PrologからJavaへのトランスレータ処理系とその応用

番原 睦則　田村 直之　井上 克巳

本論文では，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，WAM を介して，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 Java との連携，拡張性に優れた Prolog 処理系となっている．Prolog Cafe はマルチスレッドによる並列実行をサポートしており，スレッド間の通信は共有 Java オブジェクトにより実現される．また任意の Java オブジェクトを Prolog の項として取り扱う機能を有しており，Prolog からメソッド呼び出し，フィールドへのアクセスも行える．最後に Prolog Cafe の応用として，複数 SAT ソルバの並列実行システム Multisat について述べる．

We present the Prolog Cafe system that translates Prolog into Java via the WAM. Prolog Cafe has the advantages of portability, extensibility, and smooth interoperation with Java. Prolog Cafe provides multi-threaded Prolog engines. A Prolog Cafe thread seems to be conceptually an independent Prolog evaluator and communicates with each other through shared Java objects. From the Java side, the translated code of Prolog Cafe can be easily embedded into Java applications such as Applets and Servlets. From the Prolog side, any Java object can be represented as a Prolog term, and its methods and fields can be exploited from Prolog. We also give a brief introduction to the Multisat system, a parallel execution system of SAT solvers, as an application of Prolog Cafe.

1　はじめに

Prolog の実装に関しては，WAM 抽象機械 (Warren Abstract Machine) [1][21] が事実上の標準となっている．WAM は柔軟性が高く，高階プログラミング，並列プログラミング，制約プログラミング，線形論理プログラミングなど，数多くの拡張がなされている．また，Prolog から C へのコンパイラ処理系 [6] の設計にも応用されている．

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．

本論文の目的は，Prolog と Java の両言語を連携させた実用的な Prolog 処理系の開発することであり，そのためには以下の点が重要となる．

Java との連携： データ共有，Java から Prolog の利用，Prolog から Java の利用，マルチスレッド実行・同期，パッケージの利用，例外処理

拡張性： 言語拡張，機能拡張の容易性 (ユーザによる組込み述語の作成など)

実用性： 豊富な Java クラスライブラリを活用し

A Prolog to Java Translator System and its Application.

Mutsunori BANBARA and Naoyuki TAMURA, 神戸大学学術情報基盤センター, Information Science and Technology Center, Kobe University.

Katsumi INOUE, 国立情報学研究所, National Institute of Informatics.

コンピュータソフトウェア，Vol.XX，No.N（YYYY），pp.XX–XX.

[論文] xxxx 年 yy 月 zz 日受付.

たプログラム開発，実用に耐え得る実行速度

Prolog ユーザから見た場合，実用性 (特に，実行速度) だけを重視するのでれば，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，Java との連携，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 かつ実用性の高いアプリケーションを開発することは難しい．

本論文では，このような問題の 1 つの解決案として，Prolog から Java へのトランスレータ処理系 Prolog Cafe を提案する．本システムでは，Prolog プログラムは，WAM を介して，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．すなわち項，述語など Prolog の構成要素のすべてが Java に変換される．

Prolog Cafe の主な特徴を以下に挙げる：

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．

Java から Prolog の利用： Java から Prolog のプログラムを呼び出せる．Prolog のデータは Java のオブジェクトと，述語定義は Java のクラスと対応しており，Prolog 実行エンジンの生成，ゴールの実行 (ボックスモデル) が可能である．

Prolog から Java の利用： Prolog から Java プログラムを呼び出せる．Java の任意のオブジェクトを Prolog の項として扱え，Prolog から Java オブジェクトの生成，メソッド呼び出し，フィールドへのアクセスが可能である．

マルチスレッド： マルチスレッドによる並列実行が可能である．与えられたゴールに対して，新しい Prolog 実行エンジンを生成し，新しいスレッド上で独立的にゴールの実行をおこなう．各エンジン間のデータの受け渡しは，共有 Java オブジェクトを介して実現される．

パッケージ： 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．

例外処理： Prolog Cafe の例外処理は，従来のスタック操作による方法をベースに，Java の `try-catch` 節を用いて実装されているため，Prolog の例外処理に加え，Java で発生した例外も処理することができる．

拡張性： Prolog Cafe の (コンパイラ，組込み述語を除く) コア部分は約 50 キロバイトと小さく，また変換された Java プログラムも可読性が高いため，言語拡張，Java クラスライブラリを用いた機能拡張が容易である．

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 クラス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，十分実用に耐え得る．

応用例: Prolog Cafe は，1997 年頃から開発を開始したシステムであり，既にいくつかのアプリケーションに応用されている．以下に主な例を挙げる．

- P# [7]
  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 [12] [18]
  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 Cafe 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 [14]
  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 Cafe を拡張した言語処理

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 [3]
  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 [23]
  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 Cafe が使われている．

本論文では，第 2 節で，既存の Prolog から Java へのトランスレータ処理系 jProlog と LLPj の変換方法を紹介する．第 3 節では，Prolog Cafe 処理系の基本となる Prolog から Java への変換方法を説明した後，前述した Java との連携の中から特に重要と思われる，Java から Prolog の利用，Prolog から Java の利用，マルチスレッド実行に焦点を絞って述べる．また拡張性の一例として，ユーザによる組込み述語の作成方法について述べる．さらに計算機実験の結果，および Prolog Cafe の応用例として Multisat についても触れる．第 4 節で関連研究について述べた後，第 5 節でまとめを述べ，本論文を締めくくる．

2 既存のトランスレータ処理系

既存の Prolog から Java へのトランスレータ処理系 jProlog と LLPj について述べる．以下の例題を用いて，各処理系の変換方法を中心に説明を行う．

```
p :- q, r.
q.
```

この例題は，Prolog から C へのトランスレータ処理系に関する文献 [6] で使われており，Prolog の (決定的な場合の) 制御の流れが，どのように Java で実装されるかを見るのに適している．

2.1 jProlog

Demoen と Tarau による jProlog [8] は，最初に開発された Prolog から Java へのトランスレータ処理系である．BinProlog と同様に，継続渡しによるコンパイル手法・**バイナリ化** *(binarization)* [20] に基づいている．

各述語はクラスの集合に変換される．この集合は 1 つのエントリクラスと，節を表すクラスから構成される．各節はまずバイナリ節に変換され，その後 1 つのクラスに変換される．継続ゴールは述語ではなく項としてオブジェクトに変換される．この継続ゴールは，実行時にハッシュ表を参照し，対応する述語オブジェクトを探索または生成した後，実行される．

先に述べた例題は，まず以下のバイナリ節に変換される．ここで `Cont` は継続ゴールを表す．

```
p(Cont) :- q(r(Cont)).
q(Cont) :- call(Cont).
```

各バイナリ節は図 1 に示すクラスの集合に変換される．`Code` クラスは全ての述語のスーパークラスであり，`Exec` メソッドをもつ．変換された述語はエントリクラスの `Exec` メソッドを呼び出すことにより実行される．`Exec` メソッドの引数は `PrologMachine` クラスのオブジェクトである．このクラスは Prolog エンジン，すなわち実行時環境を実装したものであり，レジスタ，選択点スタック，トレイルスタックなどを保持している．継続ゴールは常に `Areg[0]` レジスタに項として格納される．Prolog のカット (!) は，従来の WAM ベースの Prolog 処理系と同様に実装されている．

変換された述語は，以下のようなループで実行される．

```
code = ゴールを表す述語オブジェクト;
while (ExceptionRaised == 0) {
    code = code.Exec(this) ;
}
```

上記コードのうち，`this` は `PrologMachine` クラスのオブジェクトであり，Prolog エンジンを表す．`Exec` メソッドは所定の処理を行った後，その継続ゴールを返す．このループは与えられたゴールに対して，バックトラックにより全ての試行が終わるまで繰り返し実行される．

jProlog は Java への変換について，基本的かつ重要なアイデアを含んでいる．また，バックトラック可能な破壊的更新，遅延評価などの拡張機能も備えてい

```
//述語 p/0 のエントリクラス
public class pred_p_0 extends Code {
    static Code cl1 = new pred_p_0_1();
    static Code q1cont;
    void init(PrologMachine mach) {
        //述語 q/0 のロード
        q1cont = mach.LoadPred("q",0);
    }
    Code Exec(PrologMachine mach) {
        //節 p(Cont) :- q(r(Cont)) の呼出
        return cl1.Exec(mach);
    }
}
//節 p(Cont) :- q(r(Cont)) を表すクラス
class pred_p_0_1 extends pred_p_0 {
    Code Exec(PrologMachine mach) {
        //継続ゴール Cont の取得
        PrologObject continuation = mach.Areg[0];
        //新しい継続ゴール r(Cont) の生成
        mach.Areg[0] =
            new Funct("r".intern(), continuation);
        mach.CUTB = mach.CurrentChoice;
        return q1cont; //述語 q/0 の呼出
    }
}
//述語 q/0 のエントリクラス
public class pred_q_0 extends Code {
    static Code cl1 = new pred_q_0_1();
    Code Exec(PrologMachine mach) {
        //節 q(Cont) :- call(Cont) の呼出
        return cl1.Exec(mach);
    }
}
//節 q(Cont) :- call(Cont) を表すクラス
class pred_q_0_1 extends pred_q_0 {
    Code Exec(PrologMachine mach) {
        mach.CUTB = mach.CurrentChoice;
        //継続ゴール Cont の呼出
        return UpperPrologMachine.Call1;
    }
}
```

図 1 jProlog のコード例

る．しかしながら，jProlog は実験的な処理系であり，改良できる点も数多く残されている．例えば，インデキシング，頭部単一化の最適化などが実装されておらず実行効率が悪い．また組込み述語も少なく，第 1 節に挙げた Java との連携機能も備えていない．そのため実用的な Prolog 処理系として利用することは困難である．

2.2 LLPj

LLPj [2] は LLP [24][26] から Java へのトランスレータ処理系である．LLP は線形論理に基づく論理型言語であり，Prolog の拡張言語となっている．

各述語はただ 1 つのクラスに変換され，各節はそのクラス中のメソッドに変換される．継続ゴールは項ではなく述語としてオブジェクトに変換され，ハッシュ表などを参照することなく直接実行される．このように LLPj の変換方法は jProlog と大きく異なる．

図 2 に例題の変換例を示す．`Predicate` クラスは全ての述語のスーパークラスであり，`exec` メソッド，`cont`，`trail` の 2 つのフィールドをもつ．変換された述語は，この `exec` メソッドを呼び出すことにより実行され，各節を表すメソッドを順番に呼び出す．2 つのフィールドは，継続ゴール，トレイルスタックを格納するために使われる．このように LLPj では，実行時に必要な情報が，全て述語を表すオブジェクト内に局所的に保持されており，jProlog の Prolog エンジンに相当するクラスは存在しない．Prolog のカット (!) は，Java の `try-catch` 節を用いて実装されている．

変換された述語は，以下のように実行される．

```
code = ゴールを表す述語オブジェクト;
code.exec();
```

LLPj の変換方法は非常にシンプルであり，結果として得られた処理系は，軽量かつ拡張性の高いものとなっている．また実行速度は jProlog とほぼ同等である．LLPj の短所は `exec` メソッドの実行が多重入れ子になる点である．すなわち `exec` メソッドは，節中の継続ゴールの `exec` メソッドを (`return` することなく) そのまま呼び出し，この繰り返しは，解が発見されるまで止まらない．このため，規模の大きな問題に対して，メモリオーバーフローを起こすという問題がある．

3 Prolog Cafe と Java との連携

3.1 Prolog 項の Java での表現方法

項は `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 3 参照)．`Term` クラスは全ての項のスーパークラスであり，抽

```
//述語 p/0 のエントリクラス
public class PRED_p_0 extends Predicate {
    public PRED_p_0 ( Predicate cont) {
        this.cont = cont; //継続ゴール Cont の取得
    }
    public void exec() {
        //節 p(Cont) :- q(r(Cont)) の呼出
        if(clause1()) return;
    }
    //節 p(Cont) :- q(r(Cont)) を表すメソッド
    private boolean clause1() {
        try {
            //新しい継続ゴール r(Cont) の生成
            Predicate new_cont =
                new PRED_r_0(cont);
             //述語 q/0 の呼出
            (new PRED_q_0(new_cont)).exec();
        } catch (CutException e) {
            if(e.id != this) throw e;
            return true;
        }
        return false;
    }
}
//述語 q/0 のエントリクラス
public class PRED_q_0 extends Predicate {
    public PRED_q_0 ( Predicate cont) {
        this.cont = cont; //継続ゴール Cont の取得
    }
    public void exec() {
        //節 q(Cont) :- call(Cont) の呼出
        if(clause1()) return;
    }
    //節 q(Cont) :- call(Cont) を表すメソッド
    private boolean clause1() {
        try {
            cont.exec(); //継続ゴール Cont の呼出
        } catch (CutException e) {
            if(e.id != this) throw e;
            return true;
        }
        return false;
    }
}
```

図 2　LLPj のコード例

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．

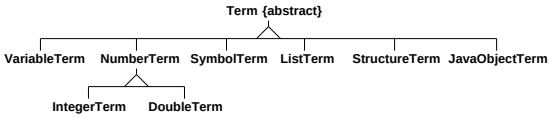

図 3　項のクラス図

3.2　Java への変換方法

jProlog の優れている点は，Prolog の項が Java のオブジェクトに，Prolog の述語が Java のクラス定義に対応する変換方法にある．実行速度だけを考えると，MINERVA のように，対応を付けせずに WAM のエミュレータを実行する方が現状では良いが，第 1 節で挙げた三つの開発ポイント，Java との連携，拡張性，実用性を実現するためには，jProlog のアプローチの方が適している．

Prolog Cafe では，jProlog の変換方法をベースに，LLPj の特徴を取り込んだ変換方法を採用している．jProlog および LLPj と比較して，以下の点が改善されている．

- インデキシング (WAM の swich_on_term に相当)，頭部単一化に関する最適化，算術式の最適化を行い，実行効率が改善されている．
- 継続ゴールは，LLPj 同様，述語を表すオブジェクトに変換される．これにより jProlog で生じるハッシュ表を参照するオーバーヘッドを避けることができる．
- 与えられたゴールは，jProlog と同様のループで処理されるため，LLPj で生じるメモリオーバーフローを回避できる．
- Prolog 実行エンジンを exec メソッドの引数として渡し，述語定義とは直交した概念とすることにより，マルチスレッド実行を可能とした．

jProlog 同様，各述語はクラスの集合に変換される．この集合は 1 つのエントリクラス，節を表すクラス，WAM の命令 (try，retry，trust など) を表すクラスから構成される．継続ゴールは項ではなく述語としてオブジェクトに変換され，直接的に実行される．

図 4 に例題の変換例を示す．Prolog Cafe では，述語 p/n は，Java のクラス PRED_p_n に変換される．

```
import jp.ac.kobe_u.cs.prolog.lang.*;
//述語 p/0 のエントリクラス
public class PRED_p_0 extends Predicate {
    public PRED_p_0(Predicate cont) {
        this.cont = cont; //継続ゴール Cont の取得
    }
    //節 p(Cont) :- q(r(Cont)) の呼出
    public Predicate exec(Prolog engine) {
        engine.setB0();
        //新しい継続ゴール r(Cont) の生成
        Predicate p1 = new PRED_r_0(cont);
        return new PRED_q_0(p1); //述語 q/0 の呼出
    }
}
//述語 q/0 のエントリクラス
public class PRED_q_0 extends Predicate {
    public PRED_q_0(Predicate cont) {
        this.cont = cont; //継続ゴール Cont の取得
    }
    //節 q(Cont) :- call(Cont) の呼出
    public Predicate exec(Prolog engine) {
        return cont; //継続ゴール Cont の呼出
    }
}
```

図 4 Prolog Cafe のコード例

3.3 Prolog から Java の利用

Prolog Cafe では，任意の Java オブジェクトを Prolog の項として取り扱うことができる．つまり Prolog からオブジェクトの生成，メソッド呼び出し，フィールドへのアクセスが可能である．この際に問題となるのが，Prolog は項が不可変 (immutable) であることを前提とした言語であるのに対し，Java のオブジェクトは，一般に状態をもつ可変 (mutable) なデータ構造であり，自然な融合は難しい．

Prolog Cafe は，Prolog から Java の機能をフル活用できることを目的として開発された言語処理系であり，メソッド呼び出し，フィールド値の変更によるオブジェクトの状態変化に対して，何ら制限を行っていない．そのため，Prolog から Java オブジェクトを扱う場合は，宣言的性質は失われ，手続き的な言語として用いることになる．これは `assert`，`retract` 同様，副作用を伴う．

Prolog Cafe では，以下の組込み述語を用いて，オブジェクトの生成，メソッド呼び出し，フィールドへのアクセスを行う．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 の項 (Java オブジェクト以外) の場合，対応する Java オブジェクトに自動的に変換される．現在のデータ変換では，アトムは `String`，整数は `Integer` または `BigInteger`，浮動小数点は `Double`，リストは `Vector` に変換される．

たとえば，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，`setVisible` メソッドにより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), _).
```

上記の組込み述語は，Java のリフレクション機能を使って実装されている．これらは，public 宣言されたコンストラクタ，メソッド，フィールドのみを対象としているが，次に示す組込み述語：

- `java_declared_constructor`(C, O)
- `java_declared_method`(O, M, R)
- `java_get_declared_field`(O, F, V)
- `java_set_declared_field`(O, F, V)

を用いれば，public 以外のモディファイヤが指定されたものも処理可能である．またコンストラクタ，メ

ソッド呼び出しのパラメータに関しては，ワイドニング変換に対応している．

3.4 Java から Prolog の利用

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
3. 述語の引数の配列 $args$ 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 $args$ 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．

以下に，Java プログラムから，Prolog のゴール father(abraham, X) を実行するコード例を示す．

```
import jp.ac.kobe_u.cs.prolog.lang.*;
PrologControl e = new PrologControl();
Predicate p = new PRED_father_2();
Term a1 = SymbolTerm.makeSymbol("abraham");
Term a2 = new VariableTerm();
Term[] args = {a1, a2};
e.setPredicate(p, args);
for (boolean r = e.call(); r; r = e.redo()) {
    System.out.println(a2.toString());
}
```

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．

3.5 ユーザによる組込み述語の作成方法

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/1 を作成する方法を述べる．

まず“msgbox(_).”という一行だけから成るファイル msgbox.pl を作成し，これをトランスレータで変換すると，PRED_msgbox_1.java が生成される．図 5 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/1 を実現したものである．

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．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，組込述語の追加を簡単に行うことができる．

3.6 マルチスレッドによる並列実行

Prolog Cafe では，Prolog の実行エンジンと Java のスレッドは一対一に対応している．Prolog の実行エンジン PrologControl オブジェクトは，複数生成することができ，逐次実行用のメソッド call，redo に加え，並列実行用のメソッド start，stop，join 等を有している．そのため，複数の実行エンジンを生成し，それぞれで Prolog のプログラム (ゴール) を実行できる．各エンジン間でのデータの受け渡し (実行結果の取り出し，通信など) は，予めゴールの引数に共有 Java オブジェクトを与え，そのオブジェクトを介して行う．

以下に，N-Queens 問題を解く Prolog プログラムを，二つの Prolog 実行エンジンを用いて並列実行する Java プログラムの例を示す．

```
import jp.ac.kobe_u.cs.prolog.lang.*;
PrologControl e1 = new PrologControl();
PrologControl e2 = new PrologControl();
Term a1[] = {new IntegerTerm(4), new VariableTerm()};
Term a2[] = {new IntegerTerm(8), new VariableTerm()};
e1.setPredicate(new PRED_queens_2(), a1);
e2.setPredicate(new PRED_queens_2(), a2);
e1.start();
e2.start();
```

さらに第 3.3 節で述べた組込み述語を使うと，Pro-

```
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 5 組込述語の例 (PRED_msgbox_1.java)

log プログラムから，Prolog 実行エンジンを複数生成し，並列実行を行える．図 6 は，並列実行のための制御プログラム例である．Prolog 実行エンジンの生成およびゴールの実行は，start/2 によって行われる．start(Goal, Engine) は，まず PrologControl オブジェクトを生成し，Engine にバインドする．次にゴールのコピーを作成し，Engine にセットする．最後に start メソッドを呼び出し，ゴールの実行を開始する．stop/1 は実行エンジンを停止する．join/1 は実行エンジンに割り当てられたゴールが成功または失敗するまで待機する．sleep/1 は指定した時間だけエンジン (スレッド) をブロック状態にする述語である．

述語 start/2 は，SICStus MT [9] の spawn/2,

```
start(Goal, Engine) :-
    java_constructor('PrologControl', Engine),
    copy_term(Goal, G0),
    java_wrap_term(G0, G),
    java_method(Engine, setPredicate(G), _),
    java_method(Engine, start, _).
stop(Engine) :- java_method(Engine, stop, _).
join(Engine) :- java_method(Engine, join, _).
sleep(Wait)  :-
  java_method('java.lang.Thread', sleep(Wait), _).
in_success(Engine) :-
  java_get_field('java.lang.Boolean', 'TRUE', T),
  java_method(Engine, in_success, T).
in_failure(Engine) :-
  java_get_field('java.lang.Boolean', 'TRUE', T),
  java_method(Engine, in_failure, T).
ready(Engine) :-
  java_get_field('java.lang.Boolean', 'TRUE', T),
  java_method(Engine, ready, T).
result(Engine, Result) :-
  java_method(Engine, next, Result).
cont(Engine) :- java_method(Engine, cont, _).
```

図 6 並列実行制御プログラムの例

Ciao Prolog [5] の launch_goal/2, Jinni [19] の new_engine/3 とほぼ同等であり，呼び出し元と共有変数をもたないゴールを，新規に作成された実行環境 (すなわち，スタック等が新規作成された環境) において，新規スレッドで実行する．述語 start/2 自体は，実行したゴールの結果 (成功・失敗) にかかわらず直ちに成功する．ゴールの実行結果 (成功・失敗) は，in_success/1，in_failure/1 で確認することができ，cont/1 を実行すると，バックトラックし別解の探索を再開する．

Prolog Cafe には，現在のところ，呼び出し元から実行結果 (Prolog の項) を取り出す組込み述語は用意されておらず，各エンジン間でのデータの受け渡しは，予めゴールの引数に共有 Java オブジェクトを与え，そのオブジェクトを介して行う．また，この共有 Java オブジェクトのロック (同期) には，組込み述語 synchronized/2 を用いる．synchronized(*Object*, *Goal*) を実行すると，*Object* はロックされ，*Goal* の実行が完了するとロックが解除される．この述語は，継続ゴールの exec メソッドを呼び出すループ処理を，局所的に Java の synchronized ブロック内で行うことにより実現されている．

本節で述べたマルチスレッドによる Prolog の並列実行は，プログラマが明示的に並列性を記述するスタイルである．このような並列実行方式を採用した理由としては，Prolog Cafe では，Prolog の実行エンジンと Java のスレッドが一対一に自然に対応しており，第 3.3 節で述べた Prolog から Java を呼び出す組込み述語を用いて，Prolog プログラム中で自分自身 (Prolog 実行エンジン) を複数生成し，Prolog プログラムだけでなく，第 3.5 節で述べた方法でユーザ自身が作成した (外部プログラムを呼び出す可能性のある) 組込み述語を，並列実行できる点が挙げられる．

3.7 計算機実験

標準的な Prolog ベンチマークを用いて，Prolog Cafe の性能評価を行った．比較対象には，Prolog Cafe と同じ Prolog から Java へのトランスレータである jProlog と，知名度の高い WAM ベースの Prolog コンパイラ処理系 SWI-Prolog を用いた．表 1 に実行結果を示す．時間は全て Linux システム (Xeon 2.8GHz, 2G メモリ) 上で測定し，単位はミリ秒である．Java 処理系には JDK 1.5.0 を用いた．各実行時間の右側には，括弧書きで Prolog Cafe の実行時間を 1.0 とした場合の実行時間比を記している．1.0 より大きいほど低速で，小さいほど高速であることを示す．表中の “?” は，jProlog が生成した Java プログラムをコンパイルする際に，エラーが発生したことを意味する．原因の大部分は組込み述語不足であった．

jProlog と比較した場合，実行時間比は 1.10〜6.88 と幅があるが，Prolog Cafe は幾何平均で 2 倍早い．SWI-Prolog と比較した場合，Prolog Cafe は `tak` と `queens_12` で各々9.43 倍，1.48 倍速い[†1]．実行時間比の幾何平均では，約 2.8 倍程度遅いが，十分実用に耐え得る．また Prolog Cafe の効率面に関しては，レジスタ配置，末尾再帰，大域的プログラム解析などの最適化が未実装であることを考慮すれば，少なからず高速化の可能性は残っている．

†1 SWI-Prolog の最適化オプション `-O` を用いると，`tak` の実行時間は 47 ミリ秒，`queens_12` は 10500 ミリ秒となり，Prolog Cafe と比較して，SWI-Prolog は各々 5.5 倍，1.2 倍程度速い．

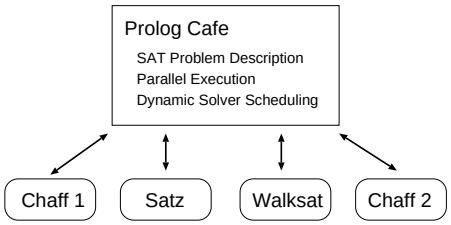

図 7 Multisat の構成例

3.8 Multisat: 複数 SAT ソルバの並列実行システム

第 3.6 節で述べた Prolog Cafe の並列実行方式は，複数異種のシステムを組み合わせたハイブリッド型システムの開発などに有用である．なぜならば，各サブシステムをラップする組込み述語を作成し，その上で，それらのサブシステムを制御するプログラムを，Prolog プログラムとして実現できる．そのため，様々な制御アルゴリズムの実装，実行実験が容易に行える．この節では Prolog Cafe の並列実行方式の応用例として，Multisat [12][18] を紹介する．

Multisat は，複数異種の SAT ソルバを競争的・協調的に並列動作させ，解探索を行うシステムである．Multisat は SAT ソルバとして，系統的ソルバである Satz [15] と Chaff [16]，確率的ソルバである Walksat [4] を利用することができる．

各 SAT ソルバは全て Java で実装されており，Prolog Cafe を用いて，Prolog プログラムから，各 SAT ソルバの生成・並列実行・停止を行うことができる．図 7 に，Satz，Walksat，2 つの Chaff を生成した場合の，Multisat の構成図を示す．

Multisat の特徴は以下の通りである．

- 競争的モード：
  複数異種の SAT ソルバが，競争的に並列動作する．各ソルバは独立に実行され，他ソルバとの相互作用 (解情報の交換などは行わない．そのため個々のソルバの短所を補い，より多くの種類の問題を効率よく判定することができる．
- 協調的モード：
  系統的ソルバである Chaff が実行時に生成するの補題を利用して，協調的な解探索を行う．補題は解探索の過程において，真偽値割り当てに矛盾が

表 1 ベンチマーク結果

Prolog プログラム	実行回数	Prolog Cafe 0.9.3	jProlog 0.1	SWI-Prolog 5.0
boyer	10	2054.1 (1.00)	2699.0 (1.31)	200.0 (0.10)
browse	10	436.4 (1.00)	3003.3 (6.88)	222.0 (0.51)
ham	10	490.1 (1.00)	709.6 (1.45)	125.0 (0.26)
nrev (300 要素)	10	22.9 (1.00)	106.3 (4.64)	11.0 (0.48)
tak	10	261.7 (1.00)	301.7 (1.15)	2469.0 (9.43)
zebra	10	55.1 (1.00)	60.8 (1.10)	3.0 (0.05)
cal	10	226.5 (1.00)	?	67.0 (0.30)
poly_10	10	170.0 (1.00)	?	11.0 (0.06)
queens_12 (全解探索)	10	13339.6 (1.00)	?	19776.0 (1.48)
sendmore	10	71.7 (1.00)	?	28.0 (0.39)
実行時間比の幾何平均		1.00	2.06	0.36

生じたときに生成される．この補題を Satz が利用することにより，同じ矛盾を回避することができる．

- 複数ソルバの簡易スケジューリング：
  各 SAT ソルバは，独立したスレッド上で並列実行されるため，Java スレッドの優先度を操作することにより，実行時に各ソルバの優先度を動的に変更するこができる．

Multisat は単体ソルバと比較して，代表的な SAT ベンチマーク SATLIB [10] の問題を平均して効率良く解くことができる．また SAT プランニング [13]，ジョブショップスケジューリング問題 [17] にも有効であることが示されている．なぜならば，Multisat は系統的ソルバと確率的ソルバの両方の長所を備えており，結果として充足可能・充足不可能を両方含むような SAT 問題群を解くのに適しているためである．これらの問題に対する Multisat の有効性を示す具体的なデータは，文献 [12] に記載されている．

Prolog Cafe の応用例として Multisat を紹介したが，Perl 等のスクリプト言語を用いて，Multisat の並列動作を実現することは可能である．しかしながら，Multisat のような複数のシステムを組み合わせたハイブリッド型システムの開発過程においては，制御アルゴリズムの設計，プロトタイプ作成，実行実験のサイクルが頻繁に生じ，このようなサイクルを伴うシステムの開発には，プロトタイピングが容易な Prolog の方が適している．

4 関連研究

本論文で述べた jProlog，LLPj，Prolog Cafe 以外にも，Java による Prolog 処理系が数多く提案されている：MINERVA, Jinni, W-Prolog, tuProlog, PROLOG+CG, JIProlog, KLIJava, JavaLog, DGKS Prolog, JLog, and XProlog.

MINERVA [11] と Jinni [19] は商用の Prolog コンパイラ処理系であり，Prolog を独自の抽象機械にコンパイルし，Java により実行する．ただし MINERVA は逐次的な処理系であり，マルチスレッドによる並列実行はサポートしていない．W-Prolog [22] およびその他のシステムは，インタプリタとして実装されている．インタプリタ処理系はシンプルであるが，実行効率の面に問題がある．

5 まとめ

本論文では，Prolog から Java へのトランスレータ処理系 Prolog Cafe について，その変換方法，Java との連携，ユーザによる組込み述語の作成，マルチスレッドによる並列実行を中心に述べた．また効率面でも，標準的な Prolog ベンチマークに対して，Prolog Cafe は既存のトランスレータ処理系 jProlog より，約 2 倍早く，WAM ベースの処理系 SWI-Prolog と比較した場合，その速度低下は 2.8 倍程度に抑えられており，十分に実用に耐え得る．さらに Prolog Cafe の応用例として，Multisat についても説明を行った．以上のことから，Prolog Cafe は Prolog と Java の両言語を連携させた実用的な Prolog 処理系であると言える．

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であり，最新版は以下の URL から取得可能である．

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

本論文で述べた Prolog Cafe の機能および Multisat は，HECS プロジェクト [25] において開発されたものである．HECS とは複数異種の制約ソルバを協調的・競争的に並列動作させることにより，効率の良い解探索を行う制約解消システムである．制約ソルバとしては，整数制約ソルバ，ブール値制約ソルバ，実数制約ソルバが利用できる．HECS は Multisat の機能の他に，OpenOffice Calc を用いた使いやすいインタフェイス，整数有限領域上での最適解探索，確率的アルゴリズム (SA，TS など) を用いた準最適解探索，実数領域上での線形な制約条件に対する最適解探索などが可能である．

謝辞

本研究は情報処理振興事業協会 (IPA) による平成 15 年度未踏ソフトウェア創造事業 紀 PM 採択プロジェクト「Java による分散協調制約解消システム」[25] の一部である．研究にあたり御助言を承りました神戸大学の玉置久教授，鳥取大学の川村尚生助教授に感謝致します．

参 考 文 献

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

[2] Banbara, M. and Tamura, N.: Java implementation of a linear logic programming language, *Proceedings of the 10th Exhibition and Symposium on Industrial Applications of Prolog*, October 1997, pp. 56–63.

[3] 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, *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS 2002)*, December 2002, pp. 6 pages.

[4] B.Selman, H.Kautz, and B.Cohen: Local Search Strategies for Satisfiability Testing, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*(D.S.Johnson and M, A.(eds.)), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, American Mathematical Society, 1996, pp. 521–531.

[5] Carro, M. and Hermenegildo, M. V.: Concurrency in Prolog Using Threads and a Shared Database, *Proceedings of the 15th International Conference on Logic Programming (ICLP'99)*(Schreye, D. D.(ed.)), November 1999, pp. 320–334.

[6] Codognet, P. and Diaz, D.: `WAMCC`: Compiling Prolog to C, *Proceedings of International Conference on Logic Programming*(Sterling, L.(ed.)), The MIT Press, Jun 1995, pp. 317–331.

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

[8] Demoen, B. and Tarau, P.: jProlog Home Page, `http://www.cs.kuleuven.ac.be/~bmd/PrologInJava/`, 1996.

[9] Eskilson, J. and Carlsson, M.: SICStus MT – Multithreaded Execution Environment for SICStus Prolog, *Proceedings of the JICSLP'98 Post Conference Workshop 7 on Implementation Technologies for Programming Languages based on Logic*(Sagonas, K.(ed.)), June 1998, pp. 59–71.

[10] Hoos, H. H. and Stutzle, T.: SATLIB: An Online Resource for Research on SAT, *SAT 2000*(I.P.Gent, H.v.Maaren, T.(ed.)), IOS Press, 2000, pp. 283–292.

[11] IF Computer: MINERVA Home Page, `http://www.ifcomputer.com/MINERVA/`, 1996.

[12] Inoue, K., Sasaura, Y., Soh, T., and Ueda, S.: A Competitive and Cooperative Approach to Propositional Satisfiability, *Discrete Applied Mathematics*, Vol. 154(2006), pp. 2291–2306.

[13] Kautz, H. and Selman, B.: Unifying SAT-based and graph-based planning, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999, pp. 318–325.

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

[15] Li, C. M. and Anbulagan: Heuristics Based on Unit Propagation for Satisfyability Problems, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997)*, 1997, pp. 366–371.

[16] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S.: Chaff: Engineering an Efficient SAT Solver, *Proceedings of the 38th Design Automation Conference (DAC 2001)*, ACM, 2001, pp. 530–535.

[17] M.R.Garey, D.S.Johnson, and R.Sethi: The Complexity of Flowshop and Jobshop Scheduling, *Mathematics Operation Research*, Vol. 1(1976), pp. 117–129.

[18] Soh, T., Inoue, K., Banbara, M., and Tamura, N.: Experimental results for solving job-shop

scheduling problems with multiple SAT solvers, *Proceedings of the 1st International Workshop on Distributed and Speculative Constraint Processing*, October 2005.

[19] Tarau, P.: Jinni: a Lightweight Java-based Logic Engine for Internet Programming, *Proceedings of JICSLP'98 Implementation of LP languages Workshop*(Sagonas, K.(ed.)), June 1998. invited talk.

[20] Tarau, P. and Boyer, M.: Elementary Logic Programs, *Proceedings of Programming Language Implementation and Logic Programming*, Lecture Notes in Computer Science, No. 456, Springer, August 1990, pp. 159–173.

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

[22] Winikoff, M.: W-Prolog Home Page, `http://goanna.cs.rmit.edu.au/~winikoff/wp/`, 1996.

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

[24] 番原睦則, 姜京順, 田村直之: 線形論理型言語のコンパイラ処理系のための抽象機械について, 日本ソフトウェア科学会論文誌 コンピュータソフトウェア, Vol. 18,No. 1(2001), pp. 39–60.

[25] 番原睦則, 田村直之, 井上克巳, 川村尚生, 玉置久: Java による分散協調制約解消システム, 2004. 平成 15 年度 IPA 未踏ソフトウェア創造事業成果報告書.

[26] 姜京順, 番原睦則, 田村直之: 線形論理型言語の効率的なリソース管理モデル, 日本ソフトウェア科学会論文誌 コンピュータソフトウェア, Vol. 18,No. 0(2001), pp. 138–154.