TM-1041

知識ベース管理システムKappaにおける検索の実現方式と集合を使ったレコード読み

五十嵐　努、杉崎　元（三菱）

May, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

知識ベース管理システムKappaにおける検索の実現方式と集合を使ったレコード読み

基本ソフトウェア開発第一部　・五十嵐　努　・杉崎　元

まえがき

知識ベース管理システムKappa (Knowledge APPlication oriented Advanced database management system) は逐次推論マシンＰＳＩ (Personal Sequential Inference machine) 上に構築されており，高度な知識処理のための環境を提供することを目的としている。そのためにはこれまでのデータベース管理システムでは困難であった複雑なデータ表現を可能にし，実世界の事象を無理なく表わせる必要がある。そこでKappaではデータモデルに非正規形モデル (nested relational model) を採用している。

非正規形モデルは属性（項目）の構造化や繰り返しを許すが，Kappaではさらに項（term）の格納も許すことでこれまでにないデータ表現が可能になっている。

現在では遺伝子配列や，辞書など，従来のデータベースで取り扱うのは困難であった情報をも格納できる知識ベース管理システムとして国内，及び海外の大学，研究機関などで使用されている。

本論文では第Ｉ部で非正規形モデル採用に伴い，高度な機能を持つことになった検索機能の実現方式について述べ，第II部で検索結果である集合を用いたレコードの読みについて，その処理方式を述べる。そして最後に検索機能と集合の今後の課題について考察する。

第Ｉ部　検索の実現方式

1．機能の概要

本章ではKappaで提供している検索機能の概要を示す。

1.1　検索の種類

Kappaが提供する検索機能は，検索対象に応じて以下の３種類がある。

- インデックス検索
  索引であるB+treeを使用して検索する。
- データ検索
  B+treeを使用せず検索対象すべてのレコードを読み検索する。
- 項検索
  インデックス／データ検索を組み合わせて単一化による検索を行う。

これらの３種類の検索はいずれも検索結果として，集合を作成する。ユーザはこの集合を使用して他の集合との演算や，レコード読みを行う（詳しくは第II部を参照）。

以下では，この中のインデックス検索と，データ検索について述べる。

1.2　検索条件

インデックス検索とデータ検索で用いる検索条件は共通で，検索条件内には種々のオプションを指定でき，多様な検索が１回のコマンド実行で行える。

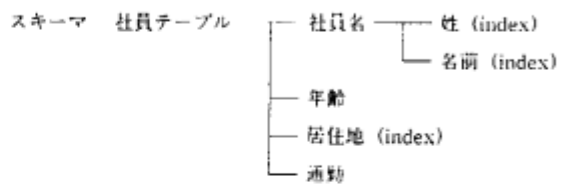
検索条件の基本となるのは検索したい属性と検索値を指定した単一検索条件である。また検索条件内に論理演算子を用い，複数の単一検索条件を指定することもできる（複合検索条件）。

Kappaの検索条件文法の主な項目をBackus Normal Form（ＢＮＦ）記法で表記したものを図Ｉ-1.1に，使用例を図Ｉ-1.2に示す。

```
Condition       ::= "" " <検索条件>
<検索条件>      ::= <条件>
                  | [<検索条件>]<論理演算子><検索条件>
                  | "("<検索条件> ")"
                  % <検索条件>を省略できるのは<論理演算子>が"not"のときのみ
<条件>          ::= <属性名><比較演算子><値1>
                  | <属性名><等号><検索値>
                  | <属性名><等号>"["<検索値>{"," <検索値>}"]"
<論理演算子>    ::= "or" | "and" | "dif" | "not"
<比較演算子>    ::= "<" | ">"
                  | "=<" | ">=" | "="
<等号>          ::= "="
<テーブル名>    ::= アトム
<属性名>        ::= アトム
<検索値>        ::= <部分値>
                  | "{" <部分値>{"," <部分値>}" }"
<部分値>        ::= <値>
                  | <曖昧検索値>
<値>            ::= 整数 | アトム | 文字列
<曖昧検索値>    ::= <値1>
                  | <ワイルドカード>
                  | <曖昧検索値><ワイルドカード>
                  | <曖昧検索値><ワイルドカード><曖昧検索値>
                  | <ワイルドカード><曖昧検索値>
                  | <ワイルドカード><曖昧検索値><ワイルドカード>
<値1>           ::= 文字列
<ワイルドカード> ::= "" * "" | "" ? ""
                  % ワイルドカード"*"を連続して指定することはできない
```

図Ⅰ－1．1　検索条件

社員テーブルに対する検索条件の例



(1) 名前（インデックス属性，属性名：名前）の中に"美"がある人か，"太郎"という名前の人を探す。

Condition＝"（名前 ＝（"＊美＊"，" 太郎"））

：search_index（Kappa，社員テーブル，Condition，Set）

(2) 居住地（属性名：居住地）が鎌倉市で，年齢（属性名：年齢）が25歳以下の人か，通勤手段（属性名：通勤）が自動車の人を探す。

Condition＝"（（（居住地 ＝" 鎌倉市"）and

（年齢＝＜ 25））or（通勤 ＝" 自動車"））

：search_data（Kappa，社員テーブル，Condition，Set）

図Ⅰ-1.2　検索条件の使用例

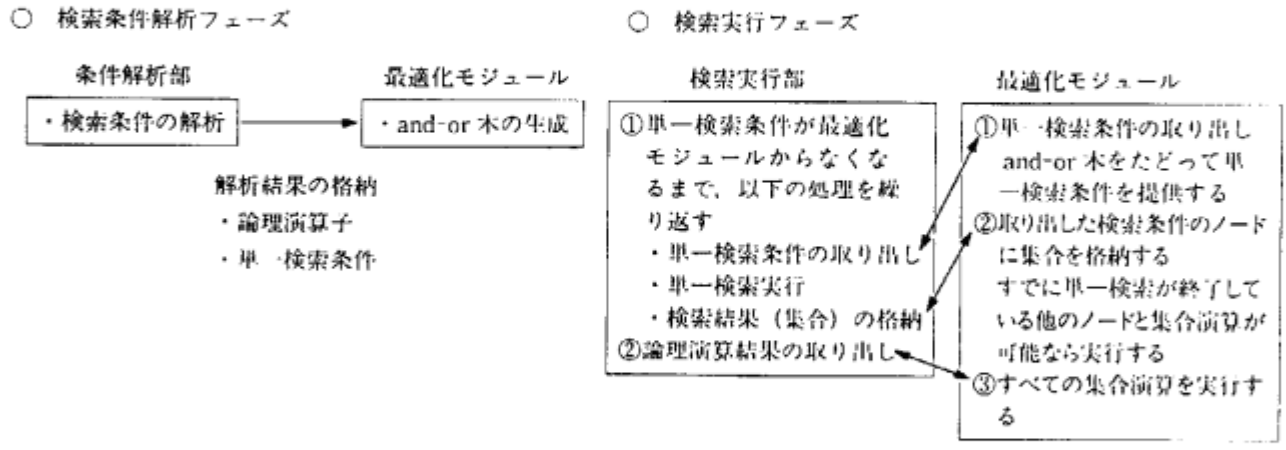
2．実現方式

本章では検索処理の実現方式について述べる。

2.1　実現方式の概要

検索コマンドは，解析部によるコマンド指定の構文解析，検索最適化モジュールによる検索条件の格納，実行部による実際の検索処理という手順で実行される。

各モジュールの関係と検索処理の流れを図Ⅰ-2.1にまとめる。



図Ⅰ-2.1　処理フェーズと各モジュールの処理

2.2　検索条件の解析

解析部は検索条件の解析と最適化モジュールへの格納を行う。その実現方式を以下に示す。

(1)　検索条件の解析

ユーザ指定の検索条件は項として指定される。項はＰＳＩ上ではスタックベクタ（ＰＳＩのデータ型の１つ。以下ＳＶとする）で表現される。入力された検索条件はＳＩＭＰＯＳ（Sequential Inference Machine Programming & Operating System＝ＰＳＩのＯＳ）によって構文解析され，検索条件内の演算子などが誤って指定されていた場合はここでエラーになる。

構文解析された検索条件は図Ｉ-2.2に示した通り，それが「単一検索条件であっても，複合検索条件であっても，2または3要素のＳＶになっている。さらに，そのＳＶの第2，3要素も3または3要素のＳＶに…」と再帰的な構造をとる。

(1) Condition =" (attr 1 = "ABC")　(2) Condition =" (attr 2 =< 10)

→ (=, attr 1, "ABC")　→ (=<, attr 2, 10)

(3) Condition =" (attr 3 =[abc, def, ghi])

→ (=, attr 3, [abc, def, ghi])

(4) Condition =" (attr 1 = jkl) and (attr 2 = mno))

→ (and, (=, attr 1, jkl), (=, attr 2, mno))

(5) Condition =" (not (attr 3 = okm))

→ (not, (=, attr 3, okm))

(6) Condition =" (((attr 1 = qaz) or ((attr 2 = wsx) dif (attr 3 = edc))

A　B　C

and ((attr 4 = rfv) or (attr 5 = tgb)))

D　E

→ ((A or (B dif C)) and (D or E))

→ (and, (or, A, (dif, B, C)), (or, D, E))

第２要素　第３要素

→ (and, (or, (=, attr 1, qaz), (dif, (=, attr 2, WSX), (=, attr 3, edc))), (or, (=, attr 4, rfv), (=, attr 3, tgb)))

図Ｉ-2.2　検索条件の構文解析

一見すると常に要素数の同じＳＶを処理するので，そのＳＶが複合検索条件を示すのか，単一検索条件を示すのか判断が難しそうであるが，検索条件内の演算子を判定の基準とすることでその問題を解決した。

構文解析を行うと，ＳＶの第1要素は必ず演算子になる。この演算子を基準に以下の第2，3要素の内容を判定するようにしたことにより，処理手順が明確にできた。その様子を図Ｉ-2.3に示す。

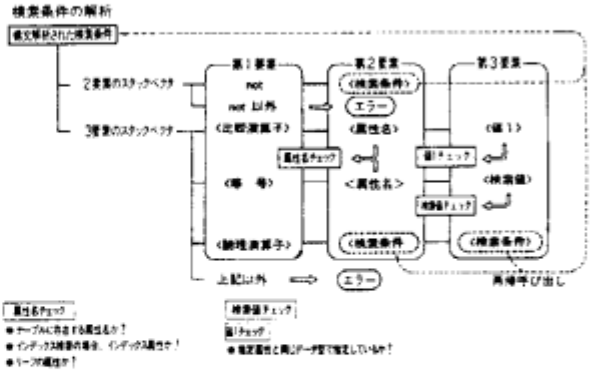


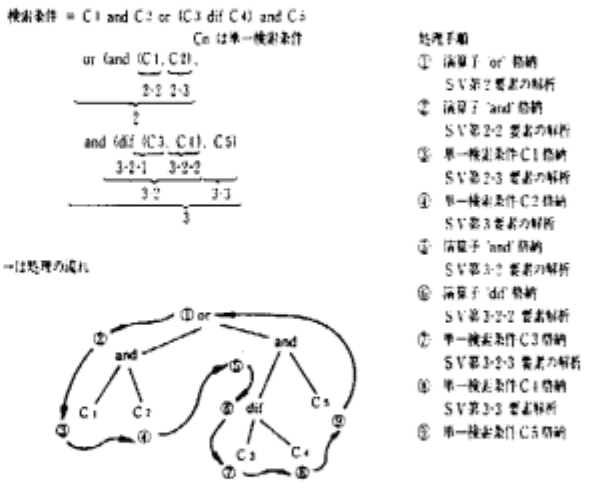
図 1-2.3　検索条件の解析

図Ｉ-2.3中の網掛け部分はＳＶの要素を示す。図中の<検索条件>は図Ｉ-2.2に示した通り複合検索条件か，単一検索条件かの判定がつかないので，再帰呼び出しで処理を行う。

解析部の行う条件文法解析処理は，項構造を解析するループが基本になっている。また<比較演算子>，<等号>を検出すると，それは単一検索条件であるとして，ＳＶ第2，3要素の内容が確定する。矩形枠の処理はその内容の検索条件文法との適合状態，スキーマとの整合性を解析する処理となる。

(2)　検索条件の格納

検索条件は図Ｉ-2.4に示す通り解析部が解析処理で検索条件を分解しながら最適化モジュールに通知し，and-or木に格納していく。and-or木の生成については次節で説明する。



図Ｉ-2.4　and_or 木への格納手順

2.3　and-or 木

and-or 木についてその実現方式を述べる。

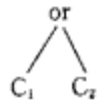
2.3.1　and-or 木を採用した背景

and-or 木は検索条件（項構造）をヒープベクタ（ＰＳＩのデータ型）化しKappa 内部に保持できるようにしたものである。通常，このような場合ＳＩＭＰＯＳの機能を使用し木構造を生成するが，Kappa では以下の要求を満たすため独自に木構造を生成した。

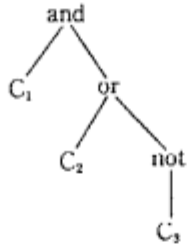
- メモリ性能の向上
- 速度性能の向上
- 検索処理の最適化のための機能

論理的には and-or 木は検索条件内の論理演算子をノードに，単一検索条件をリーフに持つ木構造で，物理的にはヒープベクタである。and-or 木は検索最適化モジュールに生成／管理され，その内部に保持されている。図Ⅰ-2.5 に論理的な構造図を示す。

例1
条件 ＝ C_1or C_2
★ and-or 木



例2
条件 $C_1$and (C_2or (not C_3))
★ and-or 木



図Ⅰ-2,5　and_or 木の論理構造

2.3.2　and-or 木の生成

2.2 節で述べた通り，and-or木の生成は検索条件の解析と並行して行っている。その理由は解析した検索条件をand-or木生成のために再び展開するのでは効率が悪いためである。このとき解析部から最適化モジュールには論理演算子や単一検索条件が1つ1つ通知されてくるので規則性が無いように見えるが，通知される情報には以下の規則があり，それを元に木を構築していく。

(1) not を除く論理演算子（リンク）が通知された場合その下に２つのノードが存在する。
　not の場合，その下のノードは１つである。

(2) 単一検索条件が通知された場合は，そのノードからの下に木の展開はない。

(3) 最初に通知された情報が単一検索条件なら，その検索条件は単一検索条件である。

(4) 論理演算子が通知されたら，次に通知されるのはそのノードの左辺で，左辺が決定した後に，右辺の通知がある。

すなわち解析部は，最適化モジュールに論理演算子と単一検索条件を解析した順番に通知する。and or 木はパージングされた検索条件に木の制御情報（位置情報，最適化情報等）を付加してヒープベクタに格納したものであるから検索条件を解析した順番に木を構成していけばよいことになる。

3．今後の課題

Kappa の検索機能の今後の課題について考察する。

(1) 最適化の実装

現在Kappa では検索処理の最適化をほとんど実装していない。機能検討，設計では最適化の内容まで詰めていたが，開発時間の関係で最適化処理は実装できなかった。

and-or木など最適化機能を実装するための環境は整っているので，今後機会があれば実装していきたい。

特に知識処理システムを構築するとすれば，検索条件も複雑化し最適化が必要になってくることが予想される。

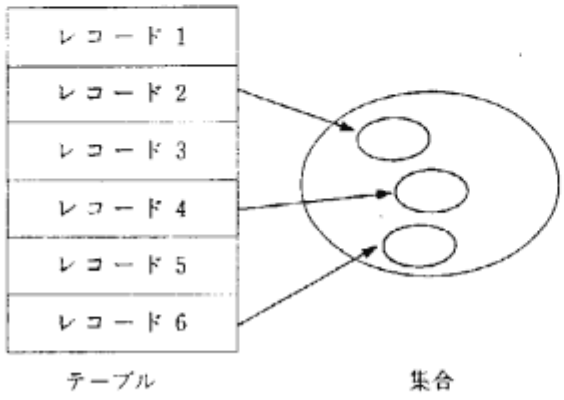
(2) 検索コマンドの統一

1章で説明した通り，Kappa の検索コマンドは現在3種類あるが，これを統一しユーザは検索対象について何も意識せずコマンドを発行できるようにしたい。Kappa 内部で検索対象に応じて処理を切り分けるようにすることにより実現可能である。

第II部　集合を用いたレコード読み

1．集合の概要

Kappa では，データ操作の効率化をおこなうためテーブルの内部表現として集合の概念を採用している。集合とは検索条件を満たすレコードの集まりであり一時テーブルとして使うことができる。



図II-1.1　テーブルと集合の関係

また集合演算（積，和，差，否定）を行うことによって更にレコードの絞り込みを行うことができる。集合では絞り込みを行ったレコードの情報として以下の情報を集合要素としている。

(1)　レコード位置情報（ＲＩＤ：レコード番号）

(2)　属性情報

- 属性名に対応した番号
- オカレンス情報（繰り返しデータの時に選択された繰り返し番号）

集合を用いたレコード読みでは(1)~(3)の情報に従ってレコードを読むことにより第I部で述べた検索条件や集合演算の結果にあったレコードの内容を読むことができる。

第II部ではこれらの情報をどのような構造で実現しているかについて述べた後，その情報を用いてレコードを読む手順について述べる。

2．実現方式

2.1　集合の内部構造

Kappa では検索や集合演算の結果，レコード位置情報（＋属性情報）を集合要素として効率良くストリング（PSI のデータ型の１つ）に格納している。通常は16bit ストリングに格納しているが，処理の高速化を図るためレコードの位置情報のみの場合は32bit ストリングに格納している。

このように集合要素をストリングに保持することによって実際にレコードを読むことなく索引検索や集合演算を行うことができる。

集合の内部構造をＢＮＦ記法で表現し，それぞれの意味を以下に述べる。

(1)　集合の定義

```
<集合>     ::="{" [<ＲＩＤ要素>{,<ＲＩＤ要素>}] "}"
<ＲＩＤ要素> ::="( "<ＲＩＤ>[", ""{"<subRID>
                  {", "<subRID> } " } " ] ")"
<ＲＩＤ>    ::= レコード番号
<subRID>   ::="("<positive>[";"<negative> ] ")"
<positive> ::=" {"<path>  {", "<path> } " } "
<negative> ::=" {"<path>  {", "<path> } " } "
<path>     ::=<attr_id>";"<occ>{":"<occ>}
<attr id>  ::= 属性情報
<occ>      ::= オカレンス情報
```

(注)・各ＲＩＤ要素はＲＩＤの昇順に並んでいる。
・各 path 情報は属性情報の昇順に並んでいる。

(2)　ＲＩＤ要素の意味

```
<ＲＩＤ要素> ::="("<ＲＩＤ>,[", ""{"<subRID>
                  {", "<subRID>"} " } ")"
```

各 subRID 間は or 関係を表している。

<RID 要素>＝(ＲＩＤ, $subRID_1$, $subRID_2$, ・・, $subRID_n$)

<=>$subRID_1$ or $subRID_2$ or ・・・or $subRID_n$

(3)　subRID の意味

```
<subRID>   ::="("<positive>[";"<negative> ] ")"
```

<positive>, <negative> 間は dif 演算を意味している。

subRID=(<positive>;<negative>)

<->positive dif negative

(4)　positive の意味

```
<positive> ::=" {"<path>{", "<path>}" } "
<path>     ::=<attr_id>":"<occ>{ ":"<occ>}
<attr_id>  ::= 属性情報
```

各 path 間は and 条件を意味している。

<positive> =($path_1$, $path_2$, ・・・, $path_n$)

<=>$path_1$ and $path_2$ and・・・, $path_n$

(5)　negative の意味

```
<negative> ::="("<path>{", "<path> } ")"
<path>     ::=<attr_id>":"<occ>{":"<occ>}
<attr_id>  ::= 属性情報
```

各 path 間は or 条件を意味している。

<negative> =($path_1$, $path_2$, ・・・, $path_n$)

$<=>path_1$ or $path_2$ or ・・・, $path_m$

(6) positive と negative の関係

positive dif negative

$<=>(p_1$ and p_2 and ・・・and $p_n)$

dif $(q_1$ or q_2 or・・・or $q_m)$

$<=>(p_1$ and p_2 and ・・・and $p_n)$

and not $(q_1$ or q_2 or・・・or $q_m)$

$<=>(p_1$ and p_2 and ・・・and $p_n)$

and not $(q_1)$and・・・and not (q_m)

(7) その他

subRID = (<positive>;<negative>)とする。

(<>;<negative>) <=> all dif negative

=not(negative) を意味する。

(<positive>;<>) <=> positive dif ϕ

= positive を意味する。

%all ::= レコード全体 (繰り返しを含む)

2.2 レコード読み

次に集合を用いてレコードを読む手順を示す。

(1) 集合から集合要素を取り出す。

(2) レコード位置情報 (ＲＩＤ) に従いレコードを読む。

(3) レコード内容を使い属性情報とオカレンス情報に従って正規形に分解する。

(4) 選択情報を満たすタプルを選択する。

(5) 選択されたタプルを繰り返し構造 (非正規形) にする順番を決める。

(6) 非正規形にまとめる。

(7) 非正規形にまとめた情報に従い内容を返す。

このような処理が必要な理由は，非正規関係の意味を正規関係と同じとしているためである。

以上の処理手順を以下に示すテーブルを例に考え，説明する。

スキーマ情報

テーブル ― 属性1「

　　 ‥ 属性2「

「は繰り返しをあらわす

レコード内容

属性1	属性2
a	10
	20
b	30

検索条件＝(属性1＝a　or　属性2＝30)

とすると検索結果の集合には以下の情報が格納される。

- ●レコードの位置情報　→　レコード1
- ●属性情報　→　属性1，属性2
- ●オカレンス情報　→　1，3

以上の情報を集合の内部構造に合わせて表記すると以下の様になる。

集合の内部構造 ::{(1,({属性1;1 },{属性2;3 }))}

(1) 集合から集合要素(レコード位置情報)を取り出す。

(2) レコード位置情報に従い内部的にレコードを読む。

レコード内容

属性1	属性2
a	10
	20
b	30

オカレンス情報との対応

属性1	属性2
1	1
	2
2	3

(3) レコード内容を元にオカレンス情報を繰り返しのない正規形に分解する。

＊は条件を満たすものを示している。

属性1	属性2
1＊	1
1＊	2
1＊	3＊
2	1
2	2
2	3＊

(4) 選択情報を満たすタプルのみ取り出す。

正規形になっているオカレンス情報の中から選択条件を満たすものだけを取り出す。

属性1	属性2
1＊	1
1＊	2
1＊	3＊
2	3＊

(5) 非正規形にするため，繰り返し構造にまとめる順番を決める。

現在Kappa では繰り返し構造にまとめる順番をスキーマ上の属性順にしている。

この例の場合は属性1，属性2の順で繰り返し構造にする。

(6) (5)で決定した順番に従い非正規形にまとめる。

正規形になっているオカレンス情報を属性1，属性2の順で繰り返し構造にまとめていく。

属性1	属性2
1＊	1
1＊	2
1＊	3＊
2	3＊

テーブルの状態

テーブル ┬ 属性1
　　　　 └ 属性2

属性1	属性2
1＊	1
1＊	2
1＊	3＊
2	3＊

↓

属性1を繰り返しにする。
属性1以外のオカレンス情報で重複しているものをまとめる。
この場合は属性2のオカレンス情報3が重複しているのでこれをまとめる。

テーブルの状態

テーブル ┬ 属性1′
　　　　 └ 属性2

属性1	属性2
1＊	1
1＊	2
1＊	3＊
2	

↓

属性2を繰り返しにする。
属性2以外の属性のオカレンス情報で重複しているものをまとめる。
この場合は属性1のオカレンス情報1が重複しているのでこれをまとめる。

属性1	属性2	
1＊	1	1つ目の繰り返し情報
	2	
1＊	3＊	2つ目の繰り返し情報
2		

これ以上1つにまとめることは不可能なので2つのオカレンス情報に分けている。こうすることで正規形の意味を損なうことなく効率良く情報を返す事が出来る。

(7) 処理の結果テーブルの状態は

テーブル ┬ 属性1′
　　　　 └ 属性2′

となり，属性1，属性2共に繰り返し構造の属性となる。また，ユーザに返すレコードは上図の繰り返し情報に従い下図の通り2つのレコードに分ける。

属性1	属性2	
a＊	i0	1つ目のレコード
	20	
a＊	30＊	2つ目のレコード
b		

3．今後の課題

検索によって集合をつくり，それを使ってレコードを読むという処理は，膨大な情報を持ったテーブルを利用する場合の基本的な手法になる。また，条件によって選択した集合要素を有効に利用し，さらに要素の絞り込みを行うため各種の集合演算が行われることが考えられる。以上のことから次に示す2つの課題をあげることができる。

(1) 検索の高速化

現在の検索ではレコードの選択情報となる集合要素を1要素ずつ集合に格納している。そのため集合要素を格納するために必要となる前後の処理に要する時間が要素数分必要となるため，処理効率は良いとはいえない。そこで集合要素をバッファリングし，一括して格納することにより集合要素を格納するための前後処理の発生回数を減らし，検索全体の処理時間を短くすることが可能である。

(2) 集合要素のチェック機能

集合を用いて処理をしている際に障害が発生した場合，集合に格納されている要素が正当なものかチェックする機能がないため障害の解析に必要以上の時間が取られる。そこで，集合要素のチェック機能を実装し，障害発生時にチェック機能を使い集合要素の確認をおこなうことにより集合要素の正当性を効率良くチェックし，障害解析に要する時間を短くすることが出来る。

むすび

現在 Kappa はＩＣＯＴ(Institute for new generation COmputer Technology)にツールとして登録されており，非正規形モデルのもつ自由なデータ表現という長所を生かしＩＣＯＴ内部の研究室だけでなくＩＣＯＴ参加各社の研究部門や国外の研究施設でも使用されている。

しかしＩ，II部の今後の課題であげたように Kappa にはまだ実装できていない機能や高速化の余地がある。今後はこれらの機能を充実させ，性能向上をおこなうと共にユーザから寄せられる要望を積極的に取り入れ，より良い知識ベース管理システムとなるよう努めていきたい。

〔参考文献〕

[1] 根本，小澤，丹羽
「知識ベース管理システム Kappa 試作システムのデータモデル」
昭和 63 年 3 月ＭＴＣ技術論文集

[2] ＩＣＯＴ
「Kappa 説明書 1.1 版」

[3] ＩＣＯＴ
「Kappa 機能設計書」

[4] ＭＴＣ
「原始コマンドマネジャ詳細設計書」

[5] 椿野
技ノートＮ-90-102
「Kappa での集合演算アルゴリズム」
平成 2 年 4 月

執筆者紹介

五十嵐　努

1986 年 4 月入社。
慶應義塾大学工学部卒
（社）情報処理学会会員
日英機械翻訳システムMELTRANJ/E（操作系）開発を担当。Kappa-II. Tool Kappa の操作系コマンドマンビュレータ，原始コマンドマネジャの開発を担当。現在拡張関係代数の開発を担当している。

杉崎　元

1986 年 4 月入社。
湘北短期大学電子工学科卒
（社）情報処理学会会員
Kappa-I 原始コマンド・マネジャ開発を担当。
Kappa-II. Tool Kappa のバッファ・マネジャ，原始コマンド・マネジャの開発を担当。現在拡張関係代数の開発を担当している。