Java アプリケーション組込み用の Lisp ドライバ

湯淺太一
京都大学情報学研究科

組込みのLisp処理系

- よく知られた例
  - Emacs Lisp
  - AutoCAD の AutoLisp
- コンパイラにも
  - S式に似た中間言語やマシン記述
  - Lisp 処理系を組み込むことによって，開発期間短縮，コスト削減
  - COINS（コンパイラの共通インフラ）プロジェクト
- 問題点
  - Lisp 処理系実装のノウハウを持たないアプリケーション開発者
  - 限られた開発期間
  - 既存処理系の改造は難しい

Java アプリケーション用の Lisp 処理系

- **設計・実装方針**
  - **処理系実装のノウハウを持たないJavaプログラマにも機能の追加・削減・変更が容易に**
  - Java **部品を扱う機能追加が容易に**
    ⇒ **処理系記述言語は** Java
  - **コンパクト**
  - **最低限のデバッグ機能**
  - **そこそこの実行性能**

組込み関数の定義 (ぶぶ)

```
public static void Lcar(BCI bci) {
  Object x = bci.vs[bci.vsbase + 1];
  if (!(x instanceof List))
    throw SE.notList(x);
  bci.acc = ((List) x).car;
}
```

- 「ぶぶ」の実装に関する様々な知識が必要
  - マルチスレッド機能
  - スタック構造
  - 引数･返り値の受け渡し方法
- 型チェックの重複

組込み関数の定義 (本処理系)

```
public static Object car(List x) {
  return x.car;
}
```

```
public static Pair cons(Object x, Object y) {
  return new Pair(x, y);
}
```

```
public static Object setCar(Pair x, Object val) {
  return x.car = val;
}
```

特殊フォームの定義

```
(if c e1 e2)
```

```
public static
Object Lif(Object c, Object e1, Object e2, Env env) {
  if (eval(c, env) != F)
    return eval(e1, env);
  else if (e2 == null)
    return List.nil;
  else
    return eval(e2, env);
}
```

Java とのインターフェイス

- Lisp レベルでは必要ない
- Java レベルで書くほうが簡単

```
public static bitvec bitvec_and(bitvec x, bitvec y) {
  return x.bitvec_and(y);
}
```

処理系のコンパクト化

- Java の機能をできるだけ利用
  - メモリ管理とごみ集め
  - Lisp データを標準のクラスで表現
  - 入出力機能
  - 例外処理機能
  - reflection 機能

最低限のデバッグ機能

- 適切なエラーメッセージ表示
- バックトレース

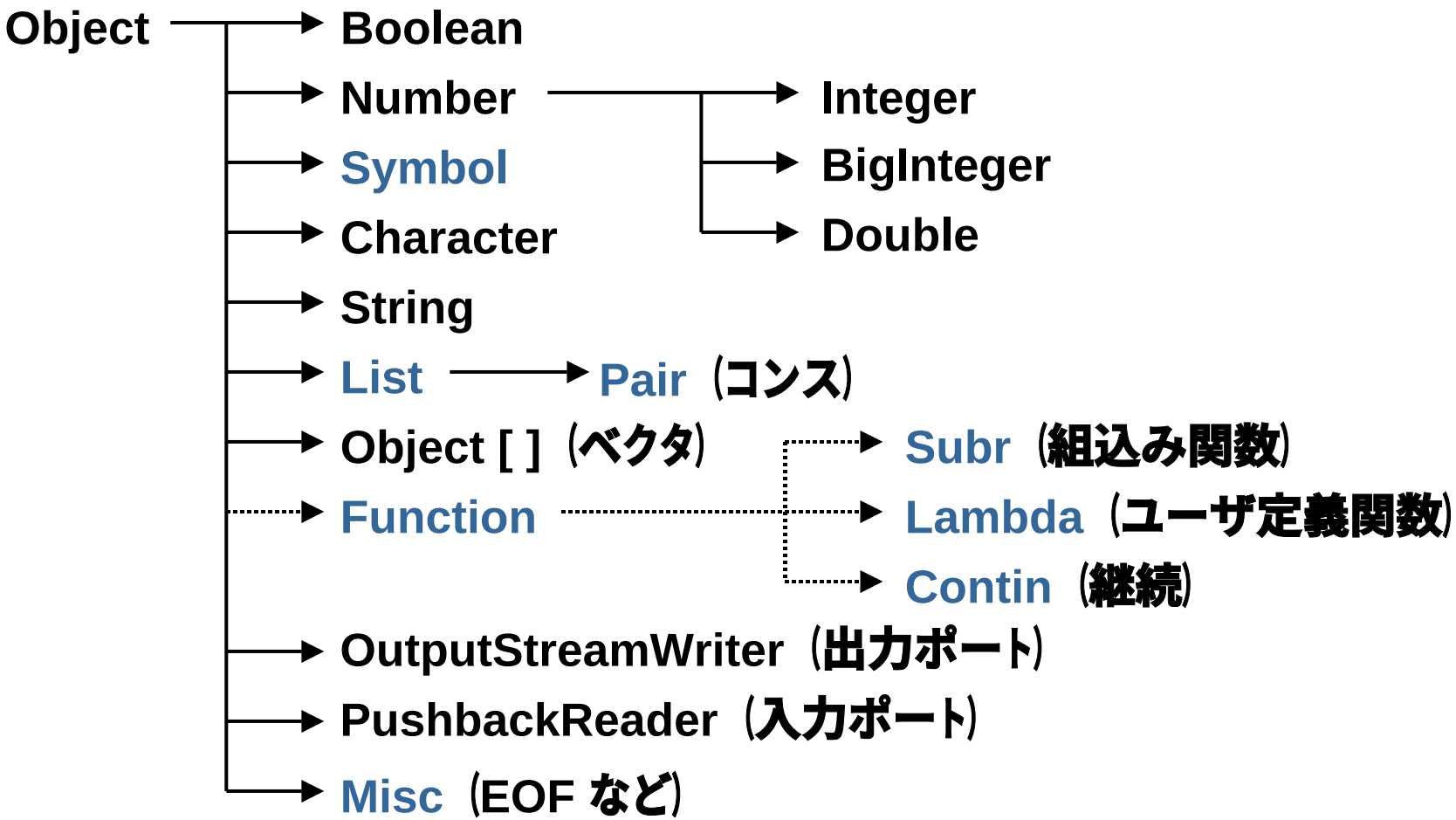
```
>(define (fact x)
   (if (zero? x)
       (/ 1 0)
       (* x (fact (- x 1)))))
>(fact 3)
ArithmeticException: / by zero
Backtrace: / < if < fact < if < fact < if < fact < if <
fact < top-level
>
```

- 関数トレース

言語仕様

- **本質的ではないが. . .**
- IEEE Scheme **のほぼフルセット**
  - **継続は** escape procedure (**非局所的脱出**)
  - **末尾再帰の最適化なし**
  - **文字列は** immutable

データ表現


Object
Boolean
Number
Integer
BigInteger
Double
Symbol
Character
String
List
Pair（コンス）
Object [ ]（ベクタ）
Function
Subr（組込み関数）
Lambda（ユーザ定義関数）
Contin（継続）
OutputStreamWriter（出力ポート）
PushbackReader（入力ポート）
Misc（EOF など）

インタープリタ

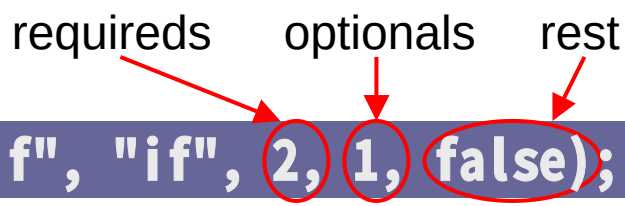
```
static Object eval(Object expr, Env env)
```

- コンパイラ方式だと，Java プログラマによる改造は絶望的
  - 「ぶぶ」の if 式実装例

```
(define (c2if fmla form1 form2)
   (if (and (eq? (car form2) c2constant)
            (eq? *value-to-go* 'trash)
            (member *exit* '(next escape)))
       (let ((tlabel (next-label)) (flabel *exit-label*))
         (dlet ((*exit* 'next) (*exit-label* tlabel)) (cjf fmla flabel))
         (wt-label tlabel)
         (c2expr form1))
       (let ((tlabel (next-label)) (flabel (next-label)))
         (dlet ((*exit* 'next) (*exit-label* tlabel)) (cjf fmla flabel))
         (wt-label tlabel)
         (case *exit*
           ((next) (dlet ((*exit* 'escape)) (c2expr form1)))
           ((unwind-next) (dlet ((*exit* 'unwind-escape)) (c2expr form1)))
           (else (c2expr form1)))
         (wt-label flabel)
         (c2expr form2))))
```

特殊フォームの定義

```
(if c e1 e2)
```




```
defSpecial("Eval", "Lif", "if", 2, 1, false);
```

```
public static
Object Lif(Object c, Object e1, Object e2, Env env) {
  if (eval(c, env) != F)
    return eval(e1, env);
  else if (e2 == null)
    return List.nil;
  else
    return eval(e2, env);
}
```

特殊フォームの定義

```
(begin e e1 ... en )
```

```
defSpecial("Eval", "begin", "begin", 1, 0, true);
```

```
public static Object begin(Object e, List es, Env env) {
  Object val = eval(e, env);
  while (es != List.nil) {
    val = eval(es.car, env);
    es = (List) es.cdr;
  }
  return val;
}
```

組込み関数の定義

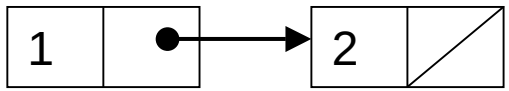
```
def("List", "car", "car", 1, 0, false);
```

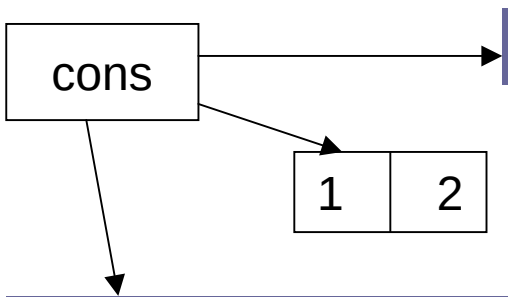
```
public static Object car(List x) {
  return x.car;
}
```

関数呼出し

- スタックを使わず，敢えて evlis 方式
  ⇒ 大域的な制御情報の廃止
- Java の reflection 機能を利用

```
(cons a b)
```






```
def("List", “cons", “cons", 2, 0, false);
```

```
public static Pair cons(Object x, Object y) {
  return new Pair(x, y);
}
```

多用途の関数実装メソッド

```
def("List", "list2vec", "list->vector", 1, 0, false)
```

```
def("List", "list2vec", "vector", 0, 0, true)
```

```
public static Object[] list2vec(List x) {
   ...
}
```

- 内部的にも利用
  - バッククオートマクロ
  - ベクタリテラルの読み込み

```
>(list->vector '(1 2 3))
#(1 2 3)
>(vector 1 2 3)
#(1 2 3)
```

エラーメッセージの変換: Java→Lisp

- Reflection **が検出するエラー**

  IllegalArgumentException: argument type mismatch

  → RuntimeException: the first argument is not a Pair

- Cast **が検出するエラー**

  ClassCastException: List

  → RuntimeException: unexpected List object

```
public static Object nth(List x, int n) {
  while (--n >= 0)
    x = (List) x.cdr;
  return x.car;
}
```

エラーメッセージの変換: Java→Lisp

- Object Subr.invoke(List args) **の関数呼出し処理**

```
引数リストの要素を引数ベクタへ格納
引数の個数が不適切なら例外発生
try {
  return method.invoke(null, argV);
} catch (InvocationTargetException e) {
  e.getTargetException() 内の
  エラーメッセージを変換して投げなおし
} catch (IllegalArgumentException e) {
  引数の型チェックと例外発生
}
```

バックトレース

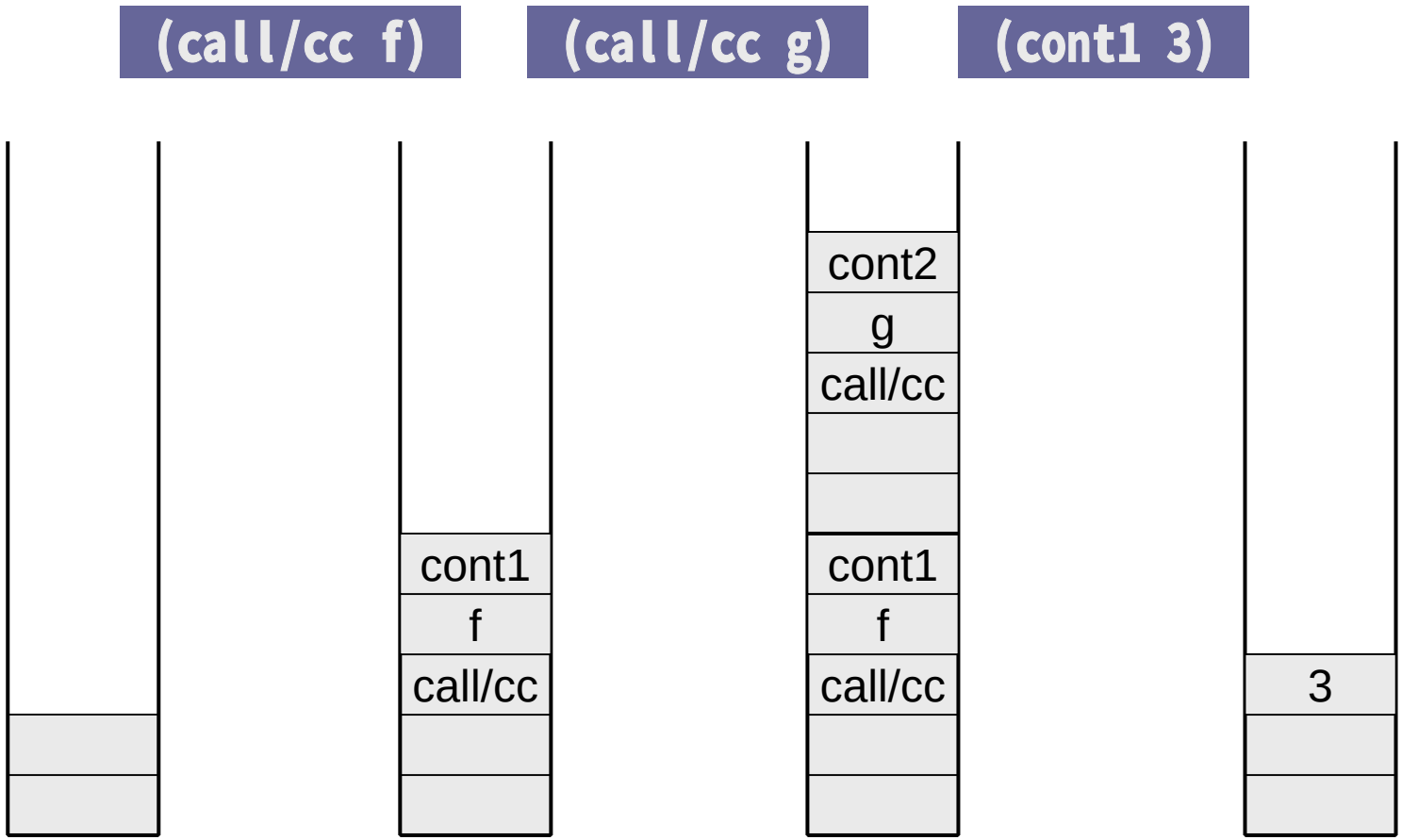
```
>(define (fact x)
   (if (zero? x)
       (/ 1 0)
       (* x (fact (- x 1)))))
>(fact 3)
ArithmeticException: / by zero
Backtrace: / < if < fact < if < fact < if < fact < if <
fact < top-level
```

バックトレース

```
try {
  関数の呼出し処理
} catch (Throwable e) {
  if (e != backtraceToken) {
    エラーメッセージを表示し，
    バックトレース表示を開始する．
  } else {
    呼び出した関数の名前を，
    バックトレースの一部として表示する．
  }
  throw backtraceToken;
}
```

```
ArithmeticException: / by zero
Backtrace: /
```

継続 (escape procedure)



継続 (escape procedure)

```
(call/cc f)
```

```
public static Object callcc(Function f) {
  Contin cont = new Contin();
  try {
    return f.invoke1(cont);
  } catch (Contin c) {
    if (c == cont)
      return c.value;
    else
      throw c;
  } finally {
    cont.canCall = false;
  }
}
```

```
try {
  関数の呼出し処理
} catch (Contin c) {
  throw c;
} catch (Throwable e) {
  ...
}
```

入出力

- **出力ポート**: OutputStreamWriter
  - OutputStream **をラッピング**
  - **指定された文字コードで日本語出力**
- **入力ポート**: PushbackReader
  - InputStreamReader **をラッピング**
    - InputStream **をラッピング**
    - **指定された文字コードで日本語入力**
  - **文字単位の** unread **を実現**

入出力

- Reader は, Common Lisp 風の「文字属性」と「入力マクロ文字」を採用
  - ただし, Lisp レベルでの変更・追加は不可
  - トークン切り出しのコードをすっきりと
  - Reader の大幅な改造を可能に

入出力

- 数値のパースは Java の機能を流用
  - 字句構造は Java とまったく同じ
- Printer は Reader と整合性
  - やはり Java のパース機能を利用
  - 記号の印字結果を保存して再利用

```
try {
  return makeInt(Integer.parseInt(s));
} catch (NumberFormatException e) { }
try {
  return new BigInteger(s);
} catch (NumberFormatException e) { }
try {
  return new Double(s);
} catch (NumberFormatException e) {
  return Symbol.intern(s);
}
```

他の処理系との比較

- Java で記述され，ソースが公開されている処理系
  - HotScheme: Scheme風の独自仕様，S式インタープリタ
  - Skij: ほぼ IEEE 仕様，S式インタープリタ
  - Kawa: ほぼ IEEE 仕様，JVM へコンパイル
  - Jscheme: ほぼ IEEE 仕様，中間コードにコンパイル
  - SISC: R5RS 仕様に完全準拠，中間コードにコンパイル

他の処理系との比較

- **特殊フォームの改造**
  - **コンパイル方式の** Kawa, Jscheme, SISC **では絶望的**
  - HotSchemeとSkij**の特殊フォームはリストを受け取り，自分でチェック ⇒ あちこちにチェック漏れ**

```
HotScheme >> (if)
error: Arg not a list:  in car of: #f
```

```
skij> (if)
SchemeException: Can't eval null
```

```
>(if)
RuntimeException: too few arguments to if
```

他の処理系との比較

- **組込み関数の実装**
  - HotScheme, Skij, Kawa
    - **組込み関数ごとにクラスを定義**
    - **膨大なクラス数，肥大化した処理系**
  - Jscheme と SISC
    - **個々の組込み関数に識別番号**
    - **実行時に** switch **文で分岐**
    - **組込み関数の追加は面倒**
- **バックトレース機能**
  - HotScheme と SISC: **バックトレースなし**
  - Kawa: Java **レベルのみ**
  - Skij と Jscheme: **本処理系より複雑，有効性は疑問**

他の処理系との比較

- **継続**
  - HotScheme: **継続なし**
  - Skij **と** Jscheme：Escape procedure
    - **呼出し時のチェックなし ⇒ 処理系が異常動作**
  - Kawa: Escape procedure
    - **継続オブジェクトと例外オブジェクトが何故か別物**
  - SISC: **本物の継続**
- **入出力**
  - Kawa: **文字属性方式を採用，パースは自前**
  - Jshcme: **パースは** Java **機能を利用，出力が整合していない**
  - HotScheme, Skij, SISC: **とても解読する気にならない**

実装用クラスとそれらのサイズ

クラス名	Java ソース		クラスファイル
	行数	バイト数	バイト数
Char	256	7,786	6,618
Contin	61	1,307	1,516
Env	66	1,655	1,888
Eval	576	18,377	12,006
Function	10	172	257
IO	767	23,128	15,446
Lambda	61	1,479	2,259
List	449	11,666	8,353
Misc	14	192	342
Num	728	22,462	12,267
Pair	10	168	287
Subr	219	6,714	6,692
Symbol	235	6,541	6,009
合計	**3,452**	**101,647**	**73,940**

処理系のサイズ比較

処理系	Java ソース		クラスファイル		Scheme ソース	
	ファイル数	バイト数	ファイル数	バイト数	ファイル数	バイト数
本処理系	13	101,647	13	73,940	0	0
HotScheme	114	139,363	121	134,811	0	0
Skij	41	151,841	173	280,427	53	122,439
Kawa	135	338,927	160	429,886	18	48,582
Jscheme	51	266,338	55	228,150	42	262,642
SISC	90	408,201	96	223,001	38	381,559
ぶぶ	55	470,255	69	309,675	8	170.232

ベンチマーク実行結果

テスト	本処理系	KCL	Kawa	Skij
ctak	15.754	1.700	8.329	16.826
deriv	8.569	1.660	2.550	6.425
div (ite)	6.505	0.890	0.829	10.587
div (rec)	8.445	1.720	0.909	8.535
fprint	0.106	0.010	0.118	0.981
fft	27.310	7.600	11.192	エラー
fread	0.851	0.030	0.506	0.059
puzzle	76.258	13.410	20.547	96.751
tak	4.399	1.486	1.021	4.172
takr	4.663	1.790	1.561	5.017
tprint	0.050	0.010	0.096	1.138
traverse (init)	69.476	15.870	20.598	132.938
traverse (run)	363.169	101.520	56.675	373.436
triangle	1067.892	230.320	219.743	1000.534

ユーザ様の声

From 某@fujixerox.co.jp Fri Sep 27 18:43:32 2002
To: "YUASA Taiichi" <yuasa@kuis.kyoto-u.ac.jp>
Subject: ありがとうございました

湯浅先生、昨日はどうもありがとうございました。

早速いただいて帰った新処理系を iPAQ 上で動かしてみました。
iPAQ にバンドルされていた JeodeRuntime という VM は
簡易コンソールがあるので、難なく実行できました。
クラスファイルのみの Jar ファイルサイズは 36KB でした。

iPAQ 上で動作する簡単な UI を実装する予定なので、
このために使わせていただきたいと思います。

某

今後の予定

- 名前を決める
- 湯淺のホームページからフリーでダウンロードできるよう設定
- 使用経験による評価
- Common Lisp 風の仕様
- 教育用にも