Ruby を使用した意味処理システムの試作

脇山正博，香月貴光*，日高康展，吉原将太**

A Trial of Semantic Processing System using Ruby Language

Masahiro WAKIYAMA, Takamitsu KATSUKI *, Yasunobu HITAKA and Shota YOSHIHARA**

Abstract

There are many situations that the human beings take communications with the computer by information technology development. There are increasing people who are interesting in the study of artificial intelligence presently. We applied to "Natural Language Processing" in the research of artificial intelligence. However, there are no approaches that have knowledge expression processing by using computer. Therefore we adopt a SD-Form Semantics Model (SD-Form). The SD-Form deals with natural language concept. Our research objective is to make the main (SI, ELAB, NCOA) functions of the semantic processing by Ruby language. The present paper shows the experimental system sdenv4. The system operations are shown by illustrating examples.

Keywords : *Artificial Intelligence, Natural Language Processing, Ruby Language, SD-Form, Semantic Processing*

1．序論

1.1 研究目的

現在情報技術の発達に伴い，人間がコンピュータを使用する機会が増加している．それにより，システムの自動化，情報伝達の効率化などの理由により人工知能への関心が高まっている．

しかし，計算機上で人工知能を実現するには，知識表現，意味解析など問題点が多数存在する[1]．原因の一つとして，自然言語が曖昧であり，概念の意味構造を表すための記述式が存在しないことが挙げられる．

そこで，本研究では自然言語概念の意味構造を表すための記述式としてＳＤ式意味表現モデル[2]を採用する．このＳＤ式意味表現モデルは自然言語概念における個々の概念，陳述表現，感情表現，あるいはシステムに与える知識データを記述するための言語であり，人間とコンピュータとの中間言語としての役割を果たす．このＳＤ式意味表現モデルを用いた意味処理システム sdenv4 (SD-form ENVironment system version 4) をプログラミング言語 Ruby によって作成することを本研究の目的とする．

1.2 本論文の構成

本論文構成として，１章は序論，２章はＳＤ式意味表現モデル，３章は Ruby 言語，４章は実験システム，５章は結論から構成されている．

2． ＳＤ式意味表現モデルの概要

ＳＤ式とは自然言語における個々の概念，陳述表現，感情表現，或はシステムに与える事実データ等を記述するための中間言語であり，その構文は曖昧さのない一つの文脈自由文法ＳＤＧで規定されている．

*　安川情報システム株式会社
**　長崎純心大学

2.1 ＳＤ式生成文法の定義

ＳＤＧは構文的な曖昧さのない記号列を生成する一つの文脈自由文法であり，形式的には

$$SDG = \{\Sigma_N, \Sigma_T, P, \Phi\}$$

と表わせる．　ここに，

Φ　は，開始記号であり Σ_N の一要素である．

Σ_N は，非終端記号の有限集合である．

Σ_T は，終端記号の有限集合であり，概念ラベル，規定子，結合子，機能項目等からなる．

P　は，生成規則の有限集合であり，

S1: X → [s(X),v(X)]　(4.0)
S2: X → [s(X),v(X),o(X)]　(6.0)
S3: X → [s(X),v(X),c(X)]　(6.0)
S4: X → [s(X),v(X),i(X),o(X)]　(8.0)
S5: X → [s(X),v(X),o(X),c(X)]　(8.0)

のような形式で定められている．S1-S5 は何れも陳述ＳＤ式を生成する規則の例である．( ) 内の数値は変数が右辺の形に置き換えられるときに加わる意味の詳述量として定義している値である．その単位を "semit" とする．

2.2 詳述関係

ＳＤ式で表記された二つの概念 D_i , D_j について，構文構造に基づく詳述関係　$ELAB_{synt}(D_i, D_j)$ と知識データに基づく詳述関係 $ELAB_{know}(D_i, D_j)$ を統合したものを，あらためて "詳述関係" と定義し，$ELAB(D_i, D_j)$ と表記する．

すなわち，　$ELAB(D_i, D_j) = \min\{ELAB_{synt}(D_i, D_j), ELAB_{know}(D_i, D_j)\}$

である．　D_i , D_j 間に詳述関係が成立することを，

$D_i \supseteq D_j$ と表わす．

このとき D_i は D_j の先祖 , D_j は D_i の子孫と呼ぶ．

詳述関係は半順序関係であるのでこれを用いて概念を体系化できる．詳述関係は一般には多段階に連鎖する．

D_i, D_j についてそれぞれに最も近い共通の先祖 D_* を D_i , D_j の最近共通先祖と呼ぶ. そして

$$DIFF(D_i, D_j, D_*) = ELAB(D_*, D_i) + ELAB(D_*, D_j)$$

となる値を D_i, D_j の意味差の尺度と定義する.

3. Ruby 言語

第3章ではシステム試作に利用したプログラミング言語である Ruby について述べる.

3.1 Ruby[3]

Ruby (ルビー) は, まつもとゆきひろにより開発されたオブジェクト指向スクリプト言語である.

オブジェクト指向とは小さなプログラム (サブルーチン) を組み合わせることで大きなプログラムを作るという考え方のことである. スクリプト言語とは, プログラムの動作内容を台本 (script) のように記述できる簡易的なプログラミング言語の総称である. 機能として, 強力な正規表現, クラス定義, 例外処理, イテレータなどがある.

3.2 正規表現

正規表現とは文字列とパターンをマッチングする際のパターン記述法のことである. 正規表現には "=~" という正規表現と文字列がマッチするかどうかを調べるメソッドが用意されており [正規表現 =~ 文字列] という形で使用する. マッチしない場合には "nil" を, マッチする場合には, マッチした文字列が始まる文字の位置を返す.

```
if(正規表現 =~ 文字列)
  マッチした場合の処理
else
  マッチしなかった場合の処理
end
```

この正規表現を利用することでマッチング処理を行う.

3.3 クラス定義

クラスはオブジェクトの種類を表すものである. Ruby のオブジェクトは例外なくなんらかのクラスに属している. オブジェクトがどのように振舞うのか, つまり, オブジェクトにメソッドを適用させたときに何が起きるのかは, オブジェクトが属するクラスによって決められている. たとえば, 数値クラスのオブジェクトであれば, 足し算や引き算などが行える. また, 文字列クラスのオブジェクトであれば, 別の文字列オブジェクトとつなぎ合わせたり, また, 個々の文字に分割したりできる. このようなメソッドが各クラスに定義されている.

3.4 例外処理

プログラムの実行中にエラーが発生すると例外が発生する. 例外が発生すると実行は一時中断し, 例外処理を探す. 例外処理とはこのような例外が発生したときに次に何を実行すればよいかを記述できることである.

例外処理の仕組みのない言語では, 処理が完全に終わったかどうかを確認しながらプログラムを書く必要がある. そのため, プログラムの多くの部分をエラー処理に費やすことになり, 繁雑になりがちである. しかし, 例外処理を用いることによって, エラーの発生場所も同時に報告されるためデバッグしやすく, プログラムの見通しもよくなるといったメリットがある.

3.5 イテレータ

イテレータとは「繰り返し」を抽象化したもの. Ruby には様々な繰り返し処理が用意されており, 使用場面に応じて書くことができる. 基本的なイテレータの一つに each メソッドがある. このメソッドは「順番に要素を取り出し, その要素を使って何かの処理をする」ためのイテレータで, Ruby の様々なクラスで定義されている.

4. 実験システム

第4章では本研究で作成した実験システムについての説明を動作例とともに述べる.

4.1 システム構成

本研究においての開発環境は以下の通りである.

- Microsoft Windows 2000 Server
- Intel Celeron CPU 2.40GHz
- Memory 448MB
- Ruby 1.8.6

4.2 sdenv4 について

本研究で作成したSD式意味表現モデルを使用した意味処理システム sdenv4 は Ruby version 1.8.6 でプログラミングしている. sdenv4 はSD式意味表現モデルを利用した様々なシステムで利用できるように, それぞれの処理をサブルーチンとして用意している. 実際に他のシステムで利用するには, Ruby の require 関数を用いてサブルーチン群を記述したファイル "sdenv4.rb" を取り込む.

sdenv4 を動作させるために必要なファイルは次の4つで, "MAIN.rb" から "sdenv4.rb", "opcont.rb" と "fact.dat" を取り込み, 処理を制御している.

MAIN.rb	: 操作画面を作成する
sdenv4.rb	: sdenv4 のサブルーチン群
opcont.rb	: 入出力のみを扱う
fact.dat	: 知識データ

sdenv4 の主な機能を以下に示す.

(1) 入力概念のSD式判別
(2) 意味量の計算
(3) 詳述関係
(4) 最近共通先祖の導出
(5) 意味差の計算

実験システムを Fig1 に示す.

各種動作確認する際は, Fig.1 において, まず入力欄に何らかの概念を入力する. このとき, 入力の途中で改行しない.

Fig. 1 Execution screen of "MAIN.rb"

また，elab，ncoa と diff の動作確認の際は，二つの概念を入力するが，その際二つの概念の区切りには半角スペースを用いる．入力後に "sd"，"si"，"elab"，"ncoa"，"diff" の中から確認したい処理のボタンを押すと，処理結果がボタンの上にあるログ画面に出力される．

4.3 入力概念のＳＤ式判別：sd

入力された概念がＳＤ式であるかどうかを調べ，ＳＤ式でなければ "no"，ＳＤ式であればその形式（Table 1 参照）を出力する．

Table 1 Return value of SD-form

ＳＤ式の形式	値
文型１	type1
文型２	type2
文型３	type3
文型４	type4
文型５	type5
文型６	type6
文型７	type7
文型８	type8
文型９	type9
呼びかけ	type_a
応答	type_r
感嘆	type_e
結合子形式	connector
規定子形式	prescriptor
修飾形式	modify
単純ラベル	label
ＳＤ式以外	no

sdenv4 実行画面（Fig. 2 参照）において，ボタンは "sd" を押す．以下にＳＤ式の形式を調べる処理の例を示す．

[動作チェック例]

入力：[s(dog/big),v(eat/past),o(meat)]

出力：type 3 （大きな犬は肉を食べた．）

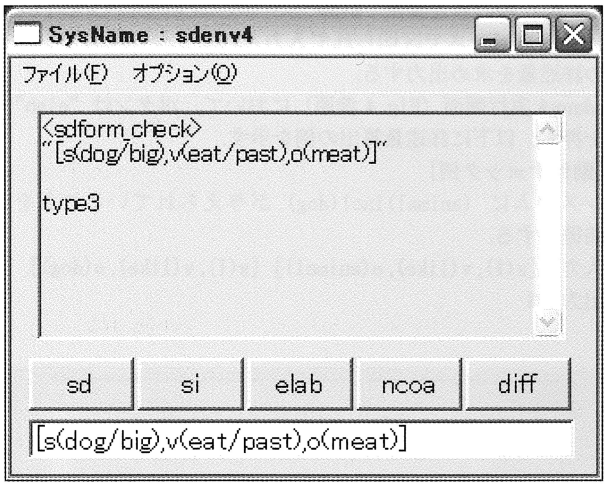

Fig. 2 Check of SD-form

4.4 意味量の計算：si

入力されたＳＤの意味量を計算し，その結果を出力する．sdenv4 実行画面（Fig. 3 参照）において，ボタンは "si" を押す．

また，本システムの各意味量は以下のように定義した．

(1) 変数ラベル "X, Y, Z, …" : 1[semit]
(2) 単純ラベル "dog, cat, …" :10[semit]
(3) 修飾子 "/" : 1[semit]
(4) 規定子 "nega, assu, …" : 2[semit]
(5) 結合子 "incl, para, …" : 1[semit]
(6) 機能項目記号 "s, v, c, …" : 1[semit]
(7) 区切り記号 "[ ]" : 1[semit]
(8) 区切り記号 "( )", "," : 0[semit]

以下にＳＤ式の意味量計算の例を示す．

[動作チェック例]

入力：[s(Taro),v(buy/past),o(book/costly)]

出力：56 （太郎は高価な本を買った）

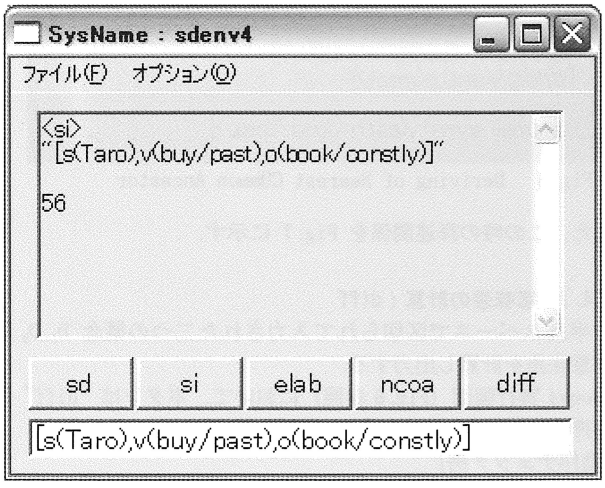

Fig. 3 Calculation of SI

4.5 一般の詳述関係：elab

半角スペースで区切られて入力された二つの概念 D_1, D_2 の詳述量を求め出力する.
sdenv4 実行画面（Fig.4 参照）において，ボタンは“elab”を押す．以下に詳述量算出の例を示す.
[動作チェック例]
システムに (animal)incl(dog) が与えられていることを前提とする.
入力：[s(I),v(like),o(animal)] [s(I),v(like),o(dog)]
出力：3

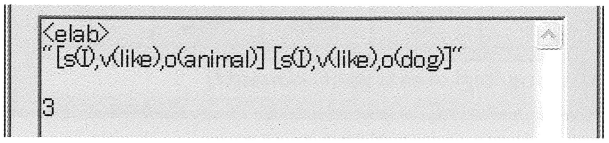

Fig.4　Calculation of ELAB

4.6 最近共通先祖の導出：ncoa

半角スペースで区切られて入力された二つの概念 D_1, D_2 の最近共通先祖を求め出力する.
sdenv4 実行画面（Fig.5 参照）において，ボタンは“ncoa”を押す．以下に最近共通先祖導出の例を示す.
[動作チェック例]
システムに次の知識データが与えられていることを前提とする.
(animal)incl(human)
(animal)incl(dog)
(food)incl(meat)
(food)incl(vegetable)
入力：[s(human),v(eat),o(vegetable)]
　　[s(dog),v(eat),o(meat)]
出力：[s(animal/some),v(eat),o(food/some)]
　　（ある生き物がある食べ物を食べる．）

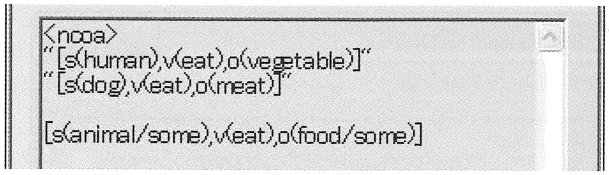

Fig.5　Deriving of Nearest COmmon Ancestor

また，この時の詳述関係を Fig.7 に示す.

4.7 意味差の計算：diff

半角スペースで区切られて入力された二つの概念 D_1,D_2 の意味差を計算し出力する.
sdenv4 実行画面（Fig.6 参照）において，ボタンは“diff”を押す.
[動作チェック例]
4.6 節と同様，システムに次の知識データが与えられていることとする.
(animal)incl(human)
(animal)incl(dog)
(food)incl(meat)
(food)incl(vegetable)
入力：[s(human),v(eat),o(vegetable)]
　　[s(dog),v(eat),o(meat)]
出力：4

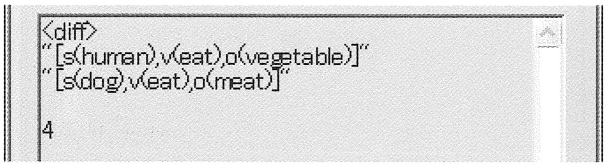

Fig.6　Calculation of semantic DIFFerence measure

例として与えられた D_1, D_2 についての最近共通先祖 D_0 に関するすべての詳述関係を Fig.7 に示す.
この Fig.7 において数字は詳述量を表している.

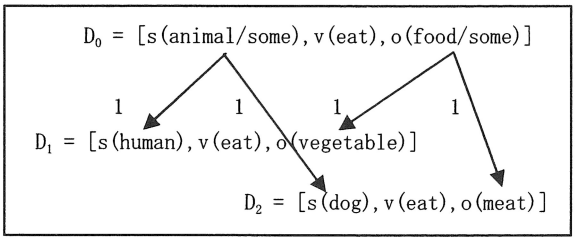

Fig.7　NCOA D_0 of D_1 and D_2

5．結論

本研究ではＳＤ式意味モデルの有効性を確認するための実験システム sdenv4 をプログラミング言語 Ruby を用いて作成した．Ruby は HTML 等の言語と組み合わせることで様々な場面での応用が可能である．また，各種処理を“sdenv4.rb”にサブルーチンとして記述したため，他のシステムに取り込む際に使用したい部分のみを使えるようになっている.
試作した sdenv4 では，「入力概念のＳＤ式判別」，「意味量の計算」，「詳述関係」，「最近共通先祖の導出」，「意味差の計算」を求める機能がある．これらの機能を用いることでＳＤ式意味表現モデルとして，他の意味処理システムに応用することができる.
今後の課題として，知識データの自動登録（自己学習機能の追加）があげられる.

参考文献

[1] 田中穂積監，辻井潤一共著：自然言語理解，
オーム社　(1989)
[2] Wakiyama,M.,Noda,H.,Hozaki,K. and Kawaguchi,E.
: Computation Algorithm of Semantics Model,
Trans. IPS Japan, Vol.40, No.3, pp.1065-1079(1999)
[3] まつもとゆきひろ監，高橋征義・後藤裕蔵著：
たのしいRuby 第２版ソフトバンク クリエイティブ
(2007)

(2009年10月９日　受理)