5J－9

論理プログラミング機能を埋め込んだ手続き型言語 c.

矢野稔裕、高田正之、小谷善行
（東京農工大学工学部電子情報工学科）

1．はじめに

論理型プログラミング言語であるPrologは、手続き型にはない特徴を数多くもつ。例えば

（1）ユニフィケーション処理
（2）非決定性処理
（3）プログラムの動的な操作

などである。このうち1と2の機能を手続き型言語であるC言語にもち込み、論理的プログラミングの手法を利用可能とする試みを行った。

複数の言語で記述されたプログラムを結合することは、すでに行われていることである。しかし、別々のパラダイムが融合された単一の言語を使うことによって、統一されたプログラミング環境を得ることができる。

2つの異なるパラダイムに基づく言語を融合しようとするとき、どちらを基礎とするかで生み出されたものの性格が異なってくる。言語c.は、C言語をもとにしてPrologの機能を埋め込んだことでPrologの持つ対話性、安全性などは失われたが、システム記述言語としての性格の強い論理プログラミング可能な言語となった。

2．言語仕様の概要

Prologの手続き（述語名とアリティが同じである節の集合）に相当する関数を拡張する。この関数を論理関数と呼び、型がlogicであると宣言する。n個の代替節に相当するn個のブロック（または単文）に分割されている。頭部引数のパターンに相当するpatternラベルが各ブロックの前に置かれる。ブロックの内部は、制限はあるがC言語の記述が可能であり、そこに含まれる論理関数の呼び出しがPrologでの本体のサブゴール呼び出しに相当する。

Prologのタグ付きデータ、リストや複合項の扱いもほとんどそのまま受け継がれ、プログラム中に項の表記が可能である。アトムについては宣言を要すことにした。

記述例を図1に示す。

3．実現手法

この試作では、早いプロトタイプの実現と見通しのよさを狙った実現法を採る。Cのプリプロセッサを通したc.ソースコードを、作製したトランスレータに通し、Prologのコンパイルドコードに相当するC言語のソースコードを生成する。

得られるコードは、データ内部表現、スタックの扱い、引数の扱いなど、ほとんどWAMモデル[1]に準じている。ただし実行制御は次に述べる方法による。

Prologでの手続きを、そのままC言語の関数に対応させて変換する。手続きpの成功による次の手続きqへの継続を、対応する関数pの中からの関数qの呼び出しに対応させ、手続きqの失敗によるバックトラックを、標準ライブラリ関数setjmp/longjmpを用いた大域ジャンプで実現する。関数qが成功すればリターンせずに次の手続きを呼び出す。論理関数は標準のリターン動作をせず、longjmpによってのみ制御を戻す。

選択点のBPフィールド（代替節へのポインタ）は削除され、相当する役目は代わりに選択点にとられたjmp_bufが担う。

CPレジスタの意味も変更する。2番目以降の論理関数の呼び出しは、それぞれの引数のput処理のコードと共に個別の補助関数として展開される。そしてそれらの関数へのポインタをもつstaticな配列が定義される。CPレジスタは、この配列の要素を指すポインタである。proceed処理はCPの指す要素中のポインタが指す関数の呼び出しとなり、call処理は、CPのインクリメント後の呼び出しとなる。

この方法では、無駄な返り番地がスタックに積まれること、選択点が大きくなることなどWAMの制御に比べ不利な点もある。しかしWAMのコードとよく対応した形となること、C言語だけで実現でき移植性、保守性が高いことなど、早期のプロトタイプ実現に非常に適している。

図1のプログラムの変換結果を図2に示す。

"c." : a Procedural Laguage with Logic Programming Feature
Toshihiro YANO, Masayuki TAKATA, Yoshiyuki KOTANI
Tokyo University of Agriculture and Technology

4. おわりに

この変換系によって、手続き型言語としての性能を損なうことなく論理プログラミング機能を利用可能となる。

Prologを用いた人工知能研究でよく見られるような、プログラムを動的に操作することに本質的に依存するプログラムは記述できないが、グローバルな記憶としてのassert/retractの利用は、この処理系では単にグローバルな変数を用いることで書くことができる。この点で、インタプリタと共存しているPrologコンパイラのシステムとは適用分野が異なってくる。エキスパートシステムやゲームシステムの記述に向くと考えられ、組み込みシステムの作製にも適している。

この試作ではC言語のみで記述してあり、W-AMモデルの実現に比べて効率は低下している。部分的に機械語で記述し直すことで、移植性を保ちつつ効率の改善を図ることが考えられる。

今後使用経験を重ねて、この言語仕様で手続き型言語に論理型言語の機能をエレガントに融合できたかどうか評価する。

参考文献

1) Warren, D.H.D.: An Abstract Prolog Instruction Set, SRI International Technical Note, No. 309 1983

2) J.L.Weiner, S.Ramakrishnan: A Piggy-back Compiler For Prolog, Proc. of SIGPLAN '88 Conference on Programing Language Design and Implementation Atlanta, Georgia, June 22-24, 1988

```
logic   append( TERM, TERM, TERM )
{
    pattern [], X, X:
        ;
    pattern [X|L1], L2, [X|L3]:
        append( L1, L2, L3 );
}
```

図1　記述例

```
logic   append()
{
    switch( deref(&A[0])->tag ){
    case RefTag:
        make_choice_point();
        if( !setjmp( B.cp->jb )){
    case AtomTag:
            if( get_nil( 0 ) &&
                get_value_x( 1, 2 )
            ){
                (*CP->goal)();
            }
            longjmp( B.cp->jb, Fail );
        }
        undo();
        discard_choice_point();
    case ListTag:
        if( get_list( 0 ) &&
            unify_variable_x( 3 ) &&
            unify_variable_x( 0 ) &&
            get_list( 2 ) &&
            unify_value_x( 3 ) &&
            unify_variable_x( 2 )
        ){
            append();
        }
        longjmp( B.cp->jb, Fail );
    case IntegTag:
    case StrucTag:
        longjmp( B.cp->jb, Fail );
    }
}
```

図2　変換例（読みやすくしてある）