論 文

Technical Papers

Prolog Cafe: Java上で動作するProlog処理系

Prolog Cafe: A Prolog to Java Translator System

田村 直之
Naoyuki Tamura
神戸大学 学術情報基盤センター
Information Science and Technology Center, Kobe University
tamura@kobe-u.ac.jp

番原 睦則
Mutsunori Banbara
（同　上）
banbara@kobe-u.ac.jp

keywords: Prolog, Java, logic programming, translator

1. はじめに

Prolog (PROgramming in LOGic) は，1974 年にフランスの Colmerauer らにより開発されたプログラミング言語である．特に AI プログラムの記述・開発に適していると言われ，日本では通産省の第五世代コンピュータプロジェクトの核言語として採用されたことでも知られる．

Prolog の実装に関しては，WAM 抽象機械 (Warren Abstract Machine) [Aït-Kaci 91, Warren 83] が事実上の標準となっており，これまで数多くの高速コンパイラ処理系が開発されている：SICStus Prolog, K-Prolog, IF/Prolog, Ciao Prolog, SWI Prolog, GNU Prolog, B-Prolog, YAP, BinProlog, Strawberry Prolog, MINERVA, Jinni 他 *1.

一方，Java によるネットワークプログラミングが急速に進歩しているが，ネットワーク上の自然言語処理，マルチエージェントシステムなど，ある種の知的処理を伴うソフトウェアの開発となると，部分的にでも Prolog のような特殊言語に任せたくなるのも事実である．そのため，近年 Java による Prolog 処理系が数多く提案されている．また SICStus Prolog などの WAM ベースの高速 Prolog コンパイラ処理系の多くが，Java インタフェイスを備えている．

Prolog ユーザから見た場合，効率性だけを重視するのであれば，高速 Prolog コンパイラの提供する Java インタフェイスで十分である．しかし，利便性・拡張性は制限され，Java の機能をフル活用することは難しい．Java ユーザから見た場合，既存の Prolog 処理系では 100%純 Java かつ効率の良いアプリケーションを開発することは困難である．

以上のような背景から，本稿では，Prolog から Java へのトランスレータ処理系 Prolog Cafe について述べる．本システムでは，Prolog プログラムは，Java プログラムに変換され，既存の Java 処理系を用いてコンパイル・実行される．つまり Prolog Cafe では，項，述語など Prolog の構成要素のすべてが Java に変換される．このため， Prolog Cafe は移植性，拡張性に優れた Prolog 処理系となっている．

*1 これら処理系の詳しい情報，入手先 URL などに関しては，http://bach.istc.kobe-u.ac.jp/prolog/を参照.

2. Prolog Cafe の概要

2·1 Prolog Cafe の特徴

Prolog Cafe [Banbara 05] は，以下のような特徴をもつシステムである．

100% Pure Java: Prolog Cafe はすべての機能が Java 上で実現された Prolog 処理系であり，Java さえインストールされていればすぐに利用可能である．

インタプリタとトランスレータ: インタプリタとコンパイラ (Java へのトランスレータ) が用意されており，開発ステップと運用ステップでの使い分けが可能である．

Java の利用: Prolog から Java プログラムの呼出し (すなわち，Java のクラスに対してオブジェクトの生成，メソッドの呼出し，フィールドへのアクセス等)，また Java プログラムから Prolog 述語の呼出し等が可能であり，Java との親和性が非常に高い．

実用性: 算術演算 (多倍長整数を含む)，ファイル入出力，`assert`/`retract` 等，標準的な Prolog 処理系に存在する組込述語はほぼ用意されている (ISO Prolog の規格を参考にしている)．さらに，豊富な Java のライブラリを利用して実用的なプログラム開発を行える．

実行速度: フリーの Prolog 処理系として広く利用されている SWI Prolog と比較しても，3 倍程度遅いだけであり (トランスレータの場合)，十分実用に耐え得る．

利用実績: Prolog Cafe は，1997 年頃から開発を開始

したシステムである．すでに，多くの方々に利用していただいている．以下に主な例を挙げる．

- P# [Cook 04]
  P# は，Microsoft .NET 環境で動作する Prolog 処理系であり，Prolog から C#へのトランスレータ処理系として実装されている．
- Multisat [Inoue 06, Soh 05]
  Multisat は，複数 SAT ソルバの並列実行処理システムである．単なる SAT 問題だけでなく，ジョブショップスケジューリング問題などへも応用されている．Prolog は複数 SAT ソルバの並行実行，スケジューリング等に用いられている．
- Maglog [Kawamura 04]
  Maglog は，モバイルエージェントシステムを記述するために，Prolog を拡張した言語および，その処理系である．Java RMI を用いた強マイグレーション機能，“場 (Field)” と呼ばれるエージェントのための共有空間が特徴である．
- HoloJava [Barbosa 02]
  マルチパラダイム指向言語 Holo から Java へのトランスレータ処理系である．Holo は，論理アクション (logic actions) として，Prolog プログラムをフィールド中に記述することができる．
- CAWOM [Wohlstadter 01]
  CAWOM は，レガシーシステムのためのラッパーを作成するプログラムである．その一部，構文解析のために Prolog (特に DCG) が使われている．

2·2 Prolog Cafe のインストール

Prolog Cafe を利用するためには，Java 1.4 以上が利用できる環境が必要がある．Prolog インタプリタの実行には Java 実行環境 (J2SE JRE) がインストールされていれば十分だが，自分で作成した Prolog プログラムを Java プログラムに変換して実行するには J2SE の開発環境が必要となる．

Java 開発環境が正しくインストールされているかどうかは，コマンドプロンプト (Windows XP の場合) やターミナル等を開き，以下のように “`java`” および “`javac`” が起動できるかどうかで確認する．

```
> java -version
java version "1.5.0_06"
.....
> javac -version
javac 1.5.0_06
.....
```

Prolog Cafe 自体のインストールは簡単である．Prolog Cafe の Web ページ*2から最新の Prolog Cafe パッケージ (現在は `PrologCafe091.zip`) をダウンロードして，適当なディレクトリで展開し，環境変数 `PLCAFEDIR` が展開されたディレクトリを指すようにすれば良い．環境変数が正しく設定されているかどうかは，コマンドプロンプトで以下のように入力して確認する (Unix 系 OS の場合は “`echo $PLCAFEDIR`”)．

```
> echo %PLCAFEDIR%
C:¥PrologCafe091
```

なお，Mac OS X, Linux 等の Unix 系の OS 用には，コンパイルや実行のための Perl スクリプトが用意されているが，本稿では説明を省略し，スクリプトを利用せず直接起動する方法について説明する．

3. Prolog Cafe 入門

本節では，Prolog Cafe パッケージ中にも例題として含まれている N クイーンプログラムを例題として，Prolog Cafe インタプリタおよびトランスレータでの実行方法について説明する．なお，N クイーンプログラムは，$N \times N$ のチェス盤上で，N 個のクイーンが互いに取られないような配置を探索するプログラムである．

本稿では，Prolog から Java へのトランスレータ処理系 Prolog Cafe について，その利用方法に重点をおいて述べる．そのため，Prolog および Java 自身 (言語仕様，プログラミング等) については，別途入門書を参照していただきたい．Prolog の入門書としては，[Sterling 88] [後藤 84], [黒川 85], [塚本 89] 等がある．また，神戸大学 Prolog ホームページ*3にも，簡単な Prolog 入門がある．Java に関する最新情報については，開発元である SUN のサイト*4から入手可能である．

3·1 Prolog Cafe インタプリタによる実行

インタプリタによる実行では，Prolog プログラムは `Term` オブジェクト (後述) としてメモリ上に記憶され，Prolog インタプリタによって解釈実行される．したがって，インタプリタによる実行は，後述のトランスレータによる実行と比較して，手軽であるが実行速度は遅い (問題にもよるが 10 倍以上の実行時間が必要となる)．

まず，図 1 のプログラムをメモ帳等のエディタで入力し，ファイル名を `queens.pl` として保存する．次に，コマンドプロンプトあるいはターミナル等を起動し，“`cd`” コマンドを用いて `queens.pl` を保存したディレクトリに移動する．

続けて Prolog インタプリタを起動するが，これは Windows XP 等の Windows 系 OS の場合は以下のように入力する (←の部分は，改行せずに空白を開けて続けて入力する)．

```
> java -classpath ←
  "%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria
```

*2 http://kaminari.istc.kobe-u.ac.jp/PrologCafe/

*3 http://bach.istc.kobe-u.ac.jp/prolog/intro/

*4 http://java.sun.com/

```
main :-
    queens(8, Qs), write(Qs), nl, fail.

queens(N,Qs) :-
    range(1,N,Ns), queens(Ns,[],Qs).

queens([],Qs,Qs).
queens(UnplacedQs,SafeQs,Qs) :-
    select(UnplacedQs,UnplacedQs1,Q),
    not_attack(SafeQs,Q),
    queens(UnplacedQs1,[Q|SafeQs],Qs).

not_attack(Xs,X) :- not_attack(Xs,X,1).

not_attack([],_,_) :- !.
not_attack([Y|Ys],X,N) :-
    X =¥= Y+N, X =¥= Y-N, N1 is N+1,
    not_attack(Ys,X,N1).

select([X|Xs],Xs,X).
select([Y|Ys],[Y|Zs],X) :- select(Ys,Zs,X).

range(N,N,[N]) :- !.
range(M,N,[M|Ns]) :-
    M < N, M1 is M+1, range(M1,N,Ns).
```

図 1 *N* クイーンプログラム (queens.pl)

ここで，%PLCAFEDIR%¥plcafe.jar が Prolog Cafe の実行環境とコンパイル環境を含んだ jar ファイル[*5]，クラス jp.ac.kobe_u.cs.prolog.lang.PrologMain が Prolog Cafe のメインプログラム，その後の引数 jp.ac.kobe_u.cs.prolog.builtin:cafeteria がメインプログラムから起動される述語名であり，Prolog インタプリタ本体である[*6]．

Mac OS X, Linux 等の Unix 系の OS の場合は，同様にターミナルウィンドウ中で“cd”コマンドを用いて queens.pl を保存したディレクトリに移動し，以下のように入力する．

```
$ java -classpath ←
  "$PLCAFEDIR/plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria
```

以下，本解説での説明は Windows 系 OS を主とし，Unix 系 OS での入力方法については説明を省略する．Unix 系 OS の場合，環境変数の指定が%PLCAFEDIR%でなく$PLCAFEDIR である点，ディレクトリの区切りが ¥ でなく/である点，classpath の区切りがセミコロン (;) でなくコロン (:) である点にさえ注意すれば，後は同様に実行できる．

インタプリタが起動すると，図 2 のような表示が現れる．入力“[queens].”により，*N* クイーンプログラム queens.pl が読み込まれ (consult され)，入力“main.”で Prolog プログラムが実行される．

```
Prolog Cafe 0.9.1
Copyright(C) 1997-2004 M.Banbara and N.Tamura
| ?- [queens].     % queens.pl を読み込む
.....
yes
| ?- main.         % プログラムの実行
[4,2,7,3,6,8,5,1]
.....
[5,7,2,6,3,1,4,8]

no
| ?- halt.         % インタプリタの停止
bye
```

図 2 *N* クイーンプログラムのインタプリタによる実行

このように Prolog Cafe は，plcafe.jar ファイルと Java の実行環境さえあれば，Prolog プログラムをインタプリタ上で実行できるという点で，非常にポータビリティの高いシステムとなっている．

3·2 Prolog Cafe トランスレータによる実行

Prolog Cafe では，Prolog プログラムを Java プログラムに変換 (トランスレート) し，さらに Java コンパイラでコンパイルした上で実行することも可能である．この場合，トランスレートという手間はかかるが，インタプリタと比較して大幅に実行速度は向上する．

まず，work ディレクトリを作成し，そのディレクトリ中で，先ほどの Prolog プログラム queens.pl を Java プログラムに変換する．

```
> mkdir work
> cd work
> java -classpath ←
  "%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.compiler.Compiler ←
  ..¥queens.pl
```

ここで，クラスファイル jp.ac.kobe_u.cs.prolog.compiler.Compiler がトランスレータのクラスであり，..¥queens.pl が変換対象の Prolog プログラムファイルである．

変換が行われると，プログラム中の各述語に対応して，PRED_main_0.java など，PRED_*p*_*n*.java (*p* は述語名，*n* は引数の個数) といった名前のファイルがカレントディレクトリ中に作成される．

次に，変換により作成された Java プログラムを以下のようにして Java コンパイラでコンパイルする．

```
> javac -d . -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" *.java
```

これで，queens.pl 中のすべての述語がカレントディレクトリ中にコンパイルされた状態になっているので，以下のように Java の classpath にカレントディレクトリを

*5 jar ファイルとは，複数の Java クラスファイル等を圧縮して格納したファイルであり，ライブラリとして利用される．環境変数 PLCAFEDIR に空白が含まれていても良いように，ダブルクォートでくくっている．

*6 コロンの前の jp.ac.kobe_u.cs.prolog.builtin がパッケージ名，コロンの後の cafeteria が実際の述語名となっている．

追加して実行する．

```
> java -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria

Prolog Cafe 0.9.1
Copyright(C) 1997-2004 M.Banbara and N.Tamura
| ?- main.
.....
```

classpath にカレントディレクトリを追加することで，N クイーンプログラムは実行可能となっている．したがって，“[queens].”と入力してプログラムを読み込ませる必要はなく，インタプリタに対して“main.”と入力すれば直ちに実行が開始される．

以下のように引数として述語名 main を指定すれば，インタプリタを介さずに直接実行することも可能である．

```
> java -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  main
```

さらに，以下のようにすればクラスファイルを jar ファイルにまとめた上で実行できる．

```
> jar cf ..¥queens.jar *.class
> cd ..
> java -classpath ←
  "queens.jar;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  main
```

この場合，実行に必要なものは plcafe.jar および queens.jar の二つの jar ファイルだけであり，work ディレクトリ中の Java ファイルやクラスファイルは必要ない．

4. Prolog Cafe と Java との連携

4·1 Prolog から Java の呼出し

Prolog Cafe には以下の述語が用意されており，Prolog プログラム中から簡単に Java プログラムを呼び出すことができる．なお，これらの述語は Prolog Cafe インタプリタからでも利用可能である．

- java_constructor(C，O)：クラス名 C のオブジェクトを生成し，O にバインドする．
- java_method(O，M，R)：オブジェクト O のメソッド M を呼び出し，返り値を R にバインドする．あるいは O がクラス名の場合は，クラス O の static メソッド M を呼び出し，返り値を R にバインドする．
- java_get_field(O，F，V)：オブジェクト O のフィールド F の値を V にバインドする．あるいは O がクラス名の場合は，クラス O の static フィールド F の値を V にバインドする．
- java_set_field(O，F，V)：オブジェクト O のフィールド F に値 V をセットする．あるいは O がクラス名の場合は，クラス O の static フィールド F に値 V をセットする．

ただし，これらの述語の引数のうち，C と M の引数 (すなわち，Java のコンストラクタ，あるいはメソッドに渡される引数) が，Prolog のアトム，整数，リスト等の場合，それぞれ Java の String, Integer, Vector 等のデータに自動的に変換される．たとえば，Prolog Cafe インタプリタで以下のように入力すれば，java.awt.Frame オブジェクトが生成され，サイズが 200×200 にセットされた後，画面上にそのウィンドウが現れる．

```
| ?- java_constructor('java.awt.Frame', X),
 java_method(X, setSize(200,200), _),
 java_get_field('java.lang.Boolean', 'TRUE', T),
 java_method(X, setVisible(T), _).
```

次に，前節の N クイーンプログラムで求めた解を，Java を用いてグラフィカルに表示する方法について述べる．

まず，図 3 のプログラムを QueensFrame.java として作成し，queens.pl と同じディレクトリに保存して，以下のようにしてコンパイルする．

```
> javac QueensFrame.java
```

次に，queens.pl を編集し，以下の行を加える．

```
show :-
  java_constructor('QueensFrame', Frame),
  queens(8, Qs),
  java_method(Frame, setQueens(Qs), _),
  W = 1000,
  java_method('java.lang.Thread', sleep(W), _),
  fail.
```

このプログラムをインタプリタで動作させる場合，以下のようにする．カレントディレクトリ中の QueensFrame.class を実行するため，Java の classpath にカレントディレクトリを含めている．

```
> java -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria

Prolog Cafe 0.9.1
Copyright(C) 1997-2004 M.Banbara and N.Tamura
| ?- [queens].      % queens.pl を読み込む
.....
yes
| ?- show.          % プログラムの実行
.....
```

述語 show の実行により，まず QueensFrame オブジェクトが生成され，画面上にフレームが表示される．次に，述語 queens(8，Qs) で求められた解が setQueens メソッドの呼出しによりフレーム上に描画される (図 4 参照)．なおメソッド呼出しの際，解である整数のリストは，Integer を要素とする Vector に変換される．java.lang.Thread クラスの sleep メソッドの呼出しにより，1 秒待った後，fail の実行により，バックトラックが起こり，別の解が探索される．

このように Prolog Cafe では，Prolog 内で Java オブジェクトを直接取り扱うプログラムを記述可能である．

```
import java.awt.*;
import java.util.Vector;

public class QueensFrame extends Frame {
  QueensPanel queensPanel;

  public QueensFrame() {
    setSize(500, 500);
    queensPanel = new QueensPanel();
    add(queensPanel);
    setVisible(true);
  }

  public void setQueens(Vector queens) {
    queensPanel.setQueens(queens);
  }
}

class QueensPanel extends Panel {
  Vector qs = null;

  public void setQueens(Vector queens) {
    qs = queens;
    repaint();
  }

  public void paint(Graphics g) {
    if (qs == null)
      return;
    int c = getSize().height / qs.size();
    Color[] bg = { Color.white, Color.gray };
    for (int i = 0; i < qs.size(); i++) {
      for (int j = 0; j < qs.size(); j++) {
        g.setColor(bg[(i + j) % 2]);
        g.fillRect(i * c, j * c, c, c);
      }
      g.setColor(Color.orange);
      int j = ((Integer)qs.get(i)).intValue()-1;
      int d = c / 10;
      g.fillOval(i*c+d, j*c+d, c-2*d, c-2*d);
    }
  }
}
```

図 3 *N* クイーン結果表示プログラム (QueensFrame.java)

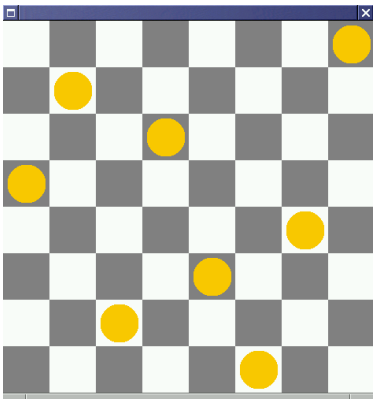

図 4 *N* クイーンの結果

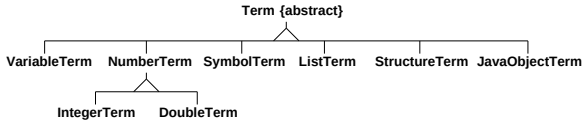

図 5 項のクラス図

4·2 Prolog データの Java での表現形式

Java プログラムから Prolog プログラムを呼び出す方法を説明する前に，Prolog データが Java オブジェクトとしてどのように表現されているかについて簡単に説明する．

一般に項と呼ばれる Prolog データは，VariableTerm, IntegerTerm, DoubleTerm, SymbolTerm, ListTerm, StructureTerm のいずれかのクラスのオブジェクトに変換される (図 5 参照)．

Term クラスは全ての項のスーパークラスであり，抽象メソッド unify をもつ．JavaObjectTerm クラスは，任意の Java オブジェクトを Prolog の項として取り扱うために使われる．つまり Java オブジェクトをこのクラスでラップすることにより，Prolog の項として利用可能となる．

4·3 Java から Prolog の呼出し

Java プログラムから Prolog 述語 p を呼出したい場合は，以下の手順で行う．

(1) Prolog の実行エンジンである PrologControl オブジェクト e を生成する．
(2) 実行したい述語のオブジェクト p を生成する[*7]．
(3) 述語の引数の配列 a を生成する
(4) Prolog の実行エンジン e に対して，実行したい述語 p と引数配列 a を setPredicate メソッドでセットする．
(5) e に対して call メソッドを実行することで，最初の解の探索が始まる．返り値が真なら成功，偽なら失敗である．
(6) e に対して redo メソッドを実行することで，別解の探索を再開する．返り値が真なら成功，偽なら失敗である．

このような call および redo による探索の制御は，Prolog の計算モデルとして知られているボックスモデルに基づいている．

ゴール “queens(8, Qs)” と同様の呼出しを行う Java プログラムを図 6 に示す．求まった解は，toString メソッドにより文字列に変換され，すべてが順に表示される．

このプログラム (Queens.java) をコンパイルするには，classpath に plcafe.jar と第 3·2 節で作成した queens.jar を加え，以下のようにする．

*7 Prolog Cafe のトランスレータでは，n 引数の述語 p は Java プログラム PRED_p_n.java に変換される．例えば，図 1 中の述語 queens/2 は，PRED_queens_2.java に変換される．

```
import jp.ac.kobe_u.cs.prolog.lang.*;

public class Queens {
  static void queens(int n) {
    PrologControl e = new PrologControl();
    Predicate p = new PRED_queens_2();
    Term arg1 = new IntegerTerm(n);
    Term arg2 = new VariableTerm();
    Term[] a = { arg1, arg2 };
    e.setPredicate(p, a);
    for (boolean r = e.call(); r; r = e.redo()) {
      System.out.println(arg2.toString());
    }
  }

  public static void main(String[] args) {
    queens(8);
  }
}
```

図 6 Java から Prolog の呼出し (Queens.java)

```
> javac -classpath ←
  "queens.jar;%PLCAFEDIR%¥plcafe.jar" ←
  Queens.java
```

実行は，classpath にカレントディレクトリを追加した形で以下のようにする．

```
> java -classpath ←
  ".;queens.jar;%PLCAFEDIR%¥plcafe.jar" ←
  Queens
```

このように Prolog Cafe では，Java プログラムから Prolog 述語を簡単に呼び出すことができる．

同様にして，Web ブラウザ上で動作する Java のアプレットから Prolog プログラムを呼出すことも可能である．動作例は，Prolog Cafe の Web ページ，あるいは Prolog Cafe パッケージの applet ディレクトリ中のプログラムで確認できる．

```
> cd %PLCAFEDIR%¥applet
> appletviewer Queens.html
```

4·4 ユーザ組込述語の作成方法

前述のように Prolog Cafe のトランスレータでは，n 引数の述語 p は Java プログラム PRED_p_n.java に変換される．逆に考えると，PRED_p_n.java という Java プログラムを作成すれば，組込述語を実現できる．

ここでは，画面に確認用のポップアップを表示するための述語 msgbox を作成することにする．

Java プログラムファイルをゼロから作成するのは手間がかかるので，“msgbox(_).” という一行だけから成るファイル msgbox.pl を作成し，3·2 節に従ってトランスレータを用いれば PRED_msgbox_1.java が生成される．

図 7 は生成された Java プログラムを編集し，“import javax.swing.*;” の 1 行の追加と，exec メソッド中に 5 行を追加し，述語 msgbox を実現したものである．

このプログラムをコンパイルするには，トランスレータで生成された Java プログラムのコンパイルと同様に以下のようにすれば良い．

```
import jp.ac.kobe_u.cs.prolog.lang.*;
import jp.ac.kobe_u.cs.prolog.builtin.*;
import javax.swing.*;

public class PRED_msgbox_1 extends Predicate {
  Term arg1;

  public PRED_msgbox_1(Term a1, Predicate cont) {
    arg1 = a1; this.cont = cont;
  }

  public PRED_msgbox_1() {}

  public void setArgument(Term[] args,
                          Predicate cont) {
    arg1 = args[0]; this.cont = cont;
  }

  public Predicate exec(Prolog engine) {
    Term a1 = arg1.dereference();
    String msg = a1.toString();
    JFrame frame = new JFrame();
    frame.setSize(300, 100);
    JOptionPane.showMessageDialog(frame, msg);
    return cont;
  }

  public int arity() { return 1; }

  public String toString() {
    return "msgbox(" + arg1 + ")";
  }
}
```

図 7 組込述語の例 (PRED_msgbox_1.java)

```
> javac -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" PRED_msgbox_1.java
```

実行は，classpath にカレントディレクトリを追加した形で以下のようにインタプリタを起動し，ゴールとして“msgbox(’メッセージ’).” と入力すれば，画面上にポップアップが表示される．

```
> java -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria

Prolog Cafe 0.9.1
Copyright(C) 1997-2004 M.Banbara and N.Tamura
| ?- msgbox(’メッセージ’).
yes
```

このように Prolog Cafe では，一つの Java ファイルを作成するだけで，組込述語の追加を非常に簡単に行うことができる．

4·5 マルチスレッドの利用

Prolog の実行エンジンである PrologControl オブジェクトは，複数生成することが可能である．

```
start(G0, Engine) :-
  copy_term(G0, G),
  java_constructor(
    'jp.ac.kobe_u.cs.prolog.lang.PrologControl',
    Engine),
  java_wrap_term((copy_term(G, G1), call(G1)),
                 NewG),
  java_method(Engine, setPredicate(NewG), _),
  java_method(Engine, start, _).

in_success(Engine) :-
  java_get_field('java.lang.Boolean', 'TRUE',
                 True),
  java_method(Engine, in_success, True).

in_failure(Engine) :-
  java_get_field('java.lang.Boolean', 'TRUE',
                 True),
  java_method(Engine, in_failure, True).

ready(Engine) :-
  java_get_field('java.lang.Boolean', 'TRUE',
                 True),
  java_method(Engine, ready, True).

result(Engine, Result) :-
  java_method(Engine, next, Result).

cont(Engine) :-
  java_method(Engine, cont, _).

stop(Engine) :-
  java_method(Engine, stop, _).

join(Engine) :-
  java_method(Engine, join, _).

sleep(Wait) :-
  java_method('java.lang.Thread',
              sleep(Wait), _).
```

図 8 マルチスレッド実行制御の例 (thread.pl)

したがって，複数のスレッドを生成し，それぞれで Prolog プログラムを実行できる．さらに，そのような並列処理を Prolog プログラム中から実行可能である．

詳細の説明は省略するが，図 8 がそのようなマルチスレッド実行のための制御プログラム例 thread.pl である．例えば，以下のように入力すると二つの 8 クイーンプログラムが並列実行される．ここで classpath 中の queens.jar は，第 3·2 節で作成した jar ファイルである．

```
> java -classpath ←
  "queens.jar;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria

| ?- [thread].
.....
| ?- start(main, E1), start(main, E2),
     join(E1), join(E2).
```

Prolog インタプリタではゴールの実行後，最初に見つかった解 (ゴール中に現れる変数のバインド情報) が表示され，入力待ち状態となる．ここでリターンを押すと終了，セミコロン (;) を入力後リターンを押すと次の解の探索を開始する．よって，上記ゴールを実行した場合，8 クイーンの解が出力された後，変数 E1 と E2 のバインド情報が表示され，入力待ち状態となる．

4·6 パッケージの利用

Prolog Cafe では，Java のパッケージ機能をそのまま利用して，Prolog プログラムのパッケージ化が行える．たとえば，Prolog Cafe の組込述語は jp.ac.kobe_u.cs.prolog.builtin というパッケージ中に定義されている．

Prolog Cafe のパッケージ機能を利用するのは簡単である．たとえば，queens.pl プログラムの先頭に以下のように記述を一行追加すれば (queens2.pl とする)，プログラム中のすべての述語がパッケージ queens 内の定義となる．

```
:- package queens.
```

このプログラムを 3·2 節と同様に Java プログラムに変換およびコンパイルを行う．

```
> mkdir work2
> cd work2
> java -classpath ←
  "%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.compiler.Compiler ←
  ..¥queens2.pl
> javac -d . -classpath ←
  ".;%PLCAFEDIR%¥plcafe.jar" *.java
```

すると，パッケージ名に対応して queens ディレクトリが作成されるので，以下のようにして queens2.jar ファイルを作成し，Prolog Cafe 起動時にその queens2.jar ファイルを指定する．

```
> jar cf ..¥queens2.jar queens
> cd ..
> java -classpath ←
  "queens2.jar;%PLCAFEDIR%¥plcafe.jar" ←
  jp.ac.kobe_u.cs.prolog.lang.PrologMain ←
  jp.ac.kobe_u.cs.prolog.builtin:cafeteria
```

パッケージ内の述語の呼出しは，以下のように述語名の前にコロンで釘ってパッケージ名を付ければ良い．

```
| ?- queens:main.
```

このように Prolog Cafe のパッケージ機能は，Java のパッケージ機能と親和性が高く，大規模なプログラムを開発する場合に非常に有効である．

5. お わ り に

本稿では，Prolog から Java へのトランスレータ処理系 Prolog Cafe について，その利用方法に重点をおいて述べた．Prolog Cafe の特徴は，開発言語である Java との親和性の高さにあるといえる．Prolog Cafe は GPL ライセンスに基づくオープンソースのソフトウェアとして公開されており，以下の URL から最新版を取得可能

である．

http://kaminari.istc.kobe-u.ac.jp/PrologCafe/

謝　　辞

Prolog Cafe 開発の一部については，平成 14 年度および平成 15 年度の情報処理振興事業協会 (IPA) 未踏ソフトウェア創造事業の支援を受けました．IPA の方々および紀プロジェクトマネージャに感謝いたします．

◇ 参　考　文　献 ◇

[Aït-Kaci 91] Aït-Kaci, H.: *Warren's Abstract Machine*, MIT Press (1991)

[Banbara 05] Banbara, M., Tamura, N., and Inoue, K.: Prolog Cafe: A Prolog to Java Translator System, in *Proceedings of the 16th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2005)*, pp. 45–54 (2005)

[Barbosa 02] Barbosa, J. L. V., Yamin, A. C., Augustin, I., Vargas, P. K., and Geyer, C. F. R.: Holoparadigm: a Multiparadigm Model Oriented to Development of Distributed Systems, in *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS 2002)*, p. 6 pages (2002)

[Cook 04] Cook, J. J.: P#: a concurrent Prolog for the .NET framework, *Software: Practice and Experience*, Vol. 34, pp. 815–845 (2004)

[Inoue 06] Inoue, K., Sasaura, Y., Soh, T., and Ueda, S.: A Competitive and Cooperative Approach to Propositional Satisfiability, *Discrete Applied Mathematics* (2006), to appear

[Kawamura 04] Kawamura, T., Kinoshita, S., and Sugahara, K.: Implementation of a Mobile Agent Framework on Java Environment, in Gonzalez, T. ed., *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, pp. 589–593 (2004), MIT, Cambridge, USA

[Soh 05] Soh, T., Inoue, K., Banbara, M., and Tamura, N.: Experimental results for solving job-shop scheduling problems with multiple SAT solvers, in *Proceedings of the 1st International Workshop on Distributed and Speculative Constraint Processing (held in conjunction with CP'05)*, pp. 25–38 (2005)

[Sterling 88] Sterling, L. and Shapiro, E.: Prolog の技芸：, 共立出版 (1988), 松田 利夫 訳

[Warren 83] Warren, D. H. D.: An abstract Prolog instruction set, Technical Report Technical Note 309, SRI International, Menlo Park, CA (1983)

[Wohlstadter 01] Wohlstadter, E., Jackson, S., and Devanbu, P. T.: Generating Wrappers for Command Line Programs: The Cal-Aggie Wrap-O-Matic Project, in *Proceedings of International Conference on Software Engineering*, pp. 243–252 (2001)

[後藤 84] 後藤 滋樹：Prolog 入門 – 知識情報処理の序曲, ソフトウェアライブラリ (1), サイエンス社 (1984)

[黒川 85] 黒川 利明：Prolog のソフトウェア作法, 岩波コンピュータサイエンス, 岩波書店 (1985)

[塚本 89] 塚本 龍男：わかる Prolog, 情報処理入門シリーズ, 共立出版 (1989)

〔担当委員：××○○〕

19YY 年 MM 月 DD 日　受理

——著　者　紹　介——

田村　直之

1980 年神戸大学理学部物理学科卒業．1985 年同大学院自然科学研究科博士課程システム科学専攻修了．学術博士．1985 年日本 IBM 東京基礎研究所入社．1988 年神戸大学工学部勤務．2003 年より神戸大学学術情報基盤センター学術情報処理研究部門教授．制約プログラミング，グリッド，線形論理，論理プログラミングなどに興味をもつ．日本ソフトウェア科学会，情報処理学会会員．

番原　睦則

1994 年神戸大学理学部数学科卒業．1996 年同大学大学院自然科学研究科博士課程前期数学専攻修了．1996 年国立奈良工業高等専門学校助手．1998 年同校講師．2003 年より神戸大学学術情報基盤センター学術情報処理研究部門講師．博士 (工学)．線形論理，論理プログラミング，制約プログラミングなどに興味をもつ．日本ソフトウェア科学会，情報処理学会会員．