (fileclose2)
  (terpri)
  '終了しました。
)
```

図 13 関数 cconv

3.2 フレーム型からワーキングメモリ型への変換プログラム

3.2.1 プログラムの構成

ルール変換とクラス変換を構成する関数とその機能の概要を表 5 に示す。

3.2.2 プログラムの動作

3.2.1 の表 5 で示した関数を用いたルール変換とクラス変換の手順を図 14 に示す。また上記関数中，ルール変換とクラス変換の主要関数である，change, convert, wf, の COMMON LISP による記述例をそれぞれ，図 15, 図 16, 図 17, に示す。

表 5　ルール変換とクラス変換を構成する関数とその機能の概要

関数名	機能の概要
change	関数 datain によって読み込まれたリストを要素に分け，クラス定義文やインスタンス生成文が読み込まれたなら，それぞれ専用の変換関数を呼び出して処理する。
change1	関数 change でリストと判定された要素は，この関数でフレーム型に変換を指示される。
change2	関数 change でリストと判定されなかった要素を変換する。
ch-class	要素がクラス定義文ならフレーム型のクラス定義文に変換する。
ch-ins1	インスタンス生成文を変換する。
ch-ins2	インスタンス生成文を変換する。
convert	引数に対して，ワーキングメモリ型の関数に該当するものがあれば，それをフレーム型の関数に変換し，なければ引数そのものを関数値とする。
datain	開かれた入力ファイルからリストを読み込む。
fileclose	入力ファイルと出力ファイルを閉じる。
fileopen	入力ファイル名と出力ファイル名の入力を要求する。そしてそれらのファイル名に対して検索を行い，重複や不明を表示する。異常がなければ入力ファイル名を持っているファイルを開いてストリームを読み込む。
final	変換終了のメッセージを出力する。
title	タイトルを画面に表示する。
warn	ルールの中にポイントを必要とするコマンドがある場合，そのまま変換したのでは使えないことがあるので，一応変換はするが，「修正が必要」というメッセージを画面に表示する。
wf	全体の流れを制御する。すなわち，ファイルを開いてリストを読み込み，それらを要素に分解してから，ルールとクラス定義文に分け，それぞれをフレーム型に変換する。

4. 評　価　例

4.1　フレーム型からワーキングメモリ型への変換例

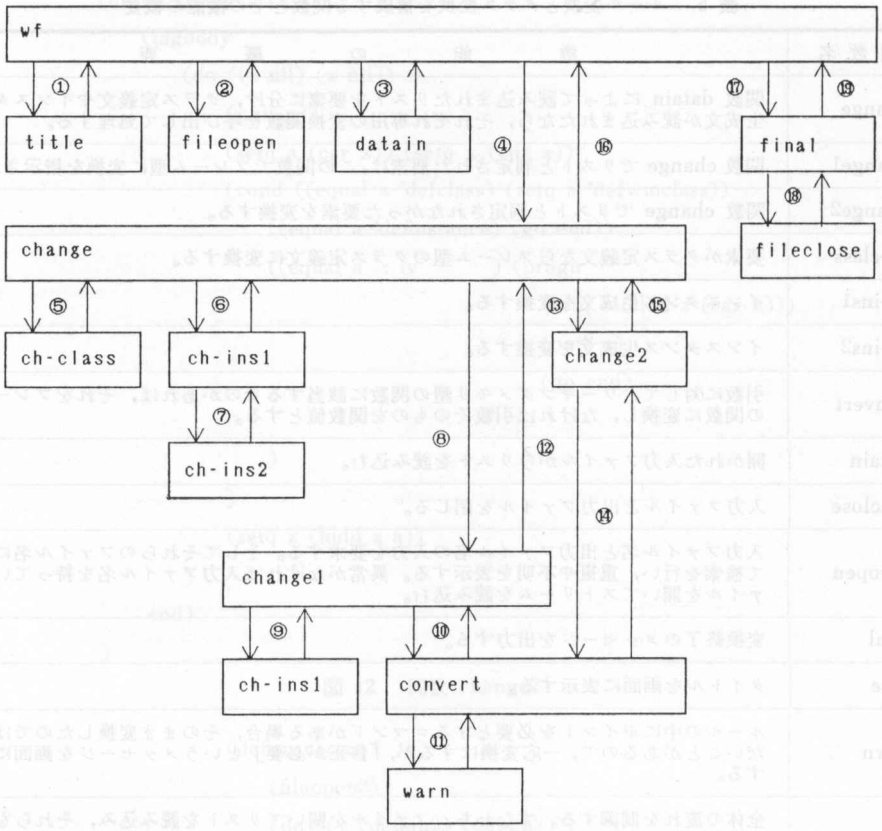
ここでは「バイクの故障診断」を例にして変換を行い，そのルールを実行して変換の妥当性を確認した。変換前のクラス定義文とルールの一部を図 18 に示し，変換後のクラス定義文とルールの一部を図 19 に示す。

4.2　ワーキングメモリ型からフレーム型への変換例

ここでは「植物の病虫害診断」を例にして変換を行い，そのルールを実行して変換の妥当性を確認した。変換前のクラス定義文とルールの一部を図 20 に示し，変換後のクラス定義文とルールの一部を図 21 に示す。

5. む　す　び

以上，KBMS の異なる二つの知識表現法に関し，相互に表現形式を変換する変換プログラムの，基本的考え方，具体的な構成・動作，さらに具体例による変換プログラムの評価例につ



【動作の説明】

①： タイトルを表示する。
②： 入出力用ファイル名の入力要求をし，それぞれファイルを開く。
③： 入力ファイルからデータを読み込む。
④： 読み込まれたデータから要素を取り出す。
⑤： 要素がクラス定義文ならフレーム型のクラス定義文に変換する。
⑥⑦： 要素がインスタンス生成文なら，変換する。
⑧： 要素がクラス定義文でなく，かつリストなら分解し，
⑨： もしリストが，create-wm-instance というものなら，CREATE-INSTANCE 変換する。(⑥の段階で判定できないことがあるため。)
⑩： リストがそれ以外ならフレーム型の関数に変換する。
⑪： リストの中にポインタを必要とするコマンドがあったら，変換せずに注意を促す表示をする。
⑫： 新たな要素を取りに行く。(あったら⑧から繰り返す。)
⑬： 要素がリストで無かったら，
⑭： フレーム型の関数に変換する。
⑮： 新たな要素を取りに行く。(あったら⑬から繰り返す。)
⑯： まだデータがあるか調べに行く。
⑰： データがなければ，画面に終了の表示をする。
⑱： 入出力用のファイルを閉じる。
⑲： すべてを終了する。

図 14 ルール変換とクラス変換の手順

```
(defun change (x)
  (cond ((eq (car x) 'defwmclass) (ch-class x))
        ((eq (car x) 'create-wm-instance) (ch-insl (cdr x)))
        (t
          (let
            ((n (length x))
             (y)
             (z)
             (ans)
            )
            (dotimes (i n ans)
              (setq y (car x))
              (setq z (if (consp y)
                          (list (change1 y))
                          (list (change2 y))
                          )
              )
              (setq ans (append ans z))
              (setq x (cdr x))
            )
          )
        )
  )
)
```

図 15　関数 change

```
(defun convert (x)
  (cond ((eq x 'wm) (setq x 'frame))
        ((eq x 'create-wm) (setq x 'create-frame))
        ((eq x 'modify-wm) (setq x 'modify-frame))
        ((eq x 'remove-wm) (setq x 'remove-frame))
        ((or (eq x 'remove-all-wm-instance)
             (eq x 'clear-wm-instance)) (setq x 'remove-all-instance))
             ((or (eq x 'delete-wmclass)
              (eq x 'undefwmclass)) (setq x 'undefclass))
        ((eq x 'get-wm-instance) (setq x 'get-instance))
        ((eq x 'get-wm-super) (setq x 'get-super))
        ((eq x 'get-wm-sub) (setq x 'get-sub))
        ((eq x 'remove-wm-instance) (warn x)
             (setq x 'remove-instance))
        ((or (eq x 'get-wm-instance-slot-value)
             (eq x 'get-wm-variable)) (warn x)
             (setq x 'get-variable))
        ((eq x 'get-wm-instance-slot-value-only) (warn x)
             (setq x 'get-variable-only))
        ((seq x 'put-wm-instance-slot-value) (warn x)
             (setq x 'put-varirble))
        ((eq x 'put-wm-instance-slot-value-only) (warn x)
             (setq x 'put-variable-only))
        ((eq x 'get-wm-slot-property) (warn x)
             (setq x 'get-property))
        ((eq x 'put-wm-slot-property) (warn x)
             (setq x 'put-property))
        ((eq x 'wm-send) (warn x) (setq x 'send))
        (t x)
  )
)
```

図 16　関数 convert

```
(defun wf ( )
  (progv '(*p*) '(nil)
    (catch 'stop
      (title)
      (fileopen)
      (let ((x nil))
        (do ((y 'start x))
            ((equal y nil) (final))
            (setq x (datain))
            (if (consp x)
                (let ((ans (change x)))
                  (pprint ans out-stream)
                  (print ans)
                )
            )
            (terpri out-stream)
            (terpri)
        )
      )
    )
  )
)
```

図 17　関数 wf

```
(DEFCLASS 点検  :IV ((プラグ NIL)(キャブ1 NIL)(フューエル NIL)(点火時期 NIL)
                     (圧  NIL)(タペット NIL)(オイル NIL)(タイヤ NIL)
                     (ホイール1 NIL)(エア NIL)(チェーン NIL)(クラッチ NIL)
                     (ステアリング NIL)(フォーク NIL)(キャブ2 NIL)
                     (ホイール2 NIL)(バッテリー NIL)))

(DEFCLASS 状態  :IV ((バッテリー NIL)(プラグ NIL)(少し NIL)))

(DEFCLASS 症状1 :IV ((番号 NIL)))

(DEFCLASS 症状2 :IV ((症状 NIL)))


(DEFRULESET (初期設定 (:STRATEGY GOAL)))

 (DEFRULE (初期設定 I1 (:PRIORITY 0))(FRAME (点検 ?Y)) --> (REMOVE ?Y))


(DEFRULESET (選択 (:STRATEGY GOAL)))

 (DEFRULE (選択 CH1 (:PRIORITY 0)) -->
  (CALL (FORMAT T "~%症状を選んで下さい"))
  (CALL (FORMAT T "~%1．エンジンの始動が困難"))
  (CALL (FORMAT T "~%2．アイドリングが不調"))
  (CALL (FORMAT T "~%3．高速回転が不調"))
  (CALL (FORMAT T "~%4．速度が上がらない、力がない"))
  (CALL (FORMAT T "~%5．走行中ハンドルが取られる、重い"))
  (CALL (FORMAT T "~%番号を入力して下さい。？"))
   (BIND ?NR (READ))
  (CREATE 症状1 (番号 ?NR)))


(DEFRULESET (調査1 (:STRATEGY GOAL)))

 (DEFRULE (調査1 EX11 (:PRIORITY 0))
  (FRAME (症状1 ? (番号 1))) --> (CREATE 症状2 (症状 エンジンの始動が困難)))

 (DEFRULE (調査1 EX12 (:PRIORITY 0))
  (FRAME (症状1 ? (番号 2))) --> (CREATE 症状2 (症状 アイドリングが不調)))

 (DEFRULE (調査1 EX13 (:PRIORITY 0))
  (FRAME (症状1 ? (番号 3))) --> (CREATE 症状2 (症状 高速回転が不調)))

 (DEFRULE (調査1 EX14 (:PRIORITY 0))
  (FRAME (症状1 ? (番号 4))) --> (CREATE 症状2 (症状 速度が上がらない、力がない)
))

 (DEFRULE (調査1 EX15 (:PRIORITY 0))
  (FRAME (症状1 ? (番号 5))) --> (CREATE 症状2 (症状 走行中ハンドルが取られる、重
い)))
```

（略）

```
(DEFRULESET (措置 (:STRATEGY GOAL)))

 (DEFRULE (措置 ME1 (:PRIORITY 0))
  (FRAME (点検 (プラグ Y))) -->
  (CALL (FORMAT T "~%★スパークプラグの点検★"))
  (CALL (TERPRI))
  (CALL (FORMAT T "~%・プラグを取り外し、汚れていたらワイヤーブラシで掃除する。電
極の隙間を点検するが、減っているものは新品に交換する。"))
  (CALL (FORMAT T "~%・プラグが湿ったり濡れている場合、チョークの使いすぎか、スロ
ットルバルブを開けすぎての始動が考えられる。プラグをブラシで清掃、乾かして使用し
てみる。"))
  (CALL-RULESET 一時停止))
 (DEFRULE (措置 ME2 (:PRIORITY 0))
  (FRAME (点検 (キャブ1 Y))) -->
  (CALL (FORMAT T "~%★キャブレターの点検★"))
  (CALL (TERPRI))
  (CALL (FORMAT T "~%・キャブレターを取り外して油面の高さ、フロートバルブの作動、
フロートの破損、バルブシートの摩耗などを点検整備する。エアジェットの詰まりなどは
エアガンで吹き飛ばしてやる。"))
  (CALL-RULESET 一時停止))

 (DEFRULE (措置 ME3 (:PRIORITY 0))
  (FRAME (点検 (フューエル Y))) -->
  (CALL (FORMAT T "~%★フューエルコックの点検★"))
  (CALL (TERPRI))
  (CALL (FORMAT T "~%・タンクにガソリンがあっても、キャブレターまで正しく流れてい
るか、フューエルホースを外して、コックをＯＮに、負圧式はＰＲＥにしてみる。流れが
悪ければフューエルコック自体、ストレーナーの詰まりなので、汚れを落として解決する。
"))
  (CALL-RULESET 一時停止))

          （略）

(DEFRULESET (MONITOR (:STRATEGY GOAL)))

 (DEFRULE (MONITOR M1 (:PRIORITY 7)) --> (CALL-RULESET 初期設定))

 (DEFRULE (MONITOR M2 (:PRIORITY 6)) --> (CALL-RULESET 選択))

 (DEFRULE (MONITOR M3 (:PRIORITY 5)) --> (CALL-RULESET 調査1))

          （略）
```

図 18　フレーム型からワーキングメモリ型への変換例（変換前）

```
(DEFWMCLASS 点検 (プラグ NIL) (キャブ1 NIL) (フューエル NIL) (点火時期 NIL) (圧 NIL) (タペット NIL) (オイル NIL) (タイヤ NIL) (ホイール1 NIL) (エア NIL) (チェーン NIL) (クラッチ NIL) (ステアリング NIL) (フォーク NIL) (キャブ2 NIL) (ホイール2 NIL) (バッテリー NIL))

(DEFWMCLASS 状態 (バッテリー NIL) (プラグ NIL) (少し NIL))

(DEFWMCLASS 症状1 (番号 NIL))

(DEFWMCLASS 症状2 (症状 NIL))


(DEFRULESET (初期設定 (:STRATEGY GOAL)))

(DEFRULE (初期設定 I1 (:PRIORITY 0)) (WM (点検 ?Y)) --> (REMOVE-WM ?Y))


(DEFRULESET (選択 (:STRATEGY GOAL)))

(DEFRULE (選択 CH1 (:PRIORITY 0)) --> (CALL (FORMAT T "~%症状を選んで下さい")) (CALL (FORMAT T "~%1．エンジンの始動が困難")) (CALL (FORMAT T "~%2．アイドリングが不調")) (CALL (FORMAT T "~%3．高速回転が不調")) (CALL (FORMAT T "~%4．速度が上がらない、力がない")) (CALL (FORMAT T "~%5．走行中ハンドルが取られる、重い")) (CALL (FORMAT T "~%番号を入力して下さい。？")) (BIND ?NR (READ)) (CREATE-WM 症状1 (番号 ?NR)))

        (略)

(DEFRULESET (調査1 (:STRATEGY GOAL)))

(DEFRULE (調査1 EX11 (:PRIORITY 0)) (WM (症状1 ? (番号 1))) --> (CREATE-WM 症状2 (症状 エンジンの始動が困難)))

(DEFRULE (調査1 EX12 (:PRIORITY 0)) (WM (症状1 ? (番号 2))) --> (CREATE-WM 症状2 (症状 アイドリングが不調)))

(DEFRULE (調査1 EX13 (:PRIORITY 0)) (WM (症状1 ? (番号 3))) --> (CREATE-WM 症状2 (症状 高速回転が不調)))

(DEFRULE (調査1 EX14 (:PRIORITY 0)) (WM (症状1 ? (番号 4))) --> (CREATE-WM 症状2 (症状 速度が上がらない、力がない;)))

(DEFRULE (調査1 EX15 (:PRIORITY 0)) (WM (症状1 ? (番号 5))) --> (CREATE-WM 症状2 (症状 走行中ハンドルが取られる、重い)))

        (略)

(DEFRULESET (措置 (:STRATEGY GOAL)))

(DEFRULE (措置 ME1 (:PRIORITY 0)) (WM (点検 (プラグ Y))) --> (CALL (FORMAT T "~%★スパークプラグの点検★")) (CALL (TERPRI)) (CALL (FORMAT T "~%・プラグを取り外し、汚れていたらワイヤーブラシで掃除する。電極の隙間を点検するが、減っているものは新品に交換する。")) (CALL (FORMAT T "~%・プラグが湿ったり濡れている場合、チョークの使いすぎか、スロットルバルブを開けすぎての始動が考えられる。プラグをブラシで清掃、乾かして使用してみる。")) (CALL-RULESET 一時停止))

(DEFRULE (措置 ME2 (:PRIORITY 0)) (WM (点検 (キャブ1 Y))) --> (CALL (FORMAT T "~%★キャブレターの点検★")) (CALL (TERPRI)) (CALL (FORMAT T "~%・キャブレターを取り外して油面の高さ、フロートバルブの作動、フロートの破損、バルブシートの摩耗などを点検整備する。エアジェットの詰まりなどはエアガンで吹き飛ばしてやる。")) (CALL-RULESET 一時停止))

(DEFRULE (措置 ME3 (:PRIORITY 0)) (WM (点検 (フューエル Y))) --> (CALL (FORMAT T "~%★フューエルコックの点検★")) (CALL (TERPRI)) (CALL (FORMAT T "~%・タンクにガソリンがあっても、キャブレターまで正しく流れているか、フューエルホースを外して、コックをＯＮに、負圧式はＰＲＥにしてみる。流れが悪ければフューエルコック自体、ストレーナーの詰まりなので、汚れを落として解決する。")) (CALL-RULESET 一時停止))

        (略)

(DEFRULESET (MONITOR (:STRATEGY GOAL)))

(DEFRULE (MONITOR M1 (:PRIORITY 7)) --> (CALL-RULESET 初期設定))

(DEFRULE (MONITOR M2 (:PRIORITY 6)) --> (CALL-RULESET 選択))

(DEFRULE (MONITOR M3 (:PRIORITY 5)) --> (CALL-RULESET 調査1))

        (略)
```

図 19　フレーム型からワーキングメモリ型への変換例（変換後）

```
(defwmclass 症状 (植物名 nil) (虫の付着 nil)
                 (虫の種類 nil) (虫の大きさ 0)
                 (虫食い跡 nil) (葉の変色 nil) (葉の異常 nil))

(defwmclass 結論 (植物名 nil) (分類 nil) (原因 nil))

(defwmclass 繰り返し (input nil))

(DEFRULESET (GENERATE (:STRATEGY GOAL)))

(DEFRULE (GENERATE G1 (:PRIORITY 0)) (NOT (WM (症状 ?))) -->
 (CALL (PRINT '植物名? )) (BIND ?A (READ)) (CALL (PRINT '虫の付着? ))
 (BIND ?B (READ)) (CALL (PRINT '虫の種類? )) (BIND ?C (READ))
 (CALL (PRINT '虫の大きさ? )) (BIND ?D (READ)) (CALL (PRINT '虫食い跡? ))
 (BIND ?E (READ)) (CALL (PRINT '葉の変色? )) (BIND ?F (READ))
 (CALL (PRINT '葉の異常? )) (BIND ?G (READ))
 (CREATE-WM 症状 花の症状 (植物名 ?A) (虫の付着 ?B) (虫の種類 ?C) (虫の大きさ ?D
) (虫食い跡 ?E)
  (葉の変色 ?F) (葉の異常 ?G)) (PUT-WM-INSTANCE-SLOT-VALUE-ONLY WMINST1 '病名 '風
邪))

        (略)

(DEFRULESET (CLASSIFICATION (:STRATEGY GOAL)))

(DEFRULE (CLASSIFICATION C1 (:PRIORITY 0))
 (WM (症状 ? (植物名 ?NAME) (虫の付着 有))) -->
 (CREATE-WM 結論 虫 (植物名 ?NAME) (分類 虫)) (CALL-RULESET INSECT))

(DEFRULE (CLASSIFICATION C2 (:PRIORITY 0))
 (WM (症状 ? (植物名 ?NAME) (虫の付着 無) (虫食い跡 無) (葉の変色 有))) -->
 (CREATE-WM 結論 病気 (植物名 ?NAME) (分類 病気)) (CALL-RULESET DISEASE))

        (略)

(DEFRULESET (INSECT (:STRATEGY GOAL)))

(DEFRULE (INSECT I1 (:PRIORITY 1))
 (WM
  (症状 ? (虫の種類 粒状) (虫の大きさ ?Y!(<= ?Y 1)) (虫食い跡 無) (葉の変色 有))
)
 (WM (結論 ?X (分類 虫) (原因 NIL))) --> (MODIFY-WM ?X (原因 ハダニ))
 (CALL (PRINT '原因はハダニによる虫害)))

(DEFRULE (INSECT I2 (:PRIORITY 1))
 (WM (症状 ? (虫の種類 粒状) (虫の大きさ ?Y!(> ?Y 1)) (虫食い跡 無)))
 (WM (結論 ?X (分類 虫) (原因 NIL))) --> (MODIFY-WM ?X (原因 アブラムシ))
 (CALL (PRINT '原因はアブラムシによる虫害))
 (CALL (GET-WM-INSTANCE-SLOT-VALUE WMINST1)))

        (略)

(DEFRULESET (DISEASE (:STRATEGY GOAL)))

(DEFRULE (DISEASE D1 (:PRIORITY 1)) (WM (症状 ? (植物名 バラ) (葉の異常 黒斑)))
 (WM (結論 ?X (分類 病気) (原因 NIL))) --> (MODIFY-WM ?X (原因 黒星病))
 (CALL (PRINT '原因は黒星病)))

(DEFRULE (DISEASE D2 (:PRIORITY 1))
 (WM (症状 ? (植物名 バラ) (葉の異常 凸凹斑)))
 (WM (結論 ?X (分類 病気) (原因 NIL))) --> (MODIFY-WM ?X (原因 サビ病))
 (CALL (PRINT '原因はサビ病)))

        (略)

(DEFRULESET (MONITOR (:STRATEGY GOAL)))

(DEFRULE (MONITOR M1 (:PRIORITY 5)) --> (CALL-RULESET GENERATE))

(DEFRULE (MONITOR M2 (:PRIORITY 4)) --> (CALL (SETQ X (GET-WM-SUPER '症状))) (C
ALL (PRINT X)) (CALL (SETQ Y (GET-WM-SUB '結論))) (CALL (PRINT Y)) (CALL-RULESET
 CLASSIFICATION))

        (略)

(DEFRULESET (TEST (:STRATEGY GOAL)))

(DEFRULE (TEST T1 (:PRIORITY 0))
  (NOT (WM (症状 ?))) --> (CALL (DELETE-WMCLASS '症状))
  (call (create-wm-instance '木 :科 'バラ :色 'ピンク :香り 'シトラス)))

(create-wm-instance '木 :科 'バラ :色 'ピンク :香り 'シトラス)

(DEFRULE (TEST T2 (:PRIORITY 0))
  (NOT (WM (結論 ?))) --> (CALL (UNDEFWMCLASS '結論)))
```

図 20　ワーキングメモリ型からフレーム型の変換例（変換前）

```
(DEFCLASS 症状 :IV ((植物名 NIL) (虫の付着 NIL)
                    (虫の種類 NIL) (虫の大きさ 0)
                    (虫食い跡 NIL) (葉の変色 NIL) (葉の異常 NIL)))

(DEFCLASS 結論 :IV ((植物名 NIL) (分類 NIL) (原因 NIL)))

(DEFCLASS 繰り返し :IV ((INPUT NIL)))


(DEFRULESET (GENERATE (:STRATEGY GOAL)))

(DEFRULE (GENERATE G1 (:PRIORITY 0)) (NOT (FRAME (症状 ?))) -->
 (CALL (PRINT '植物名? )) (BIND ?A (READ)) (CALL (PRINT '虫の付着? ))
 (BIND ?B (READ)) (CALL (PRINT '虫の種類? )) (BIND ?C (READ))
 (CALL (PRINT '虫の大きさ? )) (BIND ?D (READ)) (CALL (PRINT '虫食い跡? ))
 (BIND ?E (READ)) (CALL (PRINT '葉の変色? )) (BIND ?F (READ))
 (CALL (PRINT '葉の異常? )) (BIND ?G (READ))
 (CREATE 症状 花の症状 (植物名 ?A) (虫の付着 ?B) (虫の種類 ?C) (虫の大きさ ?D)
  (虫食い跡 ?E) (葉の変色 ?F) (葉の異常 ?G))
 ((PUT-VARIABLE ONLY) WMINST1 '病名 '風邪))

          (略)

(DEFRULESET (CLASSIFICATION (:STRATEGY GOAL)))

(DEFRULE (CLASSIFICATION C1 (:PRIORITY 0))
 (FRAME (症状 ? (植物名 ?NAME) (虫の付着 有))) -->
 (CREATE 結論 虫 (植物名 ?NAME) (分類 虫)) (CALL-RULESET INSECT))

(DEFRULE (CLASSIFICATION C2 (:PRIORITY 0))
 (FRAME (症状 ? (植物名 ?NAME) (虫の付着 無) (虫食い跡 無) (葉の変色 有))) -->
 (CREATE 結論 病気 (植物名 ?NAME) (分類 病気)) (CALL-RULESET DISEASE))

          (略)

(DEFRULESET (INSECT (:STRATEGY GOAL)))

(DEFRULE (INSECT I1 (:PRIORITY 1))
 (FRAME
  (症状 ? (虫の種類 粒状) (虫の大きさ ?Y!(<= ?Y 1)) (虫食い跡 無) (葉の変色 有))
)
 (FRAME (結論 ?X (分類 虫) (原因 NIL))) --> (MODIFY ?X (原因 ハダニ))
 (CALL (PRINT '原因はハダニによる虫害)))

(DEFRULE (INSECT I2 (:PRIORITY 1))
 (FRAME (症状 ? (虫の種類 粒状) (虫の大きさ ?Y!(> ?Y 1)) (虫食い跡 無)))
 (FRAME (結論 ?X (分類 虫) (原因 NIL))) --> (MODIFY ?X (原因 アブラムシ))
 (CALL (PRINT '原因はアブラムシによる虫害))
 (CALL ((PUT-VARIABLE-ONLY GET-VARIABLE) WMINST1)))
```

```
          (略)

(DEFRULESET (DISEASE (:STRATEGY GOAL)))

(DEFRULE (DISEASE D1 (:PRIORITY 1))
 (FRAME (症状 ? (植物名 バラ) (葉の異常 黒斑)))
 (FRAME (結論 ?X (分類 病気) (原因 NIL))) --> (MODIFY ?X (原因 黒星病))
 (CALL (PRINT '原因は黒星病)))

(DEFRULE (DISEASE D2 (:PRIORITY 1))
 (FRAME (症状 ? (植物名 バラ) (葉の異常 凸凹斑)))
 (FRAME (結論 ?X (分類 病気) (原因 NIL))) --> (MODIFY ?X (原因 サビ病))
 (CALL (PRINT '原因はサビ病)))

          (略)

(DEFRULESET (MONITOR (:STRATEGY GOAL)))

(DEFRULE (MONITOR M1 (:PRIORITY 5)) --> (CALL-RULESET GENERATE))

(DEFRULE (MONITOR M2 (:PRIORITY 4)) -->
 (CALL (SETQ X (GET-SUPER '症状))) (CALL (PRINT X))
 (CALL (SETQ Y (GET-SUB '結論))) (CALL (PRINT Y)) (CALL-RULESET CLASSIFICATION))

(DEFRULE (MONITOR M3 (:PRIORITY 3)) --> (CALL (TERPRI))
 (CALL (FORMAT T "~% もう一度診断を行いますか? (y/n)
:")) (BIND ?P (READ))
 (CREATE 繰り返し 入力 (INPUT ?P)))

          (略)

(DEFRULESET (TEST (:STRATEGY GOAL)))

(DEFRULE (TEST T1 (:PRIORITY 0)) (NOT (FRAME (症状 ?))) -->
 (CALL (UNDEFCLASS '症状))
 (CALL
  (CREATE-INSTANCE '木 NIL '科 'バラ '色 'ピンク '香り 'シトラス)))

(CREATE-INSTANCE '木 NIL '科 'バラ '色 'ピンク '香り 'シトラス)

(DEFRULE (TEST T2 (:PRIORITY 0)) (NOT (FRAME (結論 ?))) -->
 (CALL (UNDEFCLASS '結論)))
```

図 21　ワーキングメモリ型からフレーム型への変換例（変換後）

いて述べた。フレーム型からワーキングメモリ型への変換に関して，インスタンス生成に関するLISP関数が一つ実現できなかったことを除き，予定した対象について相互に変換することが確認できた。

本変換プログラムは，ソーステキストレベルの知識表現で変換すること，また変換に際しては，二つの表現方法のいずれもリスト表現をとることに着目してLISP言語(COMMON LISP)を使用したこと，を特徴としている。

今後は，KBMSとOPS 83[3]のようにツールとしては異なるが類似の設計思想に基づく知識表現方法相互の変換等が課題と考えられる。

文　献

1) 寺野隆雄：”知識ベースシステム開発ツールにおけるベンチマーク”，情報処理，vol. 31. No. 3, pp. 352-360 (1990).
2) NTT ソフト編：”KBMS/PC リファレンスマニュアル”，NTT ソフトウェア発行 (1988).
3) G. L. フォーギー：”人工知能用言語 OPS 83”，パーソナルメディア (1986).