ファジィ制約を備えた知識表現システムとそのISLISPによる実現

森谷俊洋† 伊藤貴康†

フレームとプロダクションシステムの機能を備えた高機能高性能知識表現システムとしてKL-ONEとその拡張であるLOOMなどがある．それらの基本機能を備え，特に，知識の制約表現機能を充実させた知識表現言語KRS-FZを設計し，その処理系を実現した．KRS-FZの処理系は，ワークステーション上だけでなく高性能パソコン上でも稼働するISLISP処理系TISLを用いて実現されたポータブルな高機能高性能知識表現システムである．KRS-FZにおいては，知識の制約表現として，確定制約だけでなくファジィ制約を導入し，曖昧な知識や概念の扱いを可能としている．概念・関係の解釈に，開世界仮説による解釈だけでなく，閉世界仮説による解釈を指定することによって非単調推論も実現している．このような多様な知識表現に基づく推論機構の効率の良い実現のために分類器を用い，その再帰的関数による仕様記述を与え，ISLISPにより実現した．また，KRS-FZシステムとその分類器による実験評価を行い，システムが優れた実行性能を有することを示した．

A Knowledge Representation System with Fuzzy Constraints and Its Implementation in ISLISP

Toshihiro Moriya† and Takayasu Ito†

Several sophisticated and efficient knowledge representation systems, such as KL-ONE and LOOM, have been developed, integrating *frames* and *production systems*. We designed a knowledge representation language, called the KRS-FZ, with the basic mechanisms realized in the previous systems, enriching multiple constraints for knowledge representation. The KRS-FZ processor is implemented using an ISLISP processor TISL on a high performance PC. The KRS-FZ language is equipped with not only an extended set of definitive constraints but also a set of fuzzy constraints to allow describing ambiguous knowledge and concepts. Use of the classifier enables the KRS-FZ inference mechanism to be highly efficient in monotonic and non-monotonic reasoning under multiple knowledge representations. Also, a recursive specification of the classifier is given so as to allow an implementor to realize it easily into his knowledge representation system.

1. はじめに

高機能高性能知識表現システムの研究がDARPA HPKB（High-Performance Knowledge Bases）プロジェクトに刺激されさかんに行われ[1]，また企業などにおける応用も進められている[2]．その代表的なものに，フレームとプロダクションシステムに基づき，オブジェクトを中心として対象世界の記述を可能としたKL-ONE[3]と，いわゆるKL-ONEファミリィに属するLOOM[4]，CLASSIC[5]，KRIS[6]などがある．

本研究では，フレームとプロダクションシステムの機能を基にし，知識の開世界仮説と閉世界仮説での解釈，確定制約による知識表現だけでなく曖昧な知識のファジィ制約表現による取扱いも可能とする知識表現言語KRS-FZ（Knowledge Representation System with FuZzy constraints）を設計し，処理系を実現した．KRS-FZ言語においては，上記のKL-ONEファミリィの知識表現システムに比べ知識の制約表現機能が拡張され，従来のシステムでは取扱えなかった知識表現の扱いも可能となっている．また，分類器と呼ばれる機構を利用することにより，多様な知識表現に基づく各種推論機能が効率良く実現されている．

KRS-FZ処理系の実現は，ISO標準Lisp言語ISLISP[7]の処理系TISL[8)~10)]を用いて行われたが，KRS-FZシステムはISLISPによる大規模アプリケーションの初めての試みでもある．またその実装はIS-

† 東北大学大学院情報科学研究科
Department of Computer and Mathematical Sciences, Graduate School of Information Sciences, Tohoku University

LISP の Lisp 機能とオブジェクト指向機能，TISL において導入されたパッケージ機能を利用することにより簡潔に行われている．

本論文では，2 章でフレームとプロダクションシステムに基づく知識表現システムの基本機能を備え，かつ，曖昧な知識の取扱いを可能とするファジィ知識表現機能を備えた KRS-FZ 言語を与える．3 章では KRS-FZ における各種推論機能について説明した後，推論機能の効率的実行に重要な役割を果たす分類器の機能説明とその再帰的仕様記述を与える．4 章では ISLISP とその処理系 TISL を用いて実現された KRS-FZ 処理系の概要を説明し，また，KRS-FZ を用いた例と評価実験結果について述べる．5 章では他研究との比較を行う．6 章は結言である．

2. 知識表現言語 KRS-FZ

KL-ONE ファミリィの知識表現システム[3] ではフレームとプロダクションシステムを用いて対象世界の知識と関係を記述する．また，それらの利用にあたり，制約表現によって知識間の関係を修飾し探索空間を限定するとともに，多様な知識表現の下での推論を効率良く行うために分類器（classifier）が用いられる．

KRS-FZ 言語は，KL-ONE ファミリィの言語の基本的な知識表現機能を備えた汎用的な知識表現言語である．特に，対象世界における知識の表現に必要な制約表現機能として，既存の確定制約機能を拡張するとともに，曖昧な性質を持った知識の扱いを可能とするファジィ制約機能を備えているのが大きな特徴である．曖昧さを持った知識の扱いを可能とすることは，知識表現システムの利用において重要である．ファジィ論理は，曖昧さを扱う 1 つの有効なアプローチであり，KRS-FZ は，ファジィ機能を導入した知識表現システムの初めての試みでもある．

2.1 節において，フレームとプロダクションシステムに基づく知識表現システムの基本機能を 6 つに分類し，それらを備えた知識表現言語 KRS-FZ の基本機能と構文を与える．2.2 節において，知識ベースを利用した推論を行うにあたって必要かつ有用となる知識の制約表現を与える．KRS-FZ 言語の処理系については，4 章で説明する．

2.1 知識表現言語 KRS-FZ の基本機能と構文

知識表現言語は，知識ベースに基づく知識指向プログラミングを行うための言語である．フレームとプロダクションシステムに基づく知識表現言語としては次のような機能を備えている必要がある．

(1) 対象世界のモデルであるオントロジーを知識ベースに構築する機能
(2) 事実の主張や撤回を知識ベースを基に行う機能
(3) 知識ベースに対する質問と検索の機能
(4) 対象世界におけるタスクを記述する機能
(5) 対象世界における条件付き行動を記述するためのプロダクションルールを記述する機能
(6) 知識ベースを利用した推論を有効に行うための知識の制約表現機能

知識の制約表現については 2.2 節で述べる．知識表現機能を用いて記述された知識に基づく，多様な推論実現機構である分類器については 3 章で述べる．

KRS-FZ 言語は上述の (1)〜(6) の機能を備えた汎用的な知識表現言語である．以下に，これらの機能を実現する KRS-FZ の構文について説明する．なお，構文の設計にあたっては，概念や関係の定義形式，コロン（：）を用いたキーワードの指定，演算子を前置する式の表記法（Cambridge Notation），リスト形式でのデータの表記などに LISP 流の記法を用いている．言語の処理系を LISP 言語の ISO 標準 ISLISP を用いて作成することを考え，内部表現はリスト形式としている．

2.1.1 対象世界のモデルであるオントロジーの記述

KRS-FZ 言語では，対象世界のモデルであるオントロジーを概念とそれらの間の関係として記述し，知識ベースに登録する．概念は具体的事例が属するクラスであり，関係は概念間の関係を表す述語である．概念の要素である事例間の関係はロール（role）と呼ばれる．

(1) 概念の定義

概念の定義には,次の **defconcept** 構文を用いる：

(**defconcept** *c-name c-pred c-expr attr*)

c-name は定義しようとしている概念の名前である．概念 *c-name* は述語 *c-pred* によって式 *c-expr* として定義され，その解釈は属性 *attr* によって決定される．*c-pred*，*c-expr*，*attr* は次のように記述される．

c-pred ::= :is | :implies | :defaults

c-expr ::= *C-name*
| *constraint-expr*
| (:and $c\text{-}expr_1 \cdots c\text{-}expr_n$)
| (:or $c\text{-}expr_1 \cdots c\text{-}expr_n$)
| (:not *c-expr*)

attr ::= λ | :characteristics :closed-world

たとえば，(defconcept c-name :is c-expr) は，概念 c-name が c-expr である（:is）ことを定義している．(defconcept c-name :implies c-expr) は，c-name が c-expr を論理的に含む（:implies）概念であることを

定義している．(defconcept c-name :defaults c-expr) は，概念 c-name が適用される時点の知識ベースと c-expr が矛盾しない場合にのみ c-expr であると定義されるデフォルト定義である．

C-name は定義済の概念の名前である．制約表現 *constraint-expr* については 2.2 節において詳述する．

c-expr には論理演算（:and, :or, :not）を使用できる．(defconcept c-name c-pred (:and c-expr$_1$ ··· c-expr$_n$)) の場合，c-name は各 c-expr の連言（:and）として c-pred により定義される概念である．(:or c-expr$_1$ ··· c-expr$_n$) は各 c-expr の選言（:or）を，(:not c-expr) は c-expr の論理的否定（:not）を意味する．

概念の解釈は属性 *attr* によって指定する．(defconcept c-name :characteristics :closed-world) は，c-name が閉世界仮説（:closed-world）で解釈される概念であることを意味している☆．*attr* が指定されない場合（すなわち，λ である場合），定義した概念は開世界仮説で解釈される．

概念を定義する際，その概念を包含する概念と性質の 2 つを考慮する．性質の記述には後述の制約表現を用いる．たとえば，概念「学校」には「生徒が 1 人以上存在する」という性質が考慮される．また，「男子校」は「学校」の特殊な場合であり，「生徒全員が男子である」という性質を持つ．これは次のように記述できる．

```
(defconcept School :implies (:at-least 1 has-student))
(defconcept BoysSchool
  :is (:and School (:all has-student Male)))
```

`(:at-least 1 has-student)` は述語 has-student (x) が真となる x が 1 つ以上あることを述べ，`(:all has-student Male)` は has-student(x) が真となるすべての x が Male であることを述べる制約表現である．`School` は`(:at-least 1 has-student)` を含む概念として定義され，概念`BoysSchool` は`School` と`(:all has-student Male)` の連言として定義される．上述の定義例では *attr* の指定がないので，`School` と`BoysSchool` は開世界仮説で解釈される．閉世界仮説での解釈を指定した概念の例として次の`BoysSchool-CW` を考える．

```
(defconcept BoysSchool-CW
  :is (:and School (:all has-student Male))
  :characteristics :closed-world)
```

☆ 知識ベースを基にした推論によって真であることが示されない命題は偽であると見なすことを閉世界仮説と呼んでいる．これに対して，真あるいは偽であることが示されない命題は不明であると見なすことを開世界仮説と呼んでいる．

ここで，『A 校は「男子校」に属するか』という質問を開世界仮説と閉世界仮説で行った場合について考える．

1) 開世界仮説での質問(`ask (BoysSchool school-A)`)☆☆に対し，`schoolA` が`BoysSchool` に属することが示せない場合，unknown が返される．
2) 閉世界仮説での質問(`ask (BoysSchool-CW schoolA)`) に対し，`schoolA` が`BoysSchool-CW` に属することが示せない場合，false が返される．

(2) 関係の定義

関係の定義には，次の **defrelation** 構文を用いる：

(**defrelation** *r-name r-pred r-expr r-domain r-range attr*)

関係 *r-name* は，述語 *r-pred*，式 *r-expr*，および，定義域 *r-domain*，値域 *r-range* を用いて定義され，その解釈は属性 *attr* によって決定される．*r-pred*，*r-expr*，*r-domain*，*r-range* は次のように記述される．

r-pred ::= :is | :implies
r-expr ::= *R-name* | (:and *r-expr*$_1$ ... *r-expr*$_n$)
r-domain ::= :domain *C-name*
r-range ::= :range *C-name*

たとえば，(defrelation r-name r-pred r-expr :domain C-name$_1$:range C-name$_2$) の場合，関係 r-name は，定義域が概念 C-name$_1$，値域が概念 C-name$_2$ である式 r-expr として r-pred によって定義される関係である．:domain と :range は，それぞれ，定義域と値域を指定するキーワードである．

「友人」関係を，定義域，値域ともに「人間」である関係として，次のように記述できる．

```
(defrelation friend-of
    :is :domain Person :range Person)
```

また，「男友達」(`malefriend-of`) は，定義域が「友人」(`friend-of`) であり，値域が「男性」(`Male`) である関係として次のように記述できる．

```
(defrelation malefriend-of
    :is friend-of :range Male)
```

2.1.2 対象世界の事実の主張と撤回

対象世界を記述する具体的事実を知識ベースに登録することを事実の主張と呼び，主張された事実を知識ベースから消去することを事実の撤回と呼んでいる．

事実の主張には **tell** 構文を用い，事実の撤回には **forget** 構文を用いる：

(**tell** *prop*$_1$ ... *prop*$_n$)
(**forget** *prop*$_1$ ... *prop*$_n$)

(tell prop$_1$ ... prop$_n$) の場合，各命題 prop を知識ベースに主張し，(forget prop$_1$ ... prop$_n$) の場合，各

☆☆ **ask** は知識ベースに質問を行う構文であり，2.1.3 項で述べる．

命題 prop を知識ベースから撤回する．命題 *prop* は次のように記述される．

prop ::= (*C-name i-name*) | (*R-name i-name*$_1$ *i-name*$_2$)
| (:about *i-name constraint-expr*) | (:not *prop*)
| (:hit-degree *C-name i-name* N_{01})

(C i) は事例 i が概念 C に属するという命題であり，(R i_1 i_2) は i_1 が i_2 と R の関係にあるという命題である．たとえば，「太郎は学生であり，一郎の友人である」という事実の主張は次のように記述できる．

```
(tell (Student taro) (friend-of taro ichiro))
```

(:about i constraint-expr) は事例 i に制約表現 constraint-expr が成立するという命題である．(:not prop) は prop の否定命題である．

(:hit-degree C i N_{01}) は，2.2 節で述べるファジィ概念のための構文であり，ファジィ概念 C の事例 i に対する適合度が N_{01} ($N_{01} \in [0,1]$) であるという命題である．

事実の主張・撤回後に（2.1.5 項で述べる）プロダクションルールに基づくルールマッチングを開始し，知識ベースの状態を検査するには，次の **tellm** あるいは **forgetm** を用いる：

(**tellm** *prop*$_1$ ... *prop*$_n$)
(**forgetm** *prop*$_1$ ... *prop*$_n$)

(tellm prop$_1$ ... prop$_n$) は，各 prop を知識ベースに主張した後，ルールマッチングを開始する．(forgetm prop$_1$ ... prop$_n$) は，各 prop のうち知識ベースに存在する事実を撤回した後，ルールマッチングを開始する．

また，事例 *i-name* に関する全事実を撤回する構文として，(**forget-all-about** *i-name*) が用意されている．

2.1.3 知識ベースに対する質問と検索

知識ベースに対する質問と検索は，**ask** と **retrieve** からなるクエリ言語を用いて行う．

(1) 知識ベースへの質問

知識ベースへの質問には，次の **ask** 構文を用いる：

(**ask** *query ask-option*)

クエリ表現 *query* は次のように記述される．

query ::= (*C-name i-name*) | (*R-name i-name*$_1$ *i-name*$_2$)
| (:about *i-name constraint-expr*)
| (:and *query*$_1$ ... *query*$_n$)
| (:or *query*$_1$ ... *query*$_n$)
| (:not *query*) | (:fail *query*)

たとえば，(ask (C i)) は事例 i が概念 C に属するかを知識ベースに問い，(ask (:and q_1 ... q_n)) は各クエリ q_i の連言を知識ベースに問う．q_i は，区間 [0,1] のファジィ値をとることが許され，$q_i \in [0,1]$ とすると，$and[q_1,\ldots,q_n]=min[q_1,\ldots,q_n]$ と解釈される．選言:or と論理否定:not は，それぞれ，$or[q_1,\ldots,q_n]=max[q_1,\ldots,q_n]$ と $not[q] = 1 - q$ になる．(:fail q) は，クエリ q の評価による失敗としての否定を意味し，$fail[q] = (if\ q = 1\ then\ 0\ else\ 1)$ となる．

ask は *query* の真偽値をデフォルトでは true を t に，false と unknown を nil に変換した 2 値として返すが，*ask-option* の記述によって返り値を変更できる：

ask-option ::= λ | :three-valued-p | :degree-p

(ask query :three-valued-p) は query の真偽値として，true, false, unknown の 3 値を返す．(ask query :degree-p) は query の評価値 v（$v \in [0,1]$）を返す．

:not と:fail の 2 種類の否定は厳密に区別される．「A 校は男子校ではないか？」という質問について，次の 2 つの例を考える．

1) :not を用いた例

```
(ask (:not (BoysSchool schoolA)))
```

2) :fail を用いた例

```
(ask (:fail (BoysSchool schoolA)))
```

1) の例は，`schoolA` が `BoysSchool` に属さない事実を知識ベースから導出できる場合に true を返す．一方，2) の例は，`schoolA` が `BoysSchool` に属する事実を知識ベースから導出できない場合に true を返す．

(2) 知識ベースへの検索

知識ベースへの検索には，次の **retrieve** を用いる：

(**retrieve** (*?var*$_1$ ⋯ *?var*$_n$) *query*)

query 中の検索対象となる事例に "?" を付して検索項目を指定するとともに，検索項目を (*?var*$_1$ ⋯ *?var*$_n$) と列挙する．**retrieve** 構文の返り値は *?var* に該当する事例名のリストである．

2.1.4 アクションとメソッドの定義

対象世界におけるタスクをアクションとして記述し，実行することができる．

(1) アクションの定義☆

アクションの定義には **defaction** 構文を用いる：

(**defaction** *a-name a-params a-filter-seq a-missing*)

a-name は定義しようとしているアクションの名前である．*a-params* は *a-name* の仮引数リスト，*a-filter-seq* はアクション a-name が呼び出された際に適用可能なメソッドの集合から実行メソッドを選択するフィル

☆ KRS-FZ におけるアクションとメソッドはオブジェクト指向言語における包括関数とメソッドに相当する形で実現されている．なお，包括関数は呼び出されたとき，その振舞いが引数のクラスによって決定される関数である．

タの列，*a-missing* は適用可能なメソッドが存在しない場合の処理であり，それぞれ次のように記述される．

a-params ::= (*?var*$_1$ ⋯ *?var*$_n$)
a-filter-seq ::= :filters (*a-filter*$_1$ ⋯ *a-filter*$_n$)
a-filter ::= :most-specific | :last-one | :select-one
| :select-all | :warning | :error
a-missing ::= :missing-method *a-miss-op*
a-miss-op ::= :no-op | :warning | :error

たとえば，(defaction a-name a-params :filters (a-filter$_1$ ⋯ a-filter$_n$) :missing-method a-miss-op) の場合，a-name は仮引数リストに a-params，フィルタ（:filters）にフィルタの系列 (a-filter$_1$ ⋯ a-filter$_n$) を用い，適用可能なメソッドが存在しない場合の処理（:missing-method）に a-miss-op を持つと定義されるアクションである．アクション a-name が呼び出されると，それとともに導出される適用可能なメソッドの集合に対し，ただ 1 つのメソッドが残るまでフィルタが a-filter$_1$ から a-filter$_n$ の順に適用される．適用可能なメソッドが存在しない場合には，a-miss-op に指定した処理が実行される．

フィルタ *a-filter* には 6 種類（:most-specific，:last-one，:select-one，:select-all，:warning，:error）がある．:most-specific は，適用条件が最も詳細なメソッドを選択するフィルタである．:last-one は最も新しく定義されたメソッドを，:select-one はランダムに 1 つのメソッドを，:select-all はすべてのメソッドを通すフィルタである．:warning は警告文を出力したうえですべてのメソッドを通す．:error はエラーを発生して処理を中断する．

a-miss-op には 3 種類（:no-op，:warning，:error）があり，:no-op は処理を行わないアクションである．

例．
```
(defaction eval-result (?x)
    :filters (:most-specific :last-one)
    :missing-method :error)
```

この例の場合，アクション `eval-result` が引数とともに呼び出されると，適用可能なメソッドの集合から条件部が最も詳細なメソッドが抽出され，抽出されたメソッドが 1 つのみならばそのメソッドの応答部を，複数が存在する場合にはその中で最も新しく定義されたメソッドの応答部を評価する．適用可能なメソッドが存在しない場合にはエラーを発生させる．

(2) メソッドの定義

メソッドの定義には **defactmethod** 構文を用いる：

(**defactmethod** *a-name a-params a-cond a-act*)

a-name は定義しようとしているメソッドの名前である．*a-cond* は引数に対するメソッド *a-name* の適用条件，*a-act* は *a-name* が適用される場合に評価される応答部であり，それぞれ次のように記述される．

a-cond ::= :situation *query*
a-act ::= :response (*ISLISP-expr*)

(defactmethod a-name a-params :situation query :response (ISLISP-expr)) の場合，a-name は仮引数リストに a-params をとり，条件部に query，応答部に ISLISP 式 ISLISP-expr を持つと定義されるメソッドである．たとえば，「点数が 80 点以上ならば優等生であると評価する」という手続きは次のように記述できる．

```
(defactmethod eval-result (?x)
  :situation (:about ?x (>= score 80))
  :response ((tell (Honor ?x))))
```

(3) タスクの実行

タスクの実行は (**perform** *task*) と記述され，タスク *task* はアクションとその引数となる事例からなる．

task ::= (*a-name arg*$_1$ ⋯ *arg*$_n$)

たとえば，「太郎の結果を評価する」というタスクの実行は上述の `eval-result` を用いて次のように記述される．

```
(perform (eval-result taro))
```

2.1.5 対象世界のルールを記述する機能

対象世界の条件付き行動は，条件部と行動部からなるプロダクションルールを用いて記述する[11]．ルールマッチングは **tellm** による事実の主張後，あるいは，**forgetm** による事実の撤回後に行われ，知識ベースの状態が条件部を満たすとき，そのルールは適用可能と判断されプロダクションルールの競合集合に追加される．競合集合において条件部が最も詳細なルールが実行ルールに選択され，行動部が評価される．プロダクションルールの定義には，次の **defproduction** 構文を用いる：

(**defproduction** *p-name p-cond p-act*)

p-name は定義しようとしているプロダクションルールの名前である．*p-cond* は *p-name* の条件部，*p-act* は *p-name* の行動部であり，次のように記述される．

p-cond ::= :when *query*
p-act ::= :do (*ISLISP-expr*)
| :perform (*task*$_1$ ⋯ *task*$_n$)
| :schedule (*task*$_1$ ⋯ *task*$_n$) *priority*
priority ::= :priority *queue*
queue ::= :high | :low

(defproduction p-name :when query p-act) の場合，p-name は条件部（:when）に query，行動部に

p-act を持つと定義されるプロダクションルールである．

行動部 *p-act* には，ISLISP 式の評価，タスクの実行，あるいは，タスク管理を記述できる．行動部が:do (ISLISP-expr) の場合，ISLISP 式 ISLISP-expr が実行される．行動部が:perform (task$_1$ ⋯ task$_n$)) の場合，タスクを task$_1$ から task$_n$ の順に実行する (:perform)．たとえば，「12 時ならばチャイムを鳴らしなさい」というルールは次のように記述できる．

```
(defproduction chime-rule
  :when (hour now 12) :perform ((sound-chime)))
```

行動部が:schedule (task$_1$ ⋯ task$_n$) :priority queue の場合，優先度 (:priority) が queue であるキューにタスクを task$_1$ から task$_n$ の順に追加し，ルールマッチング終了後にキューへの追加順にタスクが実行される．キューの優先度 *queue* には:high と:low の 2 つがあり，この順で保存されているタスクが実行される．

以上に述べた機能と 2.2 節で述べる制約表現に加えて，KRS-FZ 言語では，知識ベースを初期化する **clear-kb** 構文を備えている：

(**clear-kb** *partitions*)

partitions ::= λ | :partitions (*part*$_1$ ⋯ *part*$_n$)

part ::= :concepts | :relations | :instances

:concepts, :relations, :instances は，知識ベースに登録されている概念，関係，事例の集合を表す．たとえば，(clear-kb :partitions (:concepts)) は，知識ベースに登録されたすべての概念を消去する．*partitions* が指定されない (clear-kb) の場合には，すべての概念，関係，事例が消去される．

2.2 KRS-FZ における知識の制約表現

概念に属すべき事例，事実の主張・撤回やクエリ表現において記述される事例に対する修飾子が制約表現である．KL-ONE ファミリィの言語では述語論理に基づく確定制約が用いられている．KRS-FZ 言語では，確定制約を拡張するとともに，曖昧な性質を持った知識の表現を可能とするためにファジィ制約を導入した．

2.2.1 確定制約

確定制約は，個数制約，領域制約，値制約からなる．制約対象の事例を i，関係を R，i と R の関係にある事例の集合を I_v とすると，個数制約は I_v の要素の個数 $|I_v|$，領域制約は I_v の要素が属する概念，値制約は I_v の要素を制約する．$|I_v|$ を i が属するロール R の個数，I_v の各要素を i が属するロール R の値と呼ぶ．

表 1 に KRS-FZ 言語において使用される確定制約を与えた．表 1 において，個数の区間制約**:between**，領域の条件制約**:cond**，値の領域制約**:region** は，KRS-FZ 言語において導入した構文である．たとえば，区間制約**:between** を用いて，「学生数が 35〜40 人である」という表現は次のように記述される．

```
(:between 35 40 has-student)
```

また，「全生徒が，男子校では男性，女子校では女性」という表現は**:cond** を用いて次のように記述される．

```
(:cond (?x) (((BoysSchool ?x) Male)
             ((GirlsSchool ?x) Female)) has-student)
```

制約対象の事例に`?x` が束縛され，`?x` が`BoysSchool` に属する事例は`Male` であって`has-student` の値となる事例であり，`GirlsSchool` に属する事例は`Female` であって`has-student` の値となる事例であることを述べている．

また，値の領域制約 **:region** を用いて，「数学の得点は 80〜90 点，理科の得点は 60〜70 点の領域を占める」という表現は次のように記述される．

```
(:region (math-score 80 90) (science-score 60 70))
```

2.2.2 ファジィ制約

事例が曖昧な性質を有する場合，KRS-FZ 言語ではファジィ論理に基づくファジィ制約を用いて扱うことができる．また，ファジィ制約の導入とともに，概念の適合度 [0,1] を各事例に与えており，適合度がしきい値 $T \in (0,1]$ 以上の事例はその概念に属すると判断される．しきい値 T は (**set-minimum** T) で与えられる．

表 1 KRS-FZ 言語における確定制約
Table 1 Definitive constraints in KRS-FZ language.

種類	構文	解釈
個数	(:at-least I R)	$R(i_1)\wedge\ldots\wedge R(i_n) \wedge n \geq I$
	(:at-most I R)	$R(i_1)\wedge\ldots\wedge R(i_n) \wedge n \leq I$
	(:exactly I R)	$R(i_1)\wedge\ldots\wedge R(i_n) \wedge n = I$
	(:between I_1 I_2 R)	$R(i_1)\wedge\ldots\wedge R(i_n) \wedge n\in[I_1,I_2]$
領域	(:all R C)	$\forall x\ (R(x) \wedge x \in C)$
	(:some R C)	$\exists x\ (R(x) \wedge x \in C)$
	(:the R C)	$R(x) \wedge x \in C$ となる x がただ 1 つ存在する
	(:cond (?var)	if q_1(?var)
	((q_1 C_1)	then $\forall x\ (R(x) \wedge x \in C_1)$
	... (q_n C_n)) R)	... else if q_n(?var) then $\forall x\ (R(x) \wedge x \in C_n)$
値	(:filled-by R i)	R(i)
	(*rel* R N)	$R(x)\wedge(x\ rel\ N)$ となる x
	rel ::= <|>|<=|>=|=|/=	
	(:region	$(R_1(x_1)\wedge x_1\in[N_{11},N_{12}])\wedge$
	(R_1 N_{11} N_{12})	$\ldots\wedge(R_n(x_n)\wedge x_n\in[N_{n1},N_{n2}])$
	... (R_n N_{n1} N_{n2}))	となる $(x_1,\ldots,x_n)$

R：関係，I：非負整数，C：概念，q：クエリ表現，i：事例，N：実数

表 2 KRS-FZ 言語におけるファジィ制約
Table 2 Fuzzy constraints in KRS-FZ language.

種類	構文	解釈
個数	(:fuzzy-number {v} F R)	$R(i_1)\wedge\ldots\wedge R(i_n) \wedge F_v(n)\geq T$
	(:fuzzy-between {v_1} F_1 {v_2} F_2 R)	$R(i_1)\wedge\ldots\wedge R(i_n) \wedge (\forall y_1,y_2\ F_{v1}(y_1)\geq T \wedge F_{v2}(y_2)\geq T \wedge n\in[y_1,y_2])$
領域	(:fuzzy-all R {v} F C)	$\forall x\ (R(x) \wedge F_v(C_{mem}(x))\geq T)$
	(:fuzzy-some R {v} F C)	$\exists x\ (R(x) \wedge F_v(C_{mem}(x))\geq T)$
	(:fuzzy-the R {v} F C)	$R(x) \wedge F_v(C_{mem}(x))\geq T$ となる x がただ 1 つ存在する.
	(:cond (?var)	if q_1(?var) then $\forall x\ (R(x) \wedge F_{v1}(C_{mem1}(x))\geq T)$
	((q_1 {v_1} F_1 C_1)	...
	... (q_n {v_n} F_n C_n)) R)	else if q_n(?var) then $\forall x\ (R(x) \wedge F_{vn}(C_{memn}(x))\geq T)$
値	(:filled-by R *f-val*)	$R(x) \wedge x \in$ *f-val* となる x
	f-val ::= (:fuzzy-value {v} F)	*f-val* = {x| $F_v(x)\geq T$}
	(*rel* R *f-val*)	$R(x) \wedge (\forall y\ y\in$*f-val* $\wedge$ (x *rel* y)) となる x
	rel ::= <|>|<=|>=|=|/=	
	(:region (R_1 *f-val*$_{11}$ *f-val*$_{12}$)	$(R_1(x_1) \wedge (\forall y_1,y_2\ y_1\in$*f-val*$_{11}\wedge\ y_2\in$*f-val*$_{12} \wedge x_1\in[y_1,y_2])) \wedge$
	... (R_n *f-val*$_{n1}$ *f-val*$_{n2}$))	$\ldots\wedge (R_n(x_n) \wedge (\forall y_1,y_2\ y_1\in$*f-val*$_{n1}\wedge\ y_2\in$*f-val*$_{n2} \wedge x_n\in[y_1,y_2]))$ となる $(x_1,\ldots,x_n)$
比率	(:fuzzy-quantifier {v} F R C)	$(R(i_1)\wedge i_1\in C)\wedge\ldots\wedge(R(i_k)\wedge i_k\in C) \wedge R(i_{k+1})\wedge\ldots\wedge R(i_n) \wedge F_v(k/n)\geq T$

v：言語ヘッジ，F：メンバシップ関数，R：関係，F_v：v によって修飾されたメンバシップ関数
T：しきい値，i：事例，C：概念，C_{mem}：概念 C の適合度を返す関数，q：クエリ表現

表 2 に KRS-FZ 言語において使用されるファジィ制約を与えた．ファジィ制約は確定制約を拡張して与えられている．制約対象の事例を i，関係を R，i と R の関係にある事例の集合を I_v とする．このとき，ファジィ個数制約は I_v の要素の個数 $|I_v|$，ファジィ領域制約は I_v の要素に対する概念の適合度，ファジィ値制約は I_v の各要素が，言語ヘッジ v によって修飾されたメンバシップ関数 F_v が与える数値となる．

例として,「生徒数はそれほど少なくはない」という曖昧な表現の記述を考える．この場合の「少ない」は特定の要素が存在する個数を表すため，ファジィ個数制約**:fuzzy-number** を用いる.「少ない」にメンバシップ関数 few,「少ない」を修飾する「それほど〜ない」に後述の言語ヘッジ:not-very を用いると，上述の表現は次のように記述される．

```
(:fuzzy-number :not-very few has-student)
```

has-student の値となる事例の個数を n，言語ヘッジ :not-very によって修飾されたメンバシップ関数を $few_{notvery}$，しきい値を T とすると，**:fuzzy-number** 構文を用いて $few_{notvery}(n)\geq T$ が成立することを述べている．

メンバシップ関数は，次の **defmembership** 構文を用いて定義する：

(**defmembership** *f-name f-body*)

f-body ::= (:bell p_1 p_2 p_3 p_4) | (:l-trpzd p_1 p_2 p_3 p_4)
| (:r-trpzd p_1 p_2 p_3 p_4) | (:m-trpzd p_1 p_2 p_3 p_4 p_5 p_6)

:bell, :l-trpzd, :r-trpzd, :m-trpzd は，**図 1** の各関数に相当する．たとえば，(defmembership f-name (:bell p1 p2 p3 p4)) の場合，メンバシップ関数 f-name は数値パラメータ p1〜p4 を持つ:bell 関数と定義される．

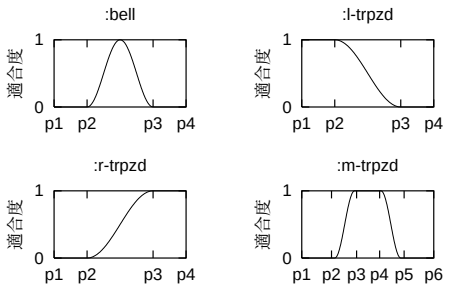


図 1 KRS-FZ において使用可能なメンバシップ関数
Fig. 1 Membership functions in KRS-FZ.

また，言語ヘッジ[12] *v-hedge* は次の 5 つからなる．

v-hedge ::= :very | :more-or-less | :not
| :not-very | :not-more-or-less

:fuzzy-quantifier は，ファジィ論理におけるファジィ限定命題[13] の考え方を参考にして導入した限定子で，特定の事例が存在する比率を表す制約表現の記述に用いられる．(:fuzzy-quantifier v F R C) は，制約対象と R の関係にある事例の個数と，そのうち概念 C に属する事例の個数との比が，言語ヘッジ v によって修飾されたメンバシップ関数 F が与える値となる．例として,「全生徒のうち，男性は非常に少ない」という曖昧な表現の記述を考える．この場合の「少ない」は,「男性」が存在する比率を意味し,「少ない」を表すメンバシップ関数 r-few を，たとえば次のように定義する．

```
(defmembership r-few (:l-trpzd 0 0.1 0.5 1))
```

関数:l-trpzd にパラメータ (0 0.1 0.5 1) が与えら

れ，メンバシップ関数 $y = r\text{-}few(x)$ は，比率が 1 割以下（$x \leq 0.1$）では “完全に少なく”（$y = 1$），比率の増加とともに “少ない” の度合いが減少し，5 割以上（$0.5 \leq x$）では “少なくはない”（$y = 0$）と定義されている．

「少ない」を修飾する副詞「非常に」に言語ヘッジ:very を用いて，上述の表現は次のように記述される．

```
(:fuzzy-quantifier :very r-few has-student Male)
```

has-student の値となる事例の数を n，そのうち概念Male に属する事例の数を k とし，言語ヘッジ:very によって修飾されたメンバシップ関数r-few を r-few$_{very}$，しきい値を T とすると，**:fuzzy-quantifier** 構文を用いて r-few$_{very}$(k/n)≥T が成立することを述べている．

また，適合度に対する関数に **return-degree** 構文と **defuzzy** 構文がある．(return-degree F a) は，メンバシップ関数 F(a) の値を返す．(defuzzy (F_1 N_{01}^1)...(F_n N_{01}^n)) は，適合度 N_{01}^j を用いてメンバシップ関数 F_j の非ファジィ化[13] を行い，その結果値を返す．

3. KRS-FZ における推論機能と分類器

KRS-FZ 言語の処理系は，概念・関係からなるフレームとプロダクションシステム機能に加えてクエリ機構や知識の制約表現などの多様な知識表現機能を有する推論エンジンである．また，その推論機構は，開世界仮説での単調推論だけでなく，閉世界仮説での非単調推論も行える必要がある．多様な知識表現の下での推論を高性能に行うのに重要な役割を果たすのが分類器である．この章では，KRS-FZ の推論機能について述べた後，分類器の機能とその実現法，分類器を用いた非単調推論の実現法について述べる．

3.1 KRS-FZ における推論機能

KRS-FZ は次のような推論機能を提供する．

(1) 概念間の包含関係の導出
概念 C_1 に属するすべての事例が概念 C_2 に属するとき，概念 C_2 は概念 C_1 を包含するという．概念間の包含関係は分類器によって導出され，知識ベースに保存される．

(2) 事例が属する概念の導出
包含関係によって形成される概念の階層構造を基に，新しい事例が属する概念を導出することを事例認識という．事例認識は分類器によって行われ，その結果は知識ベースに保存される．

(3) クエリ処理
知識ベースに対する質問と検索への解の導出である．

(4) メソッド適用
アクション呼び出し時に，引数と条件部の適合検査，実行メソッド選択，応答部適用の順に行う処理である．

(5) ルールマッチング
tellm による事実の主張後，あるいは，**forgetm** による事実の撤回後に，プロダクションルールの条件部の適合検査，条件部が最も詳細なルールを選択する競合解消，実行ルールの行動部適用の順に行う処理である．

(4) と (5) の条件部の適合検査は，知識ベースへの質問と検索を行うクエリ処理に帰着できる．(3) のクエリ処理と (5) の競合解消は，概念間の包含関係と事例認識の問題に帰着できるため，分類器によって導出される (1) と (2) の結果を参照して実現される．また，分類器は，事実の追加あるいは撤回後に初めて行われる推論の実行時に起動されるだけであるから，分類器の利用により各推論機能が効率良く実現される．

3.2 KRS-FZ における分類器

KRS-FZ 言語の **defconcept** 構文により論理式として定義された概念は，論理式を満す概念オブジェクトを規定する．概念オブジェクトの集合は，概念オブジェクト間の包含関係に基づいた階層構造をなす．KRS-FZ では，あらゆる概念オブジェクトを包含する最上位概念として Thing の存在を仮定し，KRS-FZ のプリミティブとして実装している．したがって，概念オブジェクトの階層構造は，上限要素に Thing を持つ木構造となる．なお，以下の説明では混乱が生じない限り，概念オブジェクトのことも単に概念と呼ぶ．

分類器は，新たに記述された概念と事例に対して，概念の分類と事例の認識を行う推論機構である．

概念の分類： 概念の階層構造の適切な位置に新たに記述された概念を挿入する．“適切な位置” とは，それが包含する全概念が階層の下位に存在し，かつ，それを包含する全概念が階層の上位に存在する位置である．

事例の認識： 概念の階層構造を基に，新たに記述された事例が属する概念を導出する．

図 2 に分類器 **classify** の LISP 流に書かれた再帰的仕様記述のトップレベル部分を与えた．**classify** は新たに記述されたオブジェクト obj と概念の階層構造 H を引数にとる．obj が概念の場合は概念分類器 **c-classify** が起動し，階層構造 H の適切な位置に obj を挿入する．obj が事例の場合は事例認識器 **i-classify** が起動し，H を基に obj が属する概念を導出する．概念の階層構造 H は次のリストで表される．

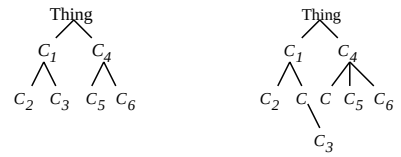
$$H ::= (c\text{-}node\ H_1 \ldots H_n)$$

$c\text{-}node$ は概念の階層構造 H の上限要素であり，

```
classify[obj,H] = if conceptp[obj] then c-classify[obj,H]  else i-classify[obj,H]{ 引数が概念ならば真を返す述語 conceptp により場合分けする }
c-classify[c,H] = if null[H] then nil                                              { 概念 c を階層構造 H に挿入する }
                  else if atom[car[H]]                                             { リスト H の先頭要素が節点の場合 }
                      then if subsume[car[H],c] then list[c,H]      {H の先頭要素が c に包含される場合，c から H の先頭要素への枝を設定する }
                           else if and[subsume[c,car[H]], not-subsume-sh[c,cdr[H]]]    {c が H の先頭要素に包含され，その子節点に }
                              then cons[car[H], cons[list[c],cdr[H]]]  {c と包含関係にある概念がない場合，H の先頭要素から c への枝を設定する }
                           else cons[car[H], c-classify[c,cdr[H]]]                 { それ以外の場合は H の先頭要素の子節点を再帰的に調べる }
                  else cons[c-classify[c,car[H]], c-classify[c,cdr[H]]]          {H が階層構造のリストである場合，各階層構造を再帰的に調べる }
not-subsume-sh[c,shl] = if null[shl] then t                                        {c と包含関係にある概念が shl に存在しない場合に真を返す }
                        else if or[subsume[caar[shl],c], subsume[c,caar[shl]]] then nil
                        else not-subsume-sh[c,cdr[shl]]
subsume[c1,c2] = if eq[c2,Thing] then t                                            {c2 が全概念を包含する概念 Thing ならば真を返す }
                 else if direct-subsume[c1,c2] then t          { c1 と c2 の定義に現れる式を個別に調べ c2 が c1 を包含するならば真を返す }
                 else if derived-subsume[c1,c2] then t           {c1 と c2 の定義に現れる式を比較し c2 が c1 を包含するならば真を返す }
                 else nil                                                          { それ以外の場合は偽を返す }
i-classify[i,H] = search[extend[concept[i],H],role[i],constraint[i],H]             { 上位概念を検出後，各概念の適合度を算出する }
```

[註] KRS-FZ における分類器は，この仕様記述に基づきながら最適化を行って実現されている．

図 2 KRS-FZ における分類器の仕様記述
Fig. 2 Specification of KRS-FZ classifier.



a) 概念の階層構造 H_0　　b) 概念Cを挿入後の階層構造

図 3 概念の階層構造の例
Fig. 3 Examples of concept hierarchy.

$H_1, \ldots, H_n$ の各階層構造の上限要素を包含する．また，階層構造 H は初期値に (Thing) というリストをとる．

3.2.1 概念の分類

概念分類器は，木構造をなす概念構造を上から下にたどりながら，以下の手順により，新たに記述された概念 C を概念の階層構造 H に挿入する．

(1) 概念の階層構造 H の節点 C_T に対し $C_T \subseteq C$ である場合，C から C_T への枝を設定する．

(2) $C \subseteq C_T$ であり，かつ，C_T の子節点のいずれに対しても概念 C との間に包含関係が成立しない場合，C_T から C を葉とする枝を設定する．

(3) (1)，(2) 以外の場合，C_T の子節点に対し再帰的に同様の処理を行う．

図 2 の概念分類器 **c-classify** において，新たに記述された概念 c と概念の階層構造 H の上限要素 $car[H]$ との包含関係は，包含関係判定器 **subsume** を用いて判定する．また，$car[H]$ の子節点と概念 c との包含関係は **not-subsume-sh** を用いて調べる．

例として，**図 3** a) に与えた概念の階層構造 H_0 を考える．このとき，H_0 は次のリストで表される．

(Thing (C_1 (C_2) (C_3)) (C_4 (C_5) (C_6)))

分類器 **classify** が，$C_3 \subseteq C \subseteq C_1$ かつ $C \subseteq C_4$ である概念 C と階層構造 H_0 を引数に実行されると，以下のように処理が行われる．

classify[C,(Thing(C_1(C_2)(C_3))(C_4(C_5)(C_6)))]

⇒**c-classify**[C,(Thing(C_1(C_2)(C_3))(C_4(C_5)(C_6)))]

⇒*cons*[Thing,**c-classify**[C,((C_1(C_2)(C_3))(C_4(C_5)(C_6)))]]

⇒*cons*[Thing,*cons*[**c-classify**[C,(C_1(C_2)(C_3))],
c-classify[C,((C_4(C_5)(C_6)))]]]

⇒*cons*[Thing,*cons*[*cons*[C_1,**c-classify**[C,((C_2)(C_3))]],
c-classify[C,((C_4(C_5)(C_6)))]]]

⇒*cons*[Thing,*cons*[*cons*[C_1,*cons*[**c-classify**[C,(C_2)],
c-classify[C,((C_3))]]],
c-classify[C,((C_4 (C_5) (C_6)))]]]

. . . . . .

⇒(Thing (C_1 (C_2) (C (C_3))) (C_4 (C) (C_5) (C_6)))

概念 C を挿入後の階層構造は図 3 b) のようになる．

3.2.2 包含関係の判定

概念 C の包含関係は，**subsume**[C,C'] によって次のように判定される．

(1) C' が概念 Thing の場合，C は C' に包含される．

(2) 概念 C' が (defconcept C' :is c-expr) あるいは (defconcept C' :implies c-expr) と定義されているとき，

- c-expr が明示的に C を含む場合には，c-expr の式の論理的関係を分析して個々に包含関係を判定する．これを行うのが **direct-subsume** である．
- c-expr が C を明示的に含まない場合には，C の定義に現れる式と C' の定義に現れる式について包含関係を調べる．これを行うのが **derived-subsume** である．

例として，次の 3 つの概念間の包含関係を考える．

```
(defconcept Person)
(defconcept Parent1 :is
    (:and Person (:exactly 1 has-child)))
(defconcept Parent2 :is
    (:and Person (:at-least 1 has-child)))
```

subsume[Parent1,Person] が適用されると，**direct-subsume** が，概念Parent1 の定義に述語:is

により記述された式(:and Person (:exactly 1 has-child))から *Parent1* ⊆ *Person* であるという包含関係を導出する☆．一方，**subsume**[Parent1,Parent2]が適用された場合，**direct-subsume**からは概念オブジェクト *Parent1* と *Parent2* の包含関係が導出されない．しかし，**derived-subsume**が次のように処理を行う．Parent1 の定義から *Parent1* ⊆ *Person* であり，Parent2 の定義から *Parent2* ⊆ *Person* である．また，制約表現(:exactly 1 has-child)と(:at-least 1 has-child)によって定義される概念オブジェクトをそれぞれ $C_{\alpha 1}$，$C_{\alpha 2}$ とすると，$C_{\alpha 1} \subseteq C_{\alpha 2}$ となる．これらを利用して，**derived-subsume** が *Parent1* ⊆ *Parent2* ⊆ *Person* を導出する．

3.2.3 事例の認識とファジィ事象の処理

事例認識器 **i-classify** は，新たに記述された事例 i と概念の階層構造 H を引数にとり，i が属する概念のリストを返す．**i-classify** の主な処理は次の 2 つである．

(1) **extend**[told-c-lst,H]：事例 i が属すると主張された概念のリスト told-c-lst と概念の階層構造 H を引数にとり，told-c-lst を構成する各概念の上位概念を概念の階層構造 H を基に導出し，それらに事例 i が属すると判断する．**extend** の返り値は，導出した概念からなるリストである．

(2) **search**[c-lst,r-lst,constr-lst,H]：**extend** によって導出された概念のリスト c-lst と事例 i に成立すると主張されたロールのリスト r-lst，制約表現のリスト constr-lst を基に，概念の階層構造 H を構成する各概念の適合度を算出する．c-lst を構成する各概念の定義に記述された制約表現は事例 i に対しても成立し，constr-lst を構成する制約表現とともに適合度の算出に用いられる．適合度がしきい値以上の概念は，(明示的な記述はなくても) 事例 i が属する概念であると判断する．**search** の返り値は，**extend** によって導出された概念と適合度がしきい値以上の概念からなるリストである．

3.3 分類器による非単調推論の実現

知識ベースを基に真であると示された命題が，その後の知識ベースの状態変化によって真であると示せない場合には命題の撤回を実現する推論を非単調推論と呼んでいる．KRS-FZ システムには，分類器を用いて次の 3 つの非単調推論が実現されている．

(1) 事実の追加・撤回による知識ベースの修正

知識ベースに事実が追加されるとき，あるいは，知識ベースから事実が撤回されるときに，分類器が事実の導出を行う．次の例を考える．

例.

```
(defconcept Mail-owner :is (:at-least 1 has-mail))
(tellm (has-mail taro m1))
```

このとき，**tellm** 文を用いて追加された事実(has-mail taro m1)により，分類器はtaroが概念Mail-ownerに属すると判断する．その後，

```
(forgetm (has-mail taro m1))
```

によって事実を撤回した場合，taroがMail-ownerに属するか否かは不明となる．

(2) 閉世界仮説による暗黙の限定とその扱い

分類器は，閉世界仮説による解釈と開世界仮説による解釈に応じて処理を行うように実現されている．解釈に閉世界仮説が指定された場合に生じる暗黙の限定には以下の 3 つがある．

(a) 制約表現の適用範囲外における真偽値
制約表現によって規定されていない領域での値は偽と見なす．

(b) 知識ベースにない事実への質問の解
ask によって質問された事実が知識ベースにない場合，偽を返す．

(c) 知識ベースにあるロールの個数
ロールの個数は，知識ベースにあるロールに限定して数えられる．

開世界仮説での解釈が指定された場合，(a) と (b) については「不明」となる．(c) については，明確な記述がないロールの個数を考慮する．たとえば，知識ベースに(has-mail taro m1)というロールが存在する場合，ロールhas-mailの個数は，閉世界仮説では「1 個」と限定して解釈されるが，開世界仮説では他にもロールが存在する可能性があり，「1 つ以上」と解釈される．

(3) デフォルト推論

概念定義時に，**:defaults** を用いてデフォルト規則を設定できる．次の例を考える．

例.

```
(defconcept Mail-box :defaults (:all mail Kept))
    (defconcept Guest-mail-box :implies
        (:and Mail-box (:not (:all mail Kept))))
    (tell (Guest-mail-box mb1))
```

このとき，**tell** 文によってGuest-mail-boxに属すると主張された事例mb1は概念Mail-boxに属すると推論できる．しかし，Mail-boxがデフォルトに有する性質(:all mail Kept)は，Guest-mail-box

☆ *Parent1* ⊆ *Person* は，論理式として定義されたPersonとParent1が規定する概念オブジェクト間の包含関係を意味している．

の有する性質としては明示的に否定されている．したがって，事例mb1には(:all mail Kept)は継承されないことに注意する必要がある．

分類器におけるデフォルト規則の処理は，上述のような知識の継承を考慮した実現となっている．

4. KRS-FZ 処理系の ISLISP による実現

KRS-FZ 処理系は，ISLISP を用いて実現されているが，本章では，その概要を述べた後，KRS-FZ 言語による記述例と KRS-FZ 処理系の実行性能の例を用いた評価について説明する．

4.1 ISLISP による KRS-FZ 処理系の実現

KRS-FZ 処理系は ISLISP 処理系 TISL を用いて実現されており，ISLISP による初めての大規模アプリケーションである．

(1) ISLISP とその処理系 TISL

ISLISP は，日本が提案した核言語を基に設計された Lisp 言語の ISO 標準である[7]．ISLISP は多重名前空間方式を採用し，Common Lisp 系の Lisp 言語であるが，Scheme なみにコンパクトにまとめられている．さらに，ISLISP は多重継承が可能なオブジェクト指向プログラミング機能を備えており，AI アプリケーションの開発に適している．

TISL[8),10)] は ISLISP の高性能処理系であり，ISLISP に含まれていない機能もサポートしている．その 1 つにパッケージ機能があり[9]，大規模アプリケーションの開発にも対応しやすくなっている．

KRS-FZ 処理系は，UNIX，Windows の双方で稼働する TISL で実現された，ポータブルなシステムとなっている．

(2) KRS-FZ 処理系の概要

KRS-FZ 処理系の構成図を**図 4** に与えた．構成図から知られるように，処理系は次の 5 つからなる．

分類器： 概念を包含関係に基づき分類するとともに，概念の階層構造を基に事例が属する概念を導出し，また，各種推論機構を統合する．3.2 節で述べた仕様記述に基づきながらリスト構造の flattening などの最適化を行い実現している．なお，分類器はクエリ処理により自動的に起動されるが，例外が生じた場合の対処は現在のところすべて利用者の負担で解決する必要がある．この点の改善は今後の課題である．

知識ベース： フレームを基にした概念・関係・事例のネットワークとプロダクションルールからなり，各種知識を対応する ISLISP クラスのインスタンスとして格納する．

クエリ処理機構： 知識ベースへの質問と検索を処理する．分類器の導出結果を利用して実現される．

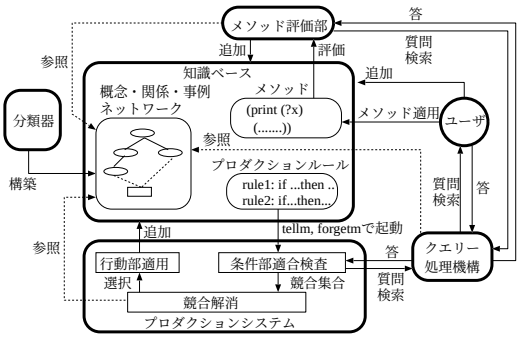


図 4 KRS-FZ 処理系の概要
Fig. 4 Outline of KRS-FZ system.

メソッド評価部： 実行メソッドを導出し，応答部の評価を行う．引数と条件部の適合検査にはクエリ処理機構を用いる．メソッドの応答部に記述された ISLISP 式の評価には，Lisp in Lisp の考えに基づいて TISL 上に作成したインタプリタを用いている．

プロダクションシステム： ルールマッチングを処理する．条件部の適合検査にはクエリ処理機構を用いる．条件部が最も詳細なルールを選択する競合解消は，次のように行う：比較すべき 2 つのルールの条件部に含まれる事例と変数の対応関係を，それらが満足すべき概念，関係，制約表現を基に検出する．そのうえで，概念の階層構造を基に双方の条件部の論理的含意関係を求める．また，行動部はメソッド評価部を呼び出して実行する．

TISL 上に実現された KRS-FZ 処理系は，プログラム行数では 15000 行余，関数とメソッドの数としては 800 個余の規模であるが，ISLISP の Lisp 機能とオブジェクト指向機能，TISL において導入されたパッケージ機能を利用して作成されている．

(3) KRS-FZ 処理系のパッケージ構成

TISL において導入されたパッケージ機能を利用し，KRS-FZ 処理系は各ブロックが 1 つのパッケージにまとめられている．KRS-FZ 処理系のパッケージ構成図を**図 5** に与えた．KRS-FZ 処理系では krs-fz 以下の 23 個のパッケージを新たに定義した．また，他の ISLISP アプリケーションからも頻繁に利用されるソート関数などの汎用関数をまとめた general パッケージがある．各パッケージは ISLISP の Lisp 関数とメソッド定義からなるコンポーネントから構成されている．

(4) 言語の処理とオジェクト指向機能の利用

KRS-FZ 言語の各構文は ISLISP の関数として実現され，他の ISLISP プログラムが呼び出すこともできる．また，クエリ処理などによる推論結果は，リスト

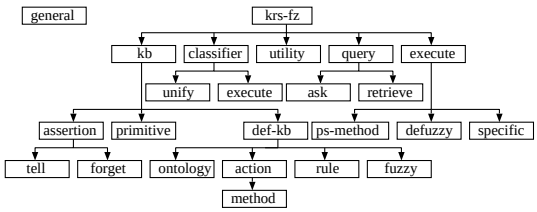


図 5 KRS-FZ のパッケージ構成図
Fig. 5 Structure of KRS-FZ packages.

や記号などの ISLISP の基本データ形で与えられるから，それらを他の ISLISP プログラムも使用できる．

defconcept などの定義形式の構文が評価されると，定義しようとしている知識に相当する ISLISP クラスのインスタンスが生成され，保存される．インスタンスはスロットを複数持ち，定義内容の内部表現を各スロットの値にとる．スロットはメソッドとともに下位クラスへと継承される．

4.2 KRS-FZ 言語による記述例

本節では，KRS-FZ 言語による記述例として実現されたメール送受信シミュレータの一部を示す．これは KL-ONE の例[3] を KRS-FZ の機能，特に，ファジィ制約機能を用いて拡張した例である．この例では次の事項を実現している．

- 仮想システムへのログイン，メールの作成と受信，それらにともなうデーモンの起動
- 「メール」など各種概念の制約表現を用いた記述
- メールの作成，開封，削除などの各種アクションにともなう非単調推論の実現

このシミュレータはオントロジーの構築，事実の主張，デーモンの実現というステップにより作成されている．

4.2.1 オントロジーの構築

「メール」(Mail)，「メールを数多く持つユーザ」(Many-mails-owner) という 2 つの概念の定義を，次のように与える．

```
(defconcept Mail :is
 (:and (:exactly 1 mail-text) (:exactly 1 mail-subject)
       (:exactly 1 send-from) (:at-least 1 send-to)))
(defconcept Many-mails-owner :is
   (:and User (:fuzzy-number many has-mail)))
```

概念 Mail は「本文，題目，送信元を各々ただ 1 つのみ持ち，送信先を 1 つ以上持つ」と定義され，概念 Many-mails-owner は，ファジィ制約を用いて「ユーザであり，メールを数多く持つ」と定義されている．メンバシップ関数 many を次のように定義する．

```
(defmembership many (:r-trpzd 0 10 20 5000))
```

図 1 の:r-trpzd を用いて関数 many を定義しているから，0 通以上 5000 通までを想定し，10 通から徐々に「多数」の度合いが高まり，20 通以上では完全に「多数」であると見なしている．

概念定義において記述された関係「送信先」(send-to) とそれに関連する関係および「メールを持つ」(has-mail) の定義は次のようになる．

```
(defrelation mail-component
   :is :domain Mail :range Thing)
(defrelation address
   :implies mail-component :range Person)
(defrelation send-to :implies address)
(defrelation has-mail :is :domain User :range Mail
   :characteristics :closed-world)
```

この例では，mail-component を定義域が Mail，値域が Thing である関係と定義し，address を mail-component の下位にあり値域が Person である関係,「送信先」(send-to) を address の下位にある関係と定義している．また，:closed-world によって has-mail に閉世界仮説での解釈を指定して所有メール数（すなわち has-mail の個数）を暗黙に限定し，非単調推論を展開する．

4.2.2 事実の主張

メール送受信システムのユーザとして taro が登録され，スーパーユーザとして root が登録されているという主張は次のように書かれる．

```
(tell (Ordinary-User taro) (Super-User root))
```

4.2.3 ルールとアクションによるデーモンの実現

このメール送受信システムでは,「メールを数多く持つユーザへ警告」を行うデーモンを，プロダクションルール warning-rule を用いて実現している．

```
(defproduction warning-rule
   :when (Many-mails-owner ?x) :perform ((warn ?x)))
```

これは,「メールを大量に持つユーザが存在するならば，その者に警告を与えよ」と解釈される．ルールマッチングが開始されると，?x は概念 Many-mails-owner に属する事例に束縛され，それを引数にアクション warn が呼び出される．warn は以下のように定義される．

```
(defaction warn (?x) :filters(:most-specific :last-one)
                     :missing-method :no-op)
(defactmethod warn (?x)
 :situation (Super-User ?x)
 :response ((format (standard-output) "Too many.")))
(defactmethod warn (?x)
 :situation (Ordinary-User ?x)
```

```
:response ((format (standard-output) "TOO MANY!->~A%"
    (* 100 (ask (Many-mails-owner ?x) :degree-p)))))
```

warn は，そのアクションの定義時に，:filters を用いて:most-specific と:last-one が指定する実行メソッドを選択し，適用可能なメソッドが存在しない場合には:missing-method を用いて:no-op を指定している．さらに，warn のメソッドは警告の対象が Super-User か Ordinary-User かによって警告の内容を変えている．一般ユーザの場合，:degree-p オプションをつけた ask 構文により，記述されたクエリの評価値であるメールの個数が「多数」である程度を数値で求め，その値を出力する．ファジィ制約の使用によって，曖昧さが適合度という定量的な尺度で扱われている．

4.3 KRS-FZ 処理系の性能評価

KRS-FZ 処理系の機能と性能の評価のため，巡回セールスマン問題と，4.2 節で述べたメール送受信シミュレータを対象に用いた．巡回セールスマン問題は単純なパターンマッチングで実現できる例であるが，分類器の利用による効率の向上とオーバヘッドの評価のために用いた．また，メール送受信シミュレータはファジィ制約や分類器を用いた非単調推論などを用いて実現される例である．これらを用いて，KRS-FZ 処理系の中核をなす分類器の実験評価を行った．実験環境として CPU が PentiumIII 500 MHz，Memory が 64 MB，OS が Windows2000 である PC 上の TISL を使用した．

4.3.1 巡回セールスマン問題の実行例

巡回セールスマン問題に対する実行時間を**表 3** に示した．表 3 の KRS-FZ は，巡回セールスマン問題を KRS-FZ 処理系によって実現し実行した実行時間を示し，実行時間のうち分類器による処理時間を “[ ]” の中に示している．KRS-FZ-PS は，KRS-FZ から分類器を除いて同じ問題を実現・実行した結果を示している．また，PS-TISL は，単純なパターンマッチングを行うプロダクションシステムを ISLISP 処理系 TISL を用いて実現し，同じ巡回セールスマン問題を実現・実行した結果を示している．以下の考察から知られるように，KRS-FZ 処理系は分類器を活用した高性能で豊富な機能を有する知識処理システムとなっている．

KRS-FZ と KRS-FZ-PS の実行時間の差は分類器の利用による性能の向上を意味する．双方ともに，巡回セールスマン問題を以下のように実現している．

(1) セールスマンが次に進む節点の選択はルールマッチングにより実現している．
(2) セールスマンの移動はルール行動部にメソッドを用いて表現し，ルールの適用時に評価される．
(3) メソッドの応答部では，隣接する節点間のコストを **retrieve** 構文を用いて検索する．

表 3 巡回セールスマン問題を用いた実験結果
Table 3 Results of traveling salesman problem.

節点数	KRS-FZ [分類器の処理]	KRS-FZ-PS	PS-TISL
4 点	0.21 [0.17]	0.52	0.02
7 点	0.66 [0.53]	2.16	0.10
10 点	1.71 [1.46]	6.62	0.29
15 点	6.08 [5.39]	27.0	1.29
20 点	15.9 [14.2]	89.4	4.12

実行時間 [sec]

KRS-FZ-PS：分類器を削除した KRS-FZ
PS-TISL：TISL で作成した単純なプロダクションシステム
実験環境 CPU：PentiumIII 500 MHz，Memory：64 MB，OS：Windows2000

(1)〜(3) のいずれの処理も，概念間の包含関係と事例認識に関する情報を必要とする．KRS-FZ では，包含関係と事例認識を事実の主張・撤回に合わせ分類器が導出し，(1)〜(3) の各処理は，知識ベースに保存された分類器の導出結果を参照して行われるため，効率良く実行される．一方，KRS-FZ-PS では，(1)〜(3) の各処理が行われる前に，毎回，ルール条件部などのクエリに起因する包含関係と事例認識の導出を行う．(1) の処理は，節点の選択がルールマッチングごとに行われるため，ルールマッチングの回数だけ実行される．また，(2) と (3) の処理は，ルールの適用回数だけ実行される．このため，KRS-FZ-PS では，事実の主張・撤回が行われる回数に比べ，(1)〜(3) の各処理が行われる回数が多く，KRS-FZ よりも多くの処理時間を要する．一般に，実行時における事実の主張・撤回はルール行動部に記述するため，他の例でも分類器は効率改善に有効である．

プロダクションシステムを TISL で実現した PS-TISL は，KRS-FZ より良い性能を示している．しかし，PS-TISL は KRS-FZ が提供する知識表現システムとしての様々な推論機能を有していない．このことを考慮すると KRS-FZ 処理系は高性能な汎用知識表現システムであるといえる．

表 3 の KRS-FZ の実行時間から知られるように，巡回セールスマン問題において，分類器による処理時間が実行時間全体の約 8〜9 割を占めている．したがって，KRS-FZ 処理系の実行速度の改善には，分類器による処理を高速化する必要がある．

分類器の処理は概念分類器 **c-classify** と事例認識器 **i-classify** からなる．巡回セールスマン問題では，「節点」などの概念がすべて静的に定義されるから，

表 4 メール送受信シミュレータを用いた実験結果
Table 4 Results of mailing simulator in KRS-FZ.

i-classify	extend	search
0.041	0.01	0.031

[sec]

i-classify：初回の分類器起動時の **i-classify** の処理時間.
extend：初回の分類器起動時の **extend** の処理時間.
search：初回の分類器起動時の **search** の処理時間.
実験環境 CPU：PentiumIII 500 MHz，Memory：64 MB，OS：Windows2000

c-classify は分類器の初回起動時にのみ適用される．一方，**i-classify** は実行時の事実の主張や撤回ごとに適用されるため，分類器の処理効率は **i-classify** に大きく依存する．**i-classify** の主な処理は，3.2.3 項で述べた **extend** と **search** からなるが，巡回セールスマン問題は **search** による処理がほとんど行われない例である．

4.3.2 メール送受信システムの例

4.2 節で述べたメール送受信システムは，ファジィ制約の使用と非単調推論の実現が必要であり，単純なプロダクションシステムでは実現できない例である．たとえば，概念 `User` に属する事例が，ファジィ制約（`:fuzzy-number many has-mail`）を定義に含む概念 `Many-mails-owner` に属するか否かは，事例が属するロール `has-mail` の個数に基づき **search** が導出する．**表 4** に，KRS-FZ を用いて実現したメール送受信シミュレータを実行した実験結果の一部を示した．表 4 において，i-classify は，分類器の初回起動時の **i-classify** の処理時間を示している．extend は同じ条件下での **extend** の処理時間を示し，search は同じ条件下での **search** の処理時間を示している．表 4 からも知られるように，この例の場合，適合度の算出を行う **search** の処理時間が相対的に大きい．

分類器の実行速度を向上するには，**extend** と **search** の処理を改良する必要がある．

5. 関 連 研 究

KL-ONE[3] の枠組みに基づく知識表現システムは，数多く提案されている．KRS-FZ は，特に KL-ONE と LOOM[4),14),15)] の基本概念を参考に設計された．KRS-FZ では確定制約を拡張し，さらに，ファジィ制約を導入し，分類器の再帰的仕様記述を与えて実現している．特に，ファジィ制約の導入によって，KL-ONE や LOOM では不可能であった曖昧な知識表現の扱いが可能となり，また，分類器の再帰的仕様記述が与えられているのも KRS-FZ の特徴である．

CLASSIC[5] は，KL-ONE の提唱者らによって作成された分類器を備えた知識表現システムである．しかし，クエリ言語やプロダクションシステムを持たず，KRS-FZ より機能として劣っている．

KL-ONE ファミリィに属する KRIS[6] は，概念定義が等号関係のみによってしか行えないなどの機能面で劣るシステムであるが，健全かつ完全なアルゴリズムを用いているという特徴がある．

分類器は，Schmolze と Lipkis によって KL-ONE に導入され[16]，KL-ONE や LOOM において実現され，KRS-FZ においても実現された．分類器とその実現法は，KL-ONE や LOOM においても簡単に述べられているが，それを実現するアルゴリズムの明示的な仕様記述を与えたのは，本研究が最初である．

KRS-FZ では，分類器による推論結果を他の推論機構が利用することによって，演繹能力と効率が向上している．分類器と類似した機能を有する動的継承演繹機構を備え，事実の変化に合わせた知識ベースの状態修正が可能なシステムが Objective-C を用いて作成された例[17] が報告されている．しかし，オブジェクト指向データベースシステムとしての側面が強く，プロダクションシステムや非単調推論機構を備えておらず，また，知識の制約表現も使用できない．これらの点が，人工知能的視点からの知識表現システムである KRS-FZ との相違である．

6. お わ り に

本論文では，フレームとプロダクションシステムに基づきファジィ制約を用いて曖昧な知識の扱いを可能とする知識表現言語として設計した KRS-FZ とその処理系について報告した．KRS-FZ は，フレームとプロダクションシステムに基づく KL-ONE ファミリィの基本機能を備えるだけでなく，曖昧な知識の扱いを可能とするファジィ制約機能を備えた知識表現言語となっている．知識の追加や撤回，知識ベースに対する質問・応答の機能などに加えて，知識の開世界仮説と閉世界仮説での解釈を指定する機能も備えられている．また，多様な知識表現の下に多様な推論を行う分類器の形式的仕様記述を与え，これを基に KRS-FZ の効率の良い推論機構を実現している．ここで与えた分類器の形式的仕様記述は，今後，分類器を備えた知識表現システムの作成にあたって参考になるものと考える．

KRS-FZ 処理系は，ISO 標準 Lisp 言語 ISLISP の処理系 TISL を用いて実現されている．これは ISLISP による大規模アプリケーションの初めての試みでもある．KRS-FZ 処理系は TISL のパッケージ機能を利用した複数のパッケージから構成され，各パッケージは

ISLISP の Lisp 関数とメソッド定義によるコンポーネントからなる．KRS-FZ システム開発経験によって TISL パッケージの有用性も確かめられた．

また，作成された KRS-FZ 処理系をいくつかの例に適用し，優れた性能を有することを示した．なお，KRS-FZ 処理系は，文献 18) の KRS-FZ ホームページに公開され，利用可能になっている．

KRS-FZ の言語機能の拡張，分類器の高速化，本格的な AI アプリケーションの開発は今後の課題である．本論文で述べた分類器の形式的仕様記述の正当性は式に関する帰納法を用いて証明できるが，その証明と仕様記述に対する実現の正当性の検証も今後の課題である．

謝辞 KRS-FZ の TISL による実装にあたって協力をいただいた泉信人氏，および，本研究に関して討論をいただいた宮川伸也氏に謝意を表する．また，投稿原稿について詳細なコメントをいただいた査読者に深謝します．

参 考 文 献

1) Tecuci, G., et al.: An Innovative Application from the DARPA Knowledge Bases Program: Rapid Development of a Course of Action Critiquer, *A.I. Magazine*, pp.43–61 (2001).
2) Preece, A., et al.: Better Knowledge Management through Knowledge Engineering, *IEEE Intelligent Systems*, Vol.16, No.1, pp.36–43 (2001).
3) Brachman, R.J. and Schmolze, J.G.: An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science*, Vol.9, No.4, pp.40–62 (1985).
4) MacGregor, R.M. and Brill, D.: Recognition Algorithms for the Loom Classifier, *Proc. 10th National Conference on Artificial Intelligence* (*AAAI 92*), pp.774–779 (1992).
5) Patel-Schneider, P., et al.: The CLASSIC Knowledge Representation System: Guiding Principles and Implementation Rationale, *SIGART Bulletin*, Vol.2, No.3, pp.108–113 (1991).
6) Baader, F. and Hollunder, B.: KRIS: Knowledge Representation and Inference System – System Description, *SIGART Bulletin*, Vol.2, No.3, pp.8–14 (1991).
7) 伊藤貴康：LISP 言語国際標準化と日本の貢献，情報処理，Vol.38, No.10, pp.932–937 (1997).
8) 泉 信人，伊藤貴康：TISL ISO 標準 Lisp 言語 ISLISP 処理系．http://www.ito.ecei.tohoku.ac.jp/TISL/.
9) 泉 信人，伊藤貴康：ISLISP 処理系 TISL のためのパッケージシステム，情報処理学会論文誌：プログラミング，Vol.40, No.SIG 10(PRO 5), pp.17–27 (1999).
10) 泉 信人，伊藤貴康：ISO 標準 Lisp 言語 ISLISP のインタプリタおよびコンパイラ，情報処理学会論文誌，Vol.40, No.9, pp.3510–3523 (1999).
11) 西田豊明：人工知能の基礎，丸善株式会社 (1999).
12) Zadeh, L.A.: A Fuzzy-Set-Theoretic Interpretation of Linguistic Hedges, *Journal of Cybernetics*, Vol.2, No.3, pp.4–34 (1972).
13) 田中英夫：ファジィモデリングとその応用，朝倉書店 (1990).
14) MacGregor, R.M.: Inside the LOOM Description Classifier, *SIGART Bulletin*, Vol.2, No.3, pp.88–92 (1991).
15) Juang, J.Y.H.-L. and MacGregor, R.: CLASP: Integrating Term Subsumption Systems and Production Systems, *IEEE Trans. Knowledge and Data Engineering*, Vol.3, No.1, pp.25–32 (1991).
16) Schmolze, J.G. and Lipkis, T.A.: Classification in the KL-ONE Knowledge Representation System, *IJCAI*, pp.330–332 (1983).
17) 柳沢 豊，塚本昌彦，劉 渤江，西尾章治郎：知識ベース独立のための演繹オブジェクト指向プログラミング，人工知能学会誌，Vol.10, No.5, pp.761–768 (1995).
18) 森谷俊洋，伊藤貴康：KRS-FZ Home Page. http://www.ito.ecei.tohoku.ac.jp/KRSFZ/
19) Bobrow, D.C. and Collins, A.M.: *Representation and Understanding: Studies in Cognitive Science*, Academic Press (1975).
20) Levesque, H. and Lakemeyer, G.: *The Logic of Knowledge Bases*, MIT Press (2000).

(平成 13 年 10 月 29 日受付)
(平成 14 年 9 月 5 日採録)

森谷 俊洋（学生会員）
1978 年生．2001 年東北大学工学部情報工学科卒業．同年東北大学大学院情報科学研究科情報基礎科学専攻博士前期課程進学．

[註]Windows は Microsoft Corp. の登録商標である．UNIX は X/Open Company Limited が認可する登録商標である．

伊藤　貴康（正会員）

1940 年生．1962 年京都大学工学部電気工学科卒業．スタンフォード大学人工知能プロジェクト研究助手，三菱電機中研を経て 1978 年から東北大学．現職，東北大学大学院情報科学研究科教授．工学博士．本会理事，東北支部長等を歴任．現在，Information and Computation の Editor，Higher-Order and Symbolic Computation の Associate Editor，IFIP TC1 member 等．専門分野，ソフトウェア基礎科学および人工知能．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，ACM 各会員．情報処理学会フェロー，電子情報通信学会フェロー．