記号処理 49－3
（1989. 1. 9）

リスト構造中間言語を用いた Prolog インタプリタの研究。

山里　拓己、　小谷　善行

東京農工大学工学部数理情報工学科

近年進んでいる Prolog の処理系の実行効率の向上の研究では、解釈実行される中間コードはレコード形式を持っている。しかし、動的な変化に対する、領域の処理や処理系自体の記述の簡潔さ等においてはリスト方式が有利である。本稿ではこの方式 Prolog の処理系について報告する。システムは、節の入力ごとにプログラムを中間コードに翻訳する。この中間コードはリスト形式でシステム内に保持されている。中間コードは位置によらず同じ形をしている。処理系はこの中間コードをその場所に応じて解釈し実行する。こうした方法により、柔軟でコンパクトな処理系を開発した。

A Prolog System with List-Structured Intermediate Code

Takumi Yamasato, Yoshiyuki Kotani

Dept. of Information Science
Tokyo University of Agriculture and Technology
24-16 Nakamachi 2-Chome Koganei-shi Tokyo Japan

In current studies of performance of Prolog processors, intermediate codes are represented in sequential form. On the other hand, list-structured intermediate code has more advantage in processing of dynamical change of clauses, description of system, etc.. We report a Prolog system by the method in this paper. It compiles each clause of a source program to a list of intermediate code, one bye one. The intermediate code has a unique list-structured form, whereever it may be in the clause. It works by interpreting the list according to its situation. The system was developed as a flexible, compact one.

1. はじめに

近年、Prologの処理系は知識処理言語として、その実行の際の効率の向上のため多くの研究がなされてきた。その場合、解釈実行される中間コードのほとんどが、レコード形式を持ち、記憶効率や処理速度の高速化を目指している。しかし、動的な変化に対する、領域の処理や処理系自体の記述の簡潔さ等においてはリスト方式が有利である。本稿ではこうした考えによるPrologの処理系について論じる。システムは、節の入力ごとにプログラムを中間言語（PLM）[1]に基づいた中間コードに翻訳する。この中間コードはリスト形式でシステム内に保持されている。処理系はこの中間コードを解釈し実行する。こうした方法により、Prologが持つデータとプログラムの等価性を反映させ、柔軟でコンパクトな処理系を開発した。

この方式の最初の版は[2], [3]に述べている。これに対して、効率を考慮して中間コードの仕様を大幅に改良している。おもに、中間言語であるPLM命令の簡略化、同一化により処理系がコンパクトとなっている。

2. 中間コード

2. 1 中間コード

この処理系では、ソースプログラムの節が入力されるごとに内部形式であるリスト構造を持つ中間コードへとコンパイルされる。この特徴は次のようなものである。

（1）項の引き数を含め、すべてがリスト形式（ポインタによるリンク構造）を持つ。

（2）項のレベル、およびゴール節か定義節か否かによらず、同じ形は同じリストになる。

この中間コードはWarrenのPLM命令の命令体系に基づいている。一般に、こうした命令体系では、項に対する引き数は連続の領域に置かれ、中間言語中にその引き数番号を持たせることにより引き数へのアクセスをしている。一方、われわれの方式では全ての内部形式をリスト構造で持つ。そのため、項に対する引き数もリストで表現される。

引き数にアクセスするには、そのリストのポインタをたどればよい。Prologでは、引き数は左から右へ出現した順に番号付けされるため、ユニファイの実行もこの順序で行われる。したがって、引き数位置を示す先頭からの番号を保持する必要はない。

また、Warrenの命令ではスケルトン中に現れるレベル1についてのユニファイ命令は、レベル0の命令とは異なる形式を持つ。プログラムであるユニファイ命令とデータを同一として扱うためにはこれらは同じ形式で表されなくてはならない。ここではレベル1の命令はifdone命令により大域変数を用いた判定フラグをたてることによって区別するものとし、中間コードとしてはレベル0とレベル1は同型の形式を持つ。

一方、定義節とゴール節とのあいだでも、対応する中間コードをおなじものとしている。例えば、example(nil,X,L,L).のプログラムのソース節、及び、ゴール節 :-example(nil,X,L,L),print(L) は、図1、図2の中間コードへ変換される。

```
example:enter
        try(c1)
c1:     atom(nil)
        void
        local(0)
        local(0)
        foot(4)
```

図１．　”example(nil,X,L,L).”に対する中間コード

```
        localinit(0,1)
        neck(1,0)
        call(ecall)
        call(pcall)
ecall:  callarg(example)
        atom(nil)
        void
        local(0)
        local(0)
        foot(4)
```

図２．　”:-example(nil,X,L,L),print(L).”に対する中間コード

図１、図２では、命令を並べた表記で表しているが、実際には、１列につながったリストである。また、そのなかのラベルも実際には、ポインタである。たとえば、try（c1）のなかのc1は、c1:の場所、つまりatom（nil）以下のリストへのポインタである。

こうして、中間コードはプログラム中のどこにあっても同じ形をとる。このようにすることにより、任意の複合項データは、実行時にダイナミックにそのままゴール節としても、定義節としても扱うことができるのである（表１）。

２．２　中間コードセル

ひとつの中間コードは内部的にはひとつのリストのセルとして格納されている。このセルは、C言語の共用体により図３のように表されており、すべてのセルが同じように扱うことができる。このため、処理系の簡潔化、及びコンパイラの手間の軽減が可能となる。プログラムは節ごとにこのようなセルを持つ木構造に形成されている。Cの再帰的呼び出しによりこの実現を容易にしている。また、動的な削除等でできた不要セルはすべて回収時にこの木をたどって解放される。

表１．　統一される中間コードの例

	レベル０	レベル１
Warrenの方式	uatom(a,1)	uatom1(a)
本方式	atom(a)	

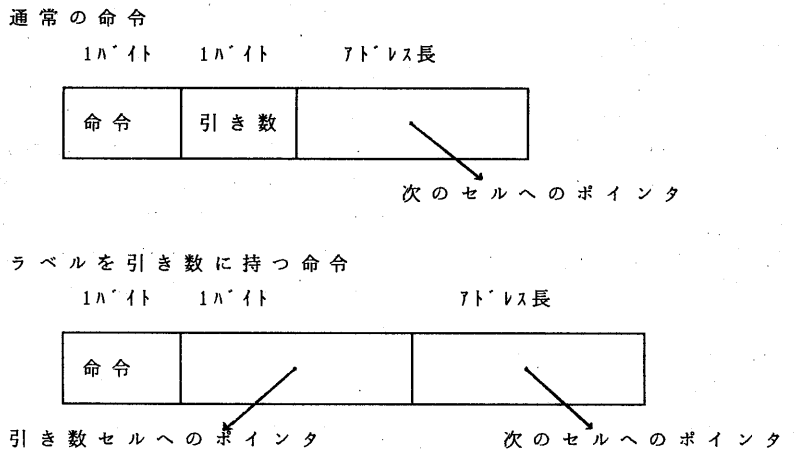

図３．　セル構造

３．データとプログラムの等価性

PLM では、現在の節（current clause）と現在のゴール（current goal）でのデータ形式は異なるものであった。しかし、ここではそのデータが、同じ型のものならば節中に現れても、ゴール中に現れても同じセルをつくる。つまり、プログラムとデータの区別は、プログラムカウンタ（PC）がそのセルを示すか、そのセルが参照の対象かによってそれらを区別する。このことにより、スタック上である変数をインスタンシエートする場合、そのセルのポインタの代入だけでよく、ソースプログラムから中間言語へコンパイルする処理系とそれを実行する処理系がかなり簡潔化される。

PC がセル atom(a) を示しているならば、この命令はユニフィケーション命令として扱われ、このセルが参照の場合はデータとして扱われる。また、PC が示しているセル自体もユニフィケーションの結果によってデータとして扱われることになる。たとえば、ゴールの参照する結果が UNDEF ならばユニフィケーション命令である atom(a) のセルのポインタが参照先に代入されることになる（図４）。

４．　まとめ

PLM 命令をリスト形式に対応させた処理系を作成した。中間言語のリスト形式が実行にそった木構造のためＣ言語の再帰呼び出しを利用でき、またセルの統一性からこの PLM 実行システムはＣ言語で約 700 行程度となっている。

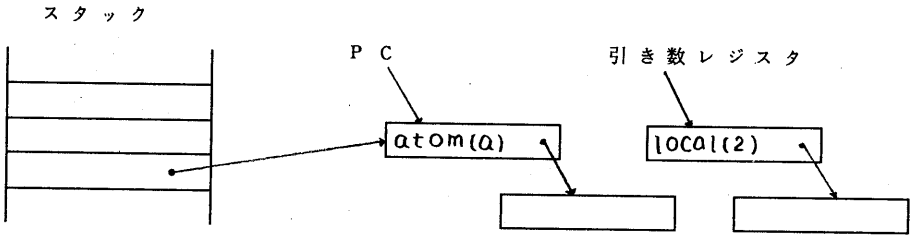

図４　変数のインスタンシエート

参考文献

［１］　Warren,D.H.D: Implementing Prolog-Compiling Predicate Logic Programs,D.A.I.ResearchReport,No.39 and 40.(1977)

［２］　山里拓己: 日本語論理型プログラム言語の処理系の試作　-中間言語の解釈系-，東京農工大学工学部数理情報工学科　卒業論文(1987)

［３］　末永富美代: 日本語論理型プログラム言語の処理系の試作　-中間言語の翻訳系-，東京農工大学工学部数理情報工学科　卒業論文(1987)