PrologからJavaへのトランスレータ処理系とその応用

A Prolog to Java Translator System and its Application

番原睦則 † 田村直之 † 井上克巳 ††
Mutsunori BANBARA Naoyuki TAMURA Katsumi INOUE

† 神戸大学学術情報基盤センター
Information Science and Technology Center, Kobe University
{banbara,tamura}@kobe-u.ac.jp
†† 国立情報学研究所
National Institute of Informatics
ki@nii.ac.jp

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 は移植性，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．実行速度については，既存の Prolog から Java へのトランスレータ処理系 jProlog と比較して，2 倍以上の高速化を実現している．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．

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 SICSus 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：

- 移植性
  100%純 Java で実装されており，Java が動作する環境であれば，基本的に利用可能である．
- 拡張性
  変換された Java プログラムは可読性が高く，Java クラスライブラリを用いて，簡単に機能拡張を行うことができる．
- 利便性 (Java との連携)
  Java ユーザは，変換された Java プログラムを，アプレット，サーブレットなどへに簡単に埋め込むことができる．Prolog ユーザは，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，メソッド呼び出し，フィールドへのアクセスを Prolog から行える．また 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．

- 並列性
  マルチスレッドによる並列実行が可能である．各スレッドは，与えられたゴールに対して，局所的な実行環境を生成し，独立的にゴールの実行をおこなう．またスレッド間の通信は，共有 Java オブジェクトを介して実現される．
- モジュール性
  Prolog のモジュールは，Java のパッケージに変換される．そのためモジュール単位での開発，Java との連携が可能である．

実行効率に関しては，既存の Prolog から Java へのトランスレータ処理系 jProlog と比較した結果，標準的な Prolog ベンチマークに対して，平均で約 2.7 倍速く，効率性も良い．旧バージョンの Prolog Cafe [25] との大きな違いは，Java との連携，マルチスレッドによる並列実行，モジュールシステム，例外処理の実装方法である．

Prolog Cafe は既にいくつかのアプリケーションに応用されており，その有用性が示されている：P# [7], Multisat [13], Maglog [15], Holoparadigm [3], CA-WOM [23]．これらのアプリケーションのうち，本論文では複数 SAT ソルバの並列実行システム Multisat について述べる．

以下に本論文の構成を示す．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 の変換方法を紹介した後，第 3 節で，Prolog Cafe の変換方法，Java との連携，マルチスレッドによる並列実行，計算機実験の結果について述べる．また Prolog Cafe の応用例として Multisat を紹介する．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

```
//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) {
        q1cont = mach.LoadPred("q",0); //述語 q/0 のロード
    }
    Code Exec(PrologMachine mach) {
        //節 p(Cont) :- q(r(Cont)) の呼び出し
        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 の呼び出し
    }
}
//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) の呼び出し
        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

```
//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)) の呼び出し
        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 の呼び出し
            (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) の呼び出し
        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 の呼び出し
        } catch (CutException e) {
            if(e.id != this) throw e;
            return true;
        }
        return false;
    }
}
```

図 2: LLPj のコード例

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．またマルチスレッドによる並列実行，モジュール，Java との連携などの機能も備えていない．

2.2 LLPj

LLPj [2] は LLP [26, 28] 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

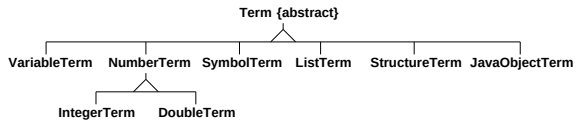

図 3: 項のクラス図

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．

```
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)) の呼び出し
    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 の呼び出し
    }
}
//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) の呼び出し
    public Predicate exec(Prolog engine) {
        return cont; //継続ゴール Cont の呼び出し
    }
}
```

図 4: Prolog Cafe のコード例

3 Prolog Cafe

この節では，Prolog Cafe の変換方法，Java との連携，マルチスレッドによる並列実行，計算機実験の結果について述べる．また Prolog Cafe の応用例として，Multisat を紹介する．

3.1 Prolog から Java への変換方法

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．

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．また，インデキシング (WAM の swich_on_term)，頭部単一化に関する最適化，算術式の最適化を行っており，実行効率に関しても改良されている．

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，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．この例題は述語 1 つに対して節 1 つの決定的な場合であるため，各述語はだた 1 つのクラスに変換される．これは jProlog が各述語に対して生成する 2 つのクラスを，1 つにまとめた形になっている．ただし，非決定的な場合はこの限りではなく，エントリクラスの exec メソッド中には，WAM の try 命令，あるいは swich_on_term 命令に対応するコードが生成される．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も回避できる．

旧バージョンの Prolog Cafe [25] との大きな違いは，Java との連携，マルチスレッドによる並列実行，モジュールシステム，例外処理の実装方法である．モジュールシステムは，Java のパッケージ機能を用いて実現されている．また例外処理は，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 の例外だけでなく，Java で発生した例外も処理することが可能となっている．本論文ではこれらの新しい機能のうち，Java との連携，マルチスレッドによる並列実行について詳しく述べる．

3.2 Java との連携

Prolog Cafe のスレッドは，PrologControl クラスによって実現される．このクラスは Prolog エンジン，ゴールを格納するフィールドを有し，スレッドに対する様々なメソッド，call，redo 等を提供す

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．

```
PrologControl p = new PrologControl();
Predicate code = new PRED_father_2();
Term a1 = SymbolTerm.makeSymbol("abraham");
Term a2 = new VariableTerm();
Term[] args = a1, a2;
p.setPredicate(code, args);
for (boolean r = p.call(); r; r = p.redo()) {
    System.out.println(a2.toString());
}
```

一方，Prolog Cafe では，任意の Java オブジェクトを Prolog の項 (以降，*Java* **項**と呼ぶ) として取り扱うことができる．Java 項は Java オブジェクトを JavaObjectTerm クラスでラップすることにより実装される．Java 項の生成，メソッド呼び出し，フィールドへのアクセスは，以下の組込み述語を通じて行う．

- java_constructor(+Class, ?Object)
- java_method(+Object, +Method, ?Return)
- java_get_field(+Object, +Field, ?Value)
- java_set_field(+Object, +Field, +Value)

Java 項の生成には，java_constructor/2 を用いる．この述語の第 1 引数はクラス名とパラメータを表すアトムまたは複合項 (リストを除く) である．アトムの場合は，デフォルトコンストラクタ (パラメータなしのコンストラクタ) を呼び出す．例えば，java_constructor($f(a_1,\ldots,a_n)$, O) を実行すると，クラス名 f，パラメータ $a'_1,\ldots,a'_n$ のオブジェクトを表す Java 項を生成し O と単一化する．各パラメータ a'_i は，Prolog の項 a_i を Java のオブジェクトに変換したものである．現在のデータ変換は，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，Java 項は保持しているオブジェクトに変換される．

メソッド呼び出しには，java_method/3 を用いる．第 1 引数は Java 項あるいはクラス名を表すアトム．第 2 引数はメソッド名とパラメータを表すアトムまたは複合項 (リストを除く) である．第 1 引数がアトムであることは，第 2 引数がクラスメソッドであることを意味する．例えば，java_method(O, $g(b_1,\ldots,b_m)$, R) を実行すると，O が表すオブジェクト (又はクラス) のメソッド $g(b'_1,\ldots,b'_m)$ を呼び出し，返り値を表す Java 項を R と単一化する．各パラメータ b'_j は，Prolog の項 b_j を Java オブジェクトに変換したものである．データ変換は先程の java_constructor/2 の場合と同様である．フィールドへのアクセスに関しても，java_get_field/3 と java_set_field/3 によって同様に行えるが，ここでは説明を省略する．

以下に簡単な例を挙げる．

```
main :-
  java_constructor('java.awt.Frame', X),
  java_method(X, setSize(400,300), _),
  java_get_field('java.lang.Boolean', 'TRUE', True),
  java_method(X, setVisible(True), _).
```

main を実行すると，java.awt.Frame クラスを表す Java 項が生成された後，setSize メソッドによりサイズが指定される．次に，java.lang.Boolean クラスの TRUE フィールドの値を表す Java 項が生成され，最後に，その Java 項を引数として setVisible メソッドが呼び出される．この結果，空のフレームが画面上に現れる．

この節で述べた組込み述語は，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/2
- java_declared_method/3
- java_get_declared_field/3
- java_set_declared_field/3

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．

3.3 マルチスレッドによる並列実行

旧 Prolog Cafe は，その変換方法上の問題からスレッドセーフが実現されておらず，マルチスレッドによる並列実行が不可能であった．旧 Prolog Cafe では，全ての述語のスーパークラス Predicate クラスに，Prolog エンジンを格納するフィールドを設けていた．しかし，実行時に述語オブジェクトを再利用する最適化を行った場合，正しい Prolog エンジンが選択されず，予期しない結果が発生した．この問題を解決するために，新しい Prolog Cafe では，Prolog エンジンは exec メソッドの引数として渡され，述語とは直交した概念となっている．

Prolog Cafe のスレッドは，1 つの独立した Prolog 実行環境と考えることができる．各スレッドでは，与えられたゴールに対して，局所的な実行環境を生成し，独立的にゴールの実行をおこなう．先に述べたように，Prolog Cafe スレッドは，PrologControl クラスによって実現される．このクラスは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．

以下に，N-Queens 問題を解く Prolog プログラムを，Prolog Cafe スレッドを用いて並列実行する Java プログラムの例を示す．

```
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();
```

第 3.2 節で述べた Java との連携機能を使うと，Prolog プログラムから，Prolog Cafe スレッドを用いた並列実行が可能である．

```
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), _).
```

スレッドの生成は，start/2 によって行われる．この述語は，SICStus MT [9] の spawn/2, Ciao Prolog [5] の launch_goal/2, Jinni [19] の new_engine/3 とほぼ同等である．start(Goal, Engine) は，まず PrologControl オブジェクトを表す Java 項を生成し，Engine に単一化する．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/2 はスレッドの実行を停止する．join/2 はスレッドに割り当てられたゴールが成功または失敗するまで待つ．sleep/1 は指定した時間だけスレッドをブロック状態にする．

Prolog Cafe では，独立したスレッド上で動作する各々のゴールは，共有 Java 項，すなわちオブジェクトを介して通信を行う．また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．

論理プログラムの並列実行に関しては，研究初期から現在に至まで，数多くの研究成果が発表されている．近年のサーベイとしては，文献 [10] がある．本節で述べたマルチスレッドによる並列実行は，プログラマが明示的に並列性を記述するスタイルであり，最近では Jinni，Ciao Prolog などのでも採用されている．

3.4 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．比較には，Prolog Cafe と同じ Prolog から Java へのトランスレータである jProlog 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 は平均で 2.75 倍高速なコードを生成する．この高速化はインデキシング，頭部単一化の最適化，算術式の最適化などを実装した効果である．

Prolog 処理系の実装に関しては，WAM 抽象機械に基づくコンパイラ処理系が主流である．よって参考のため知名度の高い WAM ベースの Prolog コンパイラ処理系 SWI-Prolog の実行時間を表 1 に記載している．SWI-Prolog と比較した場合，Prolog Cafe は tak と queens_12 に対して各々9.43 倍，1.48 倍速い[1]．しかし，tak を除いた平均では，約 2.5 倍程度遅くなっている．

SWI-Prolog と比べると，Java で実装した Prolog Cafe は効率面で劣っている，しかしながら，Prolog Cafe は効率性を最重要視した処理系ではなく，第 1 節で述べた特徴をもつ，Java と強く結び付いた Prolog 処理系を目指して設計・実装されている．しかし，効率性についても，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．

3.5 Multisat: 複数 SAT ソルバの並列実行システム

Multisat [24] は，Prolog Cafe スレッド上で，複数異

[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．

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)
実行時間比の平均		1.00	2.75	1.31
実行時間比の平均 (tak なし)		1.00	2.75	0.40

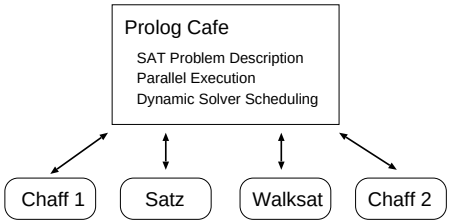

図 5: Multisat の構成例

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 [16] と Chaff [17]，確率的ソルバである Walksat [4] を利用することができる．

Multisat および各 SAT ソルバは全て Java で実装されており，Prolog Cafe 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 5 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，Prolog Cafe スレッド上で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，スレッドの優先度機能（setPriority メソッド）を用いて，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 [11] 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 [14]，ジョブショップスケジューリング問題 [18] にも有効であることが示されている [13]．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．

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 [12] 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．文献 [7] によると，MINERVA と Jinni は標準的な Prolog ベンチマークに対して，旧バージョンの Prolog Cafe より約 2 倍速いことが報告されている．しかし，これらの処理

系では，単独で実行可能なクラスファイル (.class) を生成することはできない．さらに 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，実行効率に問題がある．このような理由から，Prolog Cafe は，Prolog から Java へのトランスレート方式を採用している．

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，マルチスレッドによる並列実行を中心に述べた．また Prolog Cafe の応用例として，Multisat についても説明を行った．計算機実験の結果，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 と比較して，約 2.7 倍の高速化を実現している．また WAM ベースの処理系 SWI-Prolog と比較しても，速度低下は 2.5 倍程度に抑えられている．以上のことから，Prolog Cafe は開発言語である Java の機能を十分に活用でき，マルチエージェントシステム等，ある種の知的処理を伴う Java アプリケーションの開発に適したツールであると言える．

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 の新機能および Multisat は，HECS プロジェクト [27] 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」[27] 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助教授，鳥取大学の川村尚生助教授に感謝致します．

参考文献

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

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

[3] Jorge Luis Victória Barbosa, Adenauer Corrêa Yamin, Iara Augustin, Patrícia Kayser Vargas, and Cláudio Fernando Resin Geyer. 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, December 2002.

[4] B.Selman, H.Kautz, and B.Cohen. Local search strategies for satisfiability testing. In D.S.Johnson and .A.Trick M, editors, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, Vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 521–531. American Mathematical Society, 1996.

[5] Manuel Carro and Manuel V. Hermenegildo. Concurrency in prolog using threads and a shared database. In Danny De Schreye, editor, *Proceedings of the 15th International Conference on Logic Programming (ICLP'99)*, pp. 320–334, November 1999.

[6] Philippe Codognet and Daniel Diaz. `WAMCC`: Compiling Prolog to C. In Leon Sterling, editor, *Proceedings of International Conference on Logic Programming*, pp. 317–331. The MIT Press, Jun 1995.

[7] Jonathan J. Cook. P#: a concurrent prolog for the .net framework. *Software: Practice and Experience*, Vol. 34, pp. 815–845, July 2004.

[8] Bart Demoen and Paul Tarau. jProlog home page, 1996. `http://www.cs.kuleuven.ac.be/~bmd/PrologInJava/`.

[9] Jesper Eskilson and Mats Carlsson. Sicstus mt – multithreaded execution environment for sicstus prolog. In Konstantinos Sagonas, editor, *Proceedings of the JICSLP'98 Post Conference Workshop 7 on Implementation Technologies for Programming Languages based on Logic*, pp. 59–71, June 1998.

[10] Gopal Gupta, Enrico Pontelli, Khayri A. M. Ali, Mats Carlsson, and Manuel V. Hermenegildo. Par-

allel execution of prolog programs: a survey. *ACM Transactions on Programming Languages and Systems*, Vol. 23, No. 4, pp. 472–602, July 2001.

[11] Holger H. Hoos and Thomas Stutzle. SATLIB: An online resource for research on sat. In T.Walsh I.P.Gent, H.v.Maaren, editor, *SAT 2000*, pp. 283–292. IOS Press, 2000. `http://www.satlib.org/`.

[12] IF Computer. MINERVA home page, 1996. `http://www.ifcomputer.com/MINERVA/`.

[13] Katsumi Inoue, Yoshito Sasaura, Takehide Soh, and Seiji Ueda. A competitive and cooperative approach to propositional satisfiability. *DISCRETE APPLIED MATHEMATICS*. (submitted).

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

[15] Takao Kawamura, Shin Kinoshita, and Kazunori Sugahara. Implementation of a mobile agent framework on java environment. In Teofilo Gonzalez, editor, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, pp. 589–593, November 2004. MIT, Cambridge, USA.

[16] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfyability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997)*, pp. 366–371, 1997.

[17] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pp. 530–535. ACM, 2001.

[18] M.R.Garey, D.S.Johnson, and R.Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics Operation Research*, Vol. 1, pp. 117–129, 1976.

[19] Paul Tarau. Jinni: a lightweight java-based logic engine for internet programming. In Kostis Sagonas, editor, *Proceedings of JICSLP'98 Implementation of LP languages Workshop*, June 1998. invited talk.

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

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

[22] Michael Winikoff. W-Prolog home page. `http://goanna.cs.rmit.edu.au/~winikoff/wp/`, 1996.

[23] Eric Wohlstadter, Stoney Jackson, and Premkumar T. Devanbu. 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.

[24] 上田盛慈, 鵜飼訓史, 井上克巳, 番原睦則, 田村直之, 川村向生. Sat ソルバの並列実効に関する一考察. 電子情報通信学会「人工知能と知識処理」研究会, 2003.

[25] 番原睦則, 姜京順, 田村直之. 線形論理型言語の Java 言語による処理系の設計と実装. 情報処理学会論文誌: プログラミング, Vol. 40, No. SIG 10 (PRO 5), pp. 1–16, 12 月 1999.

[26] 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, pp. 39–60, 2001.

[27] 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.

[28] 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, pp. 138–154, 2001.