4D-5

SIMPOSのプログラミング環境 ―コンパイラの拡張機能―

近藤 誠一 萩尾 弦一郎 石橋 弘義 近山 隆
(三菱電機(株)) (ビーコンシステム(株)) ((財)ICOT)

1. はじめに

逐次型推論マシン(PSI)上に開発されたプログラミング/オペレーティング・システム(SIMPOS)はPrologにオブジェクト指向機能を付加した言語ESPによるプログラミング環境を提供している。ESPのソース・プログラム中に現れる特定のパターンに対しては、マクロ展開機能が働き、コンパイラによって自動的にあらかじめ定められたパターンに展開される。本稿ではESP言語の特徴のひとつであるマクロ展開機能を中心にコンパイラの拡張機能について述べる。

Prologでは関数型言語と異なり、引数の場所に数式を直接記述することはできない。一般にデータを記述する箇所にその満足すべき性質を併せて記述することは、プログラムの読み易さ書き易さの上で有利である。ESPのマクロはこのような機能を実現するものである。ESPでは単にマクロの対象となったパターンを展開結果に置き換えるだけでなく、それを含むゴールの前後に必要なゴール列を挿入したり、新たに生成した節を付加する機能を持つ。

2. 概要

ESP言語は図1に示すように継承クラス、クラス・スロット、クラス・メソッドを定義するクラス定義部、インスタンス・スロット、インスタンス・メソッドを定義するインスタンス定義部、ローカル述語を定義するローカル定義部に分けられる。それぞれの部分は";"で終わるPrologの節の列により構成される。マクロ展開は、与えられた項をそれぞれの節に分解する際に行われるクラス定義レベルの展開とさらに各々の節の構成要素を対象とする節レベルの展開に分けて行われる。これらのマクロはマクロ・バンクと呼ばれる定義によりユーザが自由に設定することができる。また、プログラムの読み易さを向上させるために、ユーザに対し、演算子の追加・変更機能も提供している。

```
class   クラス名  has
        クラス定義部
  instance
        インスタンス定義部
  local
        ローカル定義部
end.
```

図1. クラス定義

3. 節レベルのマクロ展開

節レベルのマクロはPrologの節のヘッド部の各引数、ボディ部のゴール及びその引数を対象としている。節レベルのマクロは以下に示すように定義される。

〈対象パターン〉 ==> 〈展開後のパターン〉
when 〈生成用ゴールの列〉
where 〈検査用ゴールの列〉
with 〈付加節の列〉
:- 〈展開条件〉

下線部がキーワードである。〈対象パターン〉とユニファイ可能なパターンがソース・プログラム中に出現すると、指定した〈展開後のパターン〉に置き換える。その際、その〈展開後のパターン〉を生成するために特定の手続きが必要な場合が生じる。たとえば、図1に示すような場合、加算結果を得るためには事前に組込述語の加算を行わなければならない。このような、生成のための述語は〈生成用ゴールの列〉で指定する。これらは通常、対象パターンが出現したゴールの直前に置かれる。ただし、図2(3)のように〈対象パターン〉がヘッド部に出現した場合は、結果を呼出し側の述語に返すという意味で、ボディ・ゴール列の最後に置かれる。逆に、〈対象パターン〉を含むゴールの実行の後処理として検査等を行いたい場合、検査のための述語は〈検査用ゴールの列〉で指定する。これらは、対

(1) 節レベルのマクロ定義の例

```
X + Y => Z when add(X,Y,Z);
```

(2) ゴール部に出現した場合

```
p(X,Y) :- q(X+Y);
        ⇓
p(X,Y) :- add(X,Y,Z), q(Z);
```

(3) ヘッド部に出現した場合

```
p(X,Y+Z) :- q(X,Y,Z);
          ⇓
p(X,W) :- q(X,Y,Z), add(Y,Z,W);
```

図2. 節レベルのマクロの例

Programming Environment of SIMPOS …Extended Features of the ESP Compiler
Sei-ichi Kondoh, Gen-ichirou Hagio, Takashi Chikayama, Hiroyoshi Ishibashi
(Mitsubishi Electric) (Bcon System) (ICOT) (ICOT)

(1) withを含むマクロ定義

```
some(X in List, Condition,Tail) =>
      dummy(List,X,Tail)
  with (dummy([XIT],X,T) :- Condition,!;
        dummy([_IR],X,T) :- dummy(R,X,T));
```

(2) 展開例

```
p(List) :-
   some(X in List, (integer(X),X>10),Tail),
   q(X,Tail);
                    ⇓
p(List) :- dummy(List,X,Tail),q(X,Tail);
dummy([XIT],X,T) :- integer(X),X>10,!;
dummy([_IR],X,T) :- dummy(R,X,T);
```

図3．withを含むマクロの例

象パターンが出現したゴールの直後（ヘッド中ならばボディの先頭）に置かれる。

withで始まる〈付加述語の列〉は指定した節の列をローカル定義に加える機能を持つ。図3のように条件が繰り返しパターンのような場合有効である。また、ユニークにアトムを生成する機能も提供しているので、生成された述語に動的に名前を与えることもできる。

〈展開条件〉は通常の節のボディ部と同様の形式で、対象パターンの条件チェックや展開結果を生成するための述語呼び出しを記述することができる。これにより、コンパイル時にマクロの対象となったパターンを部分的に評価することもできる。

4．クラス定義レベルのマクロ展開

ESPのプログラムはそれ自体ひとつの大きな項から成り立っている。クラス定義レベルのマクロ展開は与えられた定義全体の項をもとに ESPの節までトップ・ダウンに分解していき各々のパターンをマクロの対象とする。この機能を用いた前処理により他言語のプログラムを ESPのプログラムに変換することもできる。クラス定義レベルのマクロは以下に示すように定義する。

〈対象パターン〉 ==> 〈展開後のパターン〉
with 〈付加節の列〉
:- 〈展開条件〉

各々の機能は節レベルのマクロと同様である。これにより、図4に示すように節をマクロの対象とすることができる。

5．マクロ・バンクの定義

マクロ・バンクの構成を図5に示した。前述のマクロ定義はこのマクロ・バンク内に記述する。 ESPのクラス定義内にてマクロ・バンク名を記述することにより、マクロを指定することができる。マクロ・バンクは以下のような特徴を持つ。

(1) マクロ定義

```
( Functor <- List ) ==> Expanded
      :- convert(List, Functor, Expanded) ;
convert([One],F,{F,One}) :- !;
convert([One|Rest],F,({F,One};Tail)) :-
      convert(Rest,F,Tail);
```

(2) 展開例

```
author <- [kondoh,hagio,ishibashi,chikayama]
                    ⇓
author(kondoh);     author(hagio);
author(ishibashi);  author(chikayama)
```

図4．クラス定義レベルのマクロの例

```
macro_bank  マクロ・バンク名  has
  nature  継承マクロ・バンク名 ;
     マクロ定義
  local
     ローカル定義
end.
```

図5．マクロ・バンクの定義

- 他のマクロ・バンクを多重継承して、それらのマクロ定義を自身に取込むことができる。
- 展開条件や展開結果を詳細に指定するために、ローカル述語を定義することができる。
- 指定した節をクラス定義の指定した部分（クラス、インスタンス、ローカル）に無条件に挿入する機能も提供している。

クラス定義の際マクロ・バンクを指定しない場合は、コンパイラの提供する標準のマクロ・バンクを用いて展開する。すべてのマクロ・バンクは自動的にこの標準のマクロ・バンクを継承する。

6．おわりに

ESPのマクロ展開機能を中心にコンパイラの拡張機能について示した。 DCG[1] はこの機能を用いて実現することができる。また、他の特殊なPrologの多く（項記述[2]，Fuzzy を入れたProlog[3] など）も、容易に実現することができる。個別の機能を付加して特殊化するよりも、一般性のある機能を実現して必要に応じて新たにマクロとして定義したり既存のマクロを組み合わせて実現するほうがより一般的で応用範囲が広い。

［参考文献］

[1] DECsystem-10 PROLOG USER'S MANUAL

[2] 戸村 "TDProlog: 項記述可能なProlog処理系" Logic Programming Conference '85

[3] I.P.Orci,H.Karlson "PROLOG/S Programming Language Based on Possibilistic Logic" Fourth Japanese-Swedish Workshop on Fifth Generation Computer Systems 1986 July 7-8