


- 
- This slide was
 - a material for the “Reading PLDI Papers (PLDIr)” study group
 - written by Kazuhiro Inaba (www.kmonos.net), under my own understanding of the papers published at PLDI
 - So, it may include many mistakes etc
 - For your correct understanding, please consult the original paper and/or the authors’ presentation slide!

k.inaba (稲葉 一浩), reading:

PLDIr #13
Jun 30, 2010

paper written by J. Baker and W. C. Hsieh (PLDI 2002)

MAYA: MULTI-DISPATCH SYNTAX EXTENSION IN JAVA

要するにJava用マクロシステム

- たとえばこんな拡張構文が作れます

```
use EForEach;
h.keys().foreach( String s ) {
    System.err.println(s + "->" h.get(s));
}
```

- こう展開される

```
for(Enumeration e$ = h.keys();
     e$.hasMoreElements(); ) {
    String s;
    s = (String) e$.nextElement();
    System.err.println(s + "->" + h.get(s));
}
```

どんな風に記述するか？

- 例題：

```
h.keys().foreach( String s ) {  
    System.err.println(s + "->" h.get(s));  
}
```

- まず宣言

```
abstract Statement syntax(  
    MethodName(Formal) lazy(BraceTree, BlockStmts)  
);
```

- Statement ::=
 MethodName '(' Formal ')' BlockStmts
- という構文を増やします！と宣言
- (※ 下線部はMayaの予約語)

実装の本体

名前をつけて、useで
lexical scopeに選択的導入可能

```
Statement syntax EForEach (
  Expression:Enumeration enumExp
  \. foreach (Formal var)
  lazy(BraceTree, BlockStmts) body
){
  final StrictTypeName castType =
    StrictTypeName.make(var.getType());
  return new Statement {
    for(Enumeration e = $enumExp;
      e.hasMoreElements(); ) {
      $(DeclStmt.make(var))
      $(Reference.makeExpr(var.getLocation()))
      = ($castType) e.nextElement();
      $body
    }
  };
}
```

「Enumeration型の式」
など、型に基づく条件

型検査とparseの
フェーズを混ぜる
lazy-parse機能

コードテンプレート
(Lispの準クオート)

“衛生的な”変数名

```
for(Enumeration e$ = h.keys();
  e$.hasMoreElements(); ) {
  String s;
  s = (String) e$.nextElement();
  System.err.println(s + “->” + h.get(s));
}
```

特徴・評価

- Java用マクロシステム
 - <http://www.cs.utah.edu/~jbaker/maya/>
 - 識別子や型情報を使った、適用条件の制御
 - ASTオブジェクトに対する
総称関数(マルチメソッド)として実現
 - Lazy parsing/typechecking
 - Lexical Scopeで拡張を入れたり出したり可能
 - 準クォートによる、書きやすいAST生成
- MultiJava (Java + マルチメソッド) を2500行程度で実装できた
とのこと

```
class D extends C
{ int m(C c) { return 0; }
  int m(C@D c) { return 1; } }
```

k.inaba (稲葉 一浩), reading:

PLDIr #13
Jun 30, 2010

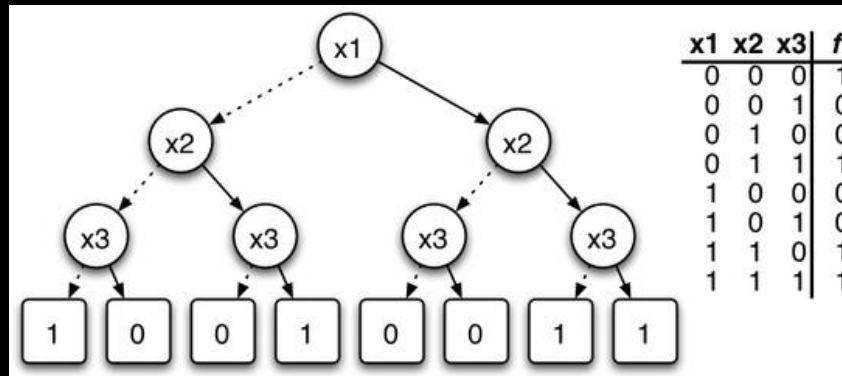
paper written by O. Lhoták and L. Hendren (PLDI 2004)

JEDD: A BDD-BASED RELATIONAL EXTENSION OF JAVA

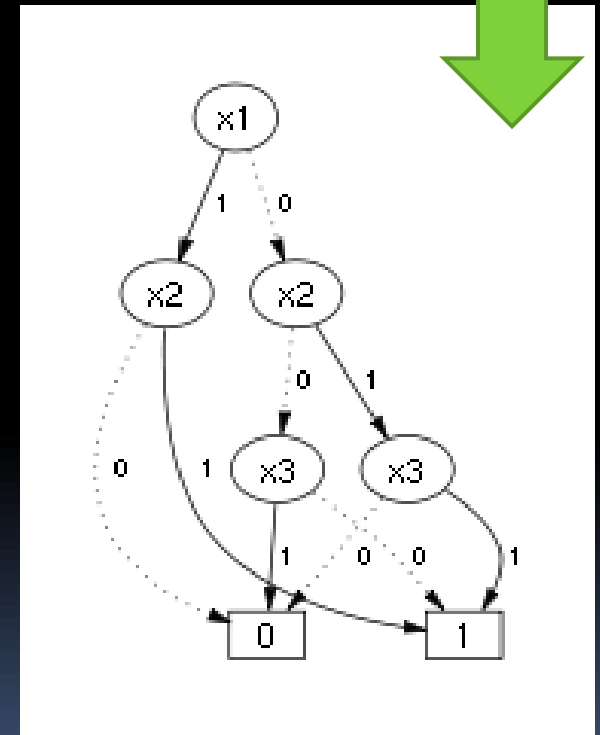
BDD (二分決定図) とは

これがBDD

- Boolⁿ から Bool への関数の効率的な表現 (下図はWikipediaから引用)



- プログラム解析などで重要なデータ構造
- and, or を始め論理演算が効率的に実現される



Jedd とは

- BDD を組み込みの言語機能として持つ
Java を拡張した言語
 - 生のBDDをそのまま見せても使いにくいので
リレーショナルDB風味の使い勝手を提供

RDB

1 st author	title	year
Baker	Maya	2002
Lhoták	Jedd	2004

Baker=00, Lhoták = 01

Maya=000, Jedd=001

2002=0, 2004=1 とする

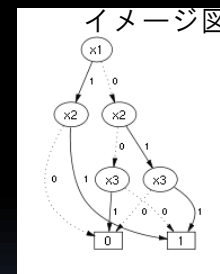
(十分なサイズのビット列割り当てはJeddが上手いことやる)

BDD

$$f(0,0,0,0,0,0) = 1$$

$$f(0,1,0,0,1,1) = 1$$

$$f(\text{それ以外}) = 0 \quad \text{という関数}$$



使用例：仮想関数呼出の実体解決

```
<rectype, signature, tgtype, method> answer = 0B;

void resolve( <type, signature> receiverTypes,
              <subtype, supertype> extend      ){
    <rectype, signature, tgtype> toResolve =
        (type=>rectype tgtype) receiverTypes;
    do {
        <rectype, signature, tgtype, method>
            resolved = toResolve{tgtype, signagure}
                << declaresMethod{type, signature};
        answer |= resolved;
        toResolve -= (method=>) resolved;
        toResolve = (supertype=>tgtype)
            (toResolve{tgtype} <> extend{subtype});
    } while( toResolve != 0B );
}
```

receiverTypes

type	signature
B	foo()
B	bar()

extend

subtype	supertype
B	A

declaresMethod

type	signature	method
A	foo()	A.foo()
B	bar()	B.bar()

<rectype, signature, tgtype, method> answer = 0B;

```
void resolve( <type, signature> receiverTypes,
             <subtype, supertype> extend ) {
```

=> は、改名兼
コピー兼射影

```
<rectype, signature, tgtype> toResolve =
  (type=>rectype tgtype) receiverTypes;
```

rectype	signature	tgtype
B	foo()	B
B	bar()	B

do {

```
<rectype, signature, tgtype, method>
```

```
  resolved = toResolve{tgtype, signagure}
```

```
    << declaresMethod{type, signature};
```

ジョイン

rectype	signature	tgtype	method
B	bar()	B	B.bar()

```
  answer |= resolved;
```

```
  toResolve -= (method=>) resolved;
```

射影して差演算

```
  toResolve = (supertype=>tgtype)
```

```
    (toResolve{tgtype} << extend{subtype});
```

合成して改名

rectype	signature	tgtype
B	foo()	B

```
  } while( toResolve != 0B );
```

rectype	signature	supertype
B	foo()	A

}

特徴・評価

- BDDを手軽に使うためのRDB風Java拡張
 - <http://www.sable.mcgill.ca/jedd/>
 - バックエンドのBDDは外部libを用いる
 - 便利な機能いろいろ
 - 意味のあるデータ名からo1列へのエンコード
 - "型"が合ってることの静的検査
 - join や合成など、高度な演算のサポート
 - 変数ドメインの分割や並び順の最適化 (SATソルバを使用)・プロファイラの提供
- 手書きC++と比べて points-to-analysis で 0.5~4%のオーバーヘッド

Benchmark	Std. lib. version	C++	Jedd
javac	1.1.8	3.4 s	3.5 s
compress	1.3.1	21.7 s	22.4 s
javac	1.3.1	25.3 s	26.3 s
sablecc	1.3.1	25.4 s	26.1 s
jedit	1.3.1	41.1 s	41.3 s