自由文思考プログラミング環境における手続的機能の実装

中村 圭介[1,a)]　安藤 建剛[1,b)]

概要：日本語等で人間の本格的な思考をプログラミングできることを目指して述語論理等宣言的な知識の処理に適した独自処理系を開発している. 一方, 人間の本格的な思考に該当する手続的機能（本格的な数値計算や文字列計算）は, 順次・分岐・反復制御等を用いた手続的なプログラミングのほうが, 人間の動的な思考様式に対応して見通しもよく, アルゴリズムの工夫に応じて効率的となる場合も多い. そこで, 標準的な論理プログラミング言語の手続的解釈における演算機能を手本にしながら, よりバックトラックの少ない独自処理系の性質を生かした比較文・制御構文・代入文・四則演算・文字列処理等を実装したので, これらについて紹介する.

キーワード：自然言語, 論理プログラミング, 比較文, 制御文, 代入文, 演算

1. はじめに

日本語等で人間の本格的な思考をプログラミングできることを目指して述語論理等宣言的な知識の処理に適した独自処理系を開発している [2], [3].

これは, 次のような機能を有し, 誰もが自由文 (ふだんの自然言語等) のまま本格的な論理プログラミングに親しめることを目指している.

- 演繹 (簡単な証明, 解の探索や合成) [2], [3], [13]
- 動的引用 (ネットの専門知識や最新情報）[4]
- 否定（証明の失敗, 科学的／道徳的タブー)[12]
- 独自の価値観や程度の定義と利用 [5]
- その他 [6], [7], [8], [9], [10], [11]

しかし, 特定のオントロジーに元づく記号論理 (atom がベースとなる Prolog 等) を採用せず, 文字がベースとなる自由文を採用したことにより「多長一致」の処理, すなわち, 変数を含む文字列パターンと定数文字列等との間にマッチの仕方が複数ありうることに対応するため, ありうる一致の仕方の数 (下図では 2 回) だけループして解候補を探索すること [3] 等が必要になる.

```
$X と $Y は兄弟
太郎と次郎と三郎は兄弟
↓
($X,$Y) = {
        (太郎,次郎と三郎),
        (太郎と次郎,三郎)
}
↓
2回のループ
```

図 1

このパターンマッチ等の負荷をなるべく減らす, すなわち不規則なパターン同士のマッチングの再計算とそこから生じる多長一致集合の管理を減らすために, 次のリストのような推論規則の中では

[1] ナレルシステム株式会社
[a)] keisukebecome@icloud.com
[b)] t.ando@marusei-sk.com

なるべくバックトラック (もどって別の解を探索すること) をせず, 同じパターン対のマッチングの回数を減らせる幅優先アルゴリズムを採用している [2], [3].

```
兄弟($X,$Y) :- $Xと$Yは兄弟 ; $Xは人間 ;
太郎と次郎と三郎は兄弟
太郎は人間
次郎は人間
三郎は人間

質問： ?兄弟$A
↓
↓パターンマッチ(1行目第1条件⇔2行目)
↓
($X,$Y) = {
        (太郎,次郎と三郎),
        (太郎と次郎,三郎)
}
↓
↓絞り込み（1行目第2条件⇔3行目)
↓
($X,$Y) = {
        (太郎,次郎と三郎)
}
↓
答え： $A=(太郎,次郎と三郎)
```

図 2

これは, 上図のように, 推論規則の成立条件であるボディ部のリテラル列 (”$X と$Y は兄弟 ;” と”$X は人間 ;”) を左から第 1 条件→第 2 条件→... と走査する (コンピュータがなぞる) 際, 左のリテラルから右のリテラルへと解候補の全体 (($X,$Y)={...}) を幅優先で管理しながら探索する手法である. なお, 標準的な Prolog の深さ優先探索とは異なるこのような方針も有利でありうることは既に指摘されている [1].

この幅優先手法では, 新しいリテラルにおいて, それまでの (=より左の) リテラルにおいて出現しなかった新しい変数が出現した場合, 管理すべき解候補の数が, その新しい変数の解のバリエーションとの組合せにより単調増加し (図 3), そうでない場合 (新しい変数が出現しない場合), 積集合を求めることにより単調減少することになる (図 2).

また, 証明の失敗による否定条件のリテラルの場合 (図 4 の 1 行目) 間引きにより単調減少することになる. 本稿の制御文の機能を実現するにあたり, この間引きは, 既に列挙されている幅優先解候補についてのみ適用することとし, 否定条件 (図 4 の 2 行目の第 1 条件) の後 (繰り返しを許容するため, 成立条件中のより右のリテラルであるとは限らない) に出現する幅優先解候補 (図 4 の 2 行目の第 2 条件による候補) には適用しないこととした.

```
兄弟($X,$Y) :- $Xと$Yは兄弟 ; $Zは人間 ;
太郎と次郎と三郎は兄弟
太郎は人間
次郎は人間
三郎は人間

質問： ?兄弟$A
↓
↓パターンマッチ(1行目第1条件⇔2行目)
↓
($X,$Y) = {
        (太郎, 次郎と三郎),
        (太郎と次郎, 三郎)
}
↓
↓単調増加（1行目第2条件⇔3,4,5行目)
↓
($X,$Y,$Z) = {
        (太郎, 次郎と三郎, 太郎)
        (太郎, 次郎と三郎, 次郎)
        (太郎, 次郎と三郎, 三郎)
        (太郎と次郎, 三郎, 太郎)
        (太郎と次郎, 三郎, 次郎)
        (太郎と次郎, 三郎, 三郎)
}
↓
↓($X,$Y)についてのみ集約
↓
答え： $A={
        (太郎, 次郎と三郎),
        (太郎と次郎, 三郎)
}
```

図 3

```
1行目: OK :- $Xと$Yは兄弟 ; ^ $Xはロボ ;
2行目: NG :- ^ $Xはロボ ; $Xと$Yは兄弟 ;
```

図 4

このような特性をもつ推論規則の中に, より本格的な手続的プログラミング機能, すなわち制御文を部分的に導入して人間の思考をより本格的に実現できるようにすることが本稿の目的である.

以降では, 導入したいプログラミング機能 (2), 幅優先の探索方針をもつ推論規則中での実現上の各問題点と解決策 (3,4,5,6), 実装のために採用した方針と考察 (7), まとめと今後の課題 (8) の順で述べる.

2. 導入したいプログラミング機能

新機能を導入する前に実装していた機能は以下のとおりである.

2.1 導入前の既存機能の確認 (2011〜2018)

- 文字列を命題とする論理の真偽の決定
- 自由な述語による成立条件を充たす変数（又は変数組）の列挙と絞り込み
- 証明（説明）の失敗としての否定を用いた成立条件による絞り込み
- 科学的／道徳的なタブーとしての否定を用いた行きすぎた自動解合成の抑制
- 解として許容する変数の値の比較（整数, 文字列, パターン）による解の絞り込み (下図)

```
チョコレートAは150円
チョコライスは120円
チョコビーンは110円
安価チョコは$Xで$Y円 :- \\
  $Xは$Y円 ; \\
  {$X===チョコ$C} ; \\
  {$Y<=120} ;
↓
質問： ?安価チョコは$A
↓
答え：$A={
  (チョコライスで120円),
  (チョコビーンで110円)
}
```

図 5

この機能の比較は次のように, 数値と文字列の一致, 相違, 大小により行う.

```
{$X==1+1} ; {$X<>6-2} ;
{$X>=11*5} ; {$X>24/4} ;
{$X<=11*(5+1)} ; {$X<(20+4)/4} ;

{$X==abc} ; {$X<>abc} ;
{$X>=abc} ; {$X>abc} ;
{$X<=abc} ; {$X<abc} ;

{$X==$Y+1} ; {$X<>($Y+1)*$Z} ;
{$X>=$Y} ; {$X>$Y} ;
{$X<=$Y} ; {$X<$Y} ;
```

- 基本処理 (代入, 四則演算, 文字列演算, 表示) の順次制御 (手続的解釈)
- 手続（推論規則）の呼出
  これについて特筆すべきは, Prolog やC 言語等の呼出と異なり, 推論規則の結論部分 (Prolog では頭部) における「文字列パターンのなんらかの共通性」を用いて, 関連する複数の推論規則を簡潔に呼び出すことが可能であることである (図 6).

```
abc :- #print " ABC!"
abd :- #print " ABD!"

質問(呼出)： ?ab$A
↓
表示(副作用)：  ABC!ABD!
↓
答え：  $A={(c),(d)}
```

図 6

- ネット上等の他プログラム (自然言語可) の動的引用
- Web のようにみんなで世界観をつないで広げるアドベンチャー型ＲＰＧ [14], [15]
- プログラムや選択肢を選択するための音声認識と音声合成
- 選言 (OR 結合) で構成された推論規則 (OR, 個数/重みによる閾値, 同義語一括カット, 等)

2.2 導入したい制御文機能

導入したい制御文は以下の３つである. なお, 既

存処理系の推論規則のうち, 順次処理とみなすことが不適当な OR 結合に類するものへの導入はエラーメッセージで禁止することとした.

- 分岐 (IF～EL～FI, 又は,IF～FI)
  ELIF や複雑な比較文は課題とする.
  ※「\\」は実際の処理系では使用不可

```
a(1,2)
a(3,6)
a(6,9)
test($A,$B) :-
  a($A,$B) ; \\
  {#IF $A>5} ; \\
    {$A=$A-1} ; \\
  {#EL} ; \\
    {$B=$B+10} ; \\
  {#FI} ;

質問： ?test($X,$Y)
↓
 ($X, $Y) ={ (1,12), (3,16), (5,9) }
```

図 7

- 繰り返し (WH～HW)
  BREAK 文等は今後の課題とする.

```
a(1,2)
a(3,6)
a(6,9)
test($A,$B) :-
  a($A,$B) ; \\
  {#WH $A>5} ; \\
    {$A=$A-1} ; \\
    {$B=$B+10} ; \\
  {#HW} ;

質問： ?test($X,$Y)
↓
 ($X, $Y) ={ (1,2), (3,6), (5,19) }
```

図 8

- 入れ子 (ネスト) 構造 (IF と WH による)
  ※「\\」は略

```
a(1,2)
a(3,6)
a(6,9)
test($A,$B) :-
  a($A,$B) ;
  {#WH $A>3} ;
    {$A=$A-1} ;
    {#IF $B>2} ;
      {$B=$B-2} ;
    {#EL} ;
      {$B=$B+5} ;
    {#FI} ;
    {$B=$B+10} ;
  {#HW} ;

質問： ?test($X,$Y)
↓
 ($X, $Y) ={ (1,2), (3,6), (3,33) }
```

図 9

3. 分岐により関連する値が増える解候補と増えない解候補の扱い

値のない変数をもつ解候補のその変数の値は「__DC__」(DONTCARE:なんでも可) として管理する.

- __DC__に四則演算や文字列追加はできない (0 や空文字 (””) とはしない)
- __DC__のまま最終結果を表示する

```
a(1,2)
a(3,6)
a(6,9)
b(8)
test($A,$B,$C) :-
  a($A,$B) ;
  {#IF $A>5} ;
    {$A=$A-1} ;      //(6,9)⇒(5,9)
    b($C) ;          //(5,9)⇒(5,9,8)
  {#EL} ;
    {$B=$B+10} ; //⇒(1,12,?) (3,16,?)
  {#FI} ;
  {$C=$B+$C} ;   //(5,9,8)⇒(5,9,17)

質問： ?test($A,$B,$C)
↓
 ($A, $B, $C) ={
        (1,12,__DC__),
        (3,16,__DC__),
        (5,9,17)
}
```

図 10

4. IF～EL～FI 分岐した両コースで解候補が消滅する場合

その分岐を含むブロックを false とする（たとえその後解候補が復活しても). 一番外側のブロックのとき. その推論規則の呼出を false として中断する.

```
a(1,2)
a(3,6)
a(6,9)
b(8)
今日はくもり
test($A,$B) :-
  a($A,$B) ;
  {#IF $A>5} ;
    {$A=$A-1} ;    //(6,9)⇒(5,9)
    今日は晴れ ;   //IFの全解消滅
    b($B) ;         //真だが通らず
  {#EL} ;
    {$B=$B+10} ;  //⇒(1,12) (3,16)
    a(5) ;        //ELの全解消滅
  {#FI} ;
  今日はくもり ;  //真だが通らず

質問： ?test($A,$B)
↓
NO
```

図 11

5. WH～HW(又は IF～FI) に入る解と入らない解がある場合

入らない解については HW や FI まで無条件で到達させる. 入った解の種類が単調増加した場合, 入らない解候補の該当変数部分は__DC__とする.

```
a(1,2)
a(3,6)
a(6,9)
b(8)
今日はくもり
test($A,$B) :-
  a($A,$B) ;
  {#WH $A>5} ;
    {$A=$A-1} ;    //(6,9)⇒(5,9)
    b($C) ;        //(5,9)⇒(5,9,8)
  {#HW} ;

質問： ?test($A,$B)
↓
 ($A, $B, $C) ={
        (1,2,__DC__),
        (3,6,__DC__),
        (5,9,8)
}
```

図 12

6. WH～HW(又は IF～FI) の「途中」で, 入った解候補がすべて消滅した場合

HW の次の処理に抜け, その時点で解候補が存在していなければ, その抜けた直後の時点でのブロック全体を false として終了する (一番外側のブロックのとき. その推論規則の呼出を false として中断する).

```
a(1,2)
a(3,6)
a(6,9)
b(8)
今日はくもり
test($A,$B) :-
  a($A,$B) ;
  {#WH $A>3} ;     //(6,9)だけが入れる
    {$A=$A-1} ;    //(6,9)⇒(5,9)⇒(4,9)
    {#IF $A<5} ;
      今日は晴れ ; //偽で(4,9)消滅
    {#FI}
  {#HW} ;          //(1,2) (3,6)は維持
  今日はくもり ;   //真故全解維持

質問： ?test($A,$B)
↓
 ($A, $B) ={
        (1,2),
        (3,6)
}
```

図 13

7. 実装のために採用した方針と考察

推論規則が再帰的に呼ばれることを前提とした各呼出文脈において, 複数ある解候補ごとに, 以下を管理するフラグのスタック (制御文のネストに対応するためのローカル変数) を設けることにより, 上記の各問題に見通し良く対応する方針とした.

- その解候補が制御構文のネスト中のどのブロック（又はサブブロック）を経由べきか
- その解候補がそのネストにおいて絞り込み等により既に無効化されていないか

なお, 標準的な Prolog 等の深さ優先探索では, このようなフラグ管理をしなくても一つ一つの解候補ごとに制御文に係る各ブロックを通るか通らないかを自然と判断することになるが, 前述した不規則なパターンマッチングの再実行が, 特に動的引用機能等によりコンパイルが難しい処理系では必要になってしまうため本処理系の幅優先探索が有利な場合も多いと考える.

8. まとめと今後の課題

自然言語等を含むパターン対の不規則なパターンマッチングの繰り返しを防ぐために解候補の幅優先探索を採用している論理プログラミング処理系において, IF 文,WHILE 文 (表記は WH) 及びその入れ子の制御文を導入して手続的な処理を容易化するために採用した実装方針について示した. 実動作環境はシンポジウム期間中のパソコンデモ及び当社 Web 環境において紹介したい.

以下を今後の課題と考えている.

- 深さ優先探索を行う処理系との演算速度比較実験
- 条件分岐における変数や定数を複雑に組み合わせた項や条件式の取扱い
- elseif 文
- for 文
- 四則演算の operator の優先順位の定義（括弧を省略可能化）
- float 型（現状は整数と文字列）
- 多次元配列変数又は連想配列
- printf のような書式つき表示（又は返信）
- キーボード, マウス, 音声認識, 疑似乱数との比較や変数代入
- 座標グラフィック（プロット, キャラクタ表示等）
- 音声出力（ＢＧＭ, 一時的効果音, 音声合成）
- 外部プログラムや一般 API の呼出可能化による手続利便性向上

謝辞　これまでの開発にあたり, ナレルシステム株式会社の安藤建剛氏, 木戸口卓矢氏並びに株式会社丸誠商会の川畑貴寛氏他の皆様方から多大なご支援を頂きました. また展示会・学会等で問題点等をご指摘・ご指導頂きました知識科学分野の諸先輩方に, この場をお借りして改めて感謝と御礼を申し上げます.

参考文献

[1] W.F.Clocksin, C.S.Mellish: **Programming in Prolog Fifth Edition**.Springer, 2003
[2] 中村圭介:**自由文による思考プログラミング**. 情報処理学会第 60 回プログラミングシンポジウム予稿集,2018
[3] 中村圭介: **特開 2014-211725 多長一致の無境界単一化法を含む基本特許**. 公開公報, 2014.
[4] 中村圭介: **特開 2017-091119 ネット最新知識の動的引用と引用スコープ**. 公開公報, 2017.
[5] 中村圭介: **特開 2017-102628 価値観や観点ごとの程度の定義と利用**. 公開公報, 2017.
[6] 中村圭介: **特開 2017-130070 コンピュータからの逆質問による判断**. 公開公報, 2017.
[7] 中村圭介: **特開 2018-032275 キャッシュによる探索高速化**. 公開公報, 2018.
[8] 中村圭介: **特開 2018-060298 言葉方程式を用いた自由文の利用方法等**. 公開公報, 2018.
[9] 中村圭介: **特開 2018-063509 難読文字等の音声の合成又は認識方法**. 公開公報, 2018.
[10] 中村圭介: **特開 2018-073015 自由文の目視による探索枝刈**. 公開公報, 2018.
[11] 中村圭介: **特開 2018-190180 自由文によるホーン節を用いた前向き推論**. 公開公報, 2018.
[12] 中村圭介: **特開 2018-190182 科学的・道徳的なタブーによる強い否定**. 公開公報, 2018.
[13] 中村圭介: **特開 2018-190184 自由文によるアイディアの自動合成**. 公開公報, 2018
[14] 中村圭介: **特開 2008-299681 コンテンツ提供方法**. 公開公報, 2008
[15] 中村圭介: **特開 2019-067162 複数コンテンツ間でデータやルールを共有**. 公開公報, 2019