Javaで書いたScheme処理系JAKLDのSICP演習用機能拡張

湯淺 太一

京都大学工学部情報学科では数年前から，１年生後期と２年生前期の１年間をかけて，Structure and Interpretation of Computer Programs（略称 SICP）を使ったプログラミングの講義を行っている．この講義には演習時間を設けていないが，ほぼ毎週演習課題を出しており，受講生は実際に Scheme 処理系を使って課題を解くことが要求される．演習に使用する計算機環境や処理系は学生によってまちまちであり，学部１年生には自分でインストールするのが難しかったり，学生によって処理系の動作が異なるといった不都合があった．これらを解消するために，Java で記述した JAKLD という Scheme 風の Lisp 処理系に手を加え，演習に必要な末尾再帰呼出しの最適化や図形言語を追加した．Java VM さえあれば，容易にインストールでき，様々な OS で利用することが可能である．処理系拡張に至った背景，従来の問題点，SICP 演習用処理系としての要求事項，実装方法などを報告する．

1 はじめに

京都大学工学部情報学科では数年前から，１年生後期と２年生前期の１年間をかけて，Structure and Interpretation of Computer Programs（略称 SICP）[1] を使ったプログラミングの講義を行っている．SICP は，基礎的なものから高水準なものまで，プログラミングの様々な概念をカバーしており，MIT をはじめ国内外の多くの大学でプログラミングの教科書として採用された実績を持っている．原著は英文であるが，邦訳[15] もあり，また英文のフルテキストを出版社の Web ページからダウンロードすることもできる[9]．例題や演習に使うプログラミング言語は Scheme [4] である．

講義科目は，１年生後期の「アルゴリズムとデータ構造入門」[7] と２年生前期の「プログラミング言語」[8] とで担当者が異なるが，いずれの講義もほぼ毎週演習課題を出しており，受講生は実際に Scheme 処理系を使って課題を解くことが要求されている．講義には演習時間を設けていないので，受講生は各自の都合に合わせて計算機環境や処理系を選択する．演習結果は翌週にレポートとして提出し，TA が採点して返却する．一部の演習課題について講義で解説を行ったり，採点過程で気がついた事項を講評することもある．

数年前に SICP を使った講義を始めた際に，学生の演習用として，IEEE 仕様に準拠する Scheme 処理系である TUTScheme/Tk[3] を，学術情報メディアセンター（かつての教育用計算機センター）にインストールした．この処理系は，筆者らの研究室で開発した TUTScheme[17] に，SICP の演習に必要な図形言語（後述）の機能を追加したものである．講義開始までに時間の余裕がなかったこともあり，TUTScheme に Tcl/Tk[14] の Tk 機能を追加して演習ができるようにしたが，Tk 機能の制約から，SICP の図形言語機能すべてをサポートすることはできなかった．

このような状況で数年間講義を行ってきて，その間に明らかになった様々な不都合を解消し，受講生が演習問題を解くことに専念できるような環境を提供するために，今回，Java で記述した JAKLD[18] という Scheme 風の Lisp 処理系に手を加え，演習に必要な諸機能を追加した．Java VM さえあれば，容易にインストールでき，様々な OS で利用することが可能である．以下では，第２節で SICP 演習に用いる処理系

Taiichi Yuasa, 京都大学大学院情報学研究科, Graduate School of Informatics, Kyoto University.

を準備する上での課題をあげ，続く第３節で JAKLD の概要を述べ，最後に，SICP 演習用の機能拡張とその実装について報告する．

2 課題

本節では，講義を進める過程で明らかになった問題点をあげ，それらを解決するために処理系のどのような特性が SICP 演習のために望ましいかを議論する．

2.1 学生の多様な演習環境

学生が演習に使う計算機は，大きく次の三つに分類される．

- メディアセンターの PC
- 自分のノート PC
- 所属クラブや親の PC

前述のように，メディアセンターには TUTScheme/Tk がインストールされており，図形言語以外の演習は支障なく行える環境が提供されている．しかし，利用可能時間が平日午後８時まで，土曜は午後６時まで，日曜は休館であり，学生にとっては夜間の時間の制約は大きいようで，敬遠されがちである．

下宿生の多くは自分のノート PC を持っている．TUTScheme/Tk は Web ページからダウンロードできるし，SICP ともっとも相性が良さそうな MIT Scheme[10] もダウンロード可能である．学生が自分の PC で演習を行いたい場合は，これらの処理系をまず PC にインストールする必要がある．しかし，学部１年生にとって，処理系の make ファイルを編集して適切なパラメータを設定するのは容易ではない．特にグラフィックスまわりの設定が難しいようで，演習を始める前にかなりの時間を make 作業に費やすことが多いと聞いている．make の問題は，クラブや親の PC を借りて演習する場合にもあてはまるが，借り物の場合は，管理者特権がないために，インストール自体が許されないという問題もある．

2.2 演習に必要な機能

Scheme の規格として公式のものは IEEE 規格[4] であるが，実際には，R4RS[2] や，それの改訂版である R5RS[6] や最新の R6RS[11] が，言語仕様として参照されることが多い．SICP の出版が 1996 年だから，当時の最新仕様であった R4RS に準拠するように本文や演習問題が著述されているものと思われるが，実際に演習に利用するためには，処理系に次の追加機能のあることが望ましい．

- `true`, `false`, `nil`: SICP ではそれぞ`#t`, `#f`, `()` を値とする変数としてコード中で使用されているが，Scheme の規格では，これらの変数は定義されていない．SICP の本文や脚注で，これらが標準でない旨が説明されているが，学生たちが気がつかない可能性が高い．学生が演習中にこれらの変数を使ってしまい，それらを定義していない処理系でテストしようとすると，「教科書に載っている変数が定義されていない」と不可解に感じるであろう．これらは，次の３行で定義できる．

  ```
  (define true #t)
  (define false #f)
  (define nil '())
  ```

  しかし，処理系があらかじめ定義しておくのにこしたことはない．
- `random`: 疑似乱数を発生するこの関数は，Scheme 規格では定義されていない．実際，SICP では

  > ... the procedure `random`, which we assume is included as a primitive in Scheme.

  と本文で注意がある．多くの処理系がこの関数を実装しているようだが，学生が利用している処理系が実装しているとは限らない．一方，`random` を使う演習問題は少なくない．まだプログラミング（しかも Scheme プログラミング！）に不慣れな学生たちがそのような状況に遭遇すると，インターネットを検索して適当な定義を入手しない限り演習問題が解けないことになる．
- グラフィック機能: SICP の 2.2.4 節では，図形言語を使ってデータ抽象の概念，特に階層的データ構造とクロージャについて学ぶ．ここで使っている図形言語は，線画や画像といった「図形」に対して，任意のアフィン変換によって拡大・縮小，反転，変形といった操作をほどこし（図１参

照），それらを組み合わせて絵を描くことができる（図 2 参照）．きわめて単純な操作によって印象的な絵を構成することができ，SICP の目玉となっている．具体的には，次のような機能が必要である．

- 頂点を指定して，それらを繋ぐ線画を描く機能
- PNG や JPEG などの画像ファイルを指定して，画像データを読み込む機能
- 画像データに対する任意のアフィン変換
- 合成した絵をディスプレイに表示したり，ファイルに画像データとして保存する機能．後者は SICP には現れない機能であるが，生成した画像をレポートに添付して提出するためには是非備わってほしいものである．

もちろんこれらの機能は標準規格では定義されていないので，既存の処理系を使う場合には，TUTScheme/Tk のように処理系の拡張が必要となる．

- `ok` を値とする `define` 式: Scheme 規格では，`define` 式の値は処理系依存となっている．「式」と呼ばれているが，その値を計算に使うことはないためである．しかし，処理系との対話において，トップレベルで `define` 式を使うと，通常はなんらかの値が表示される．例えば TUTScheme では，`define` 式が定義する変数の名前が値となる．一方，SICP の後半では，遅延評価，非決定的計算，論理プログラミングなどのパラダイムに基づいた評価器が紹介されている．それらの評価器では，`define` 式の値はすべて `ok` になっている．変数名が表示されるのを見慣れている学生は，`ok` と表示されると奇妙な印象を持つであろう．些細なことではあるが，演習用の処理系は `ok` を返すようになっているのが無難である．

2.3 演習に望ましい処理系

以上を考慮すると，SICP 演習用の処理系としては，次の特徴を持ったものが望ましい．

1. 前節であげた「必要な機能」を完備していること

図 1　アフィン変換の例（左が原画，右が変換結果）

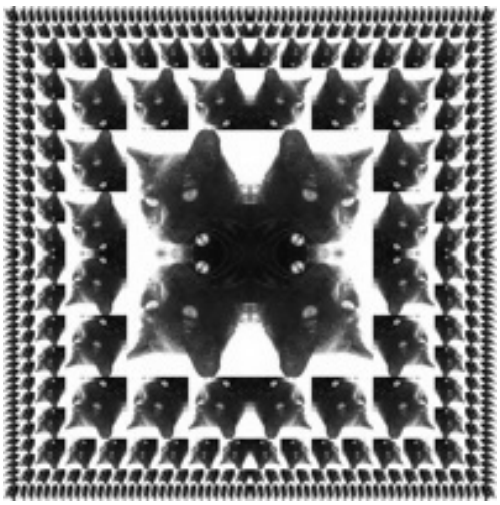

図 2　図形言語による描画例

2. 同一仕様で多様な計算機環境で動作すること．大多数の受講生は Windows PC を使っているが，なかには Unix や Mac OS を使う者もいる．様々な OS 上で同じように動作する処理系が望ましい．さらに，演習に必要なプログラムが共通に使えることが強く望まれるが，これについては後述とする．
3. インストールが簡単であること
4. 管理者権限がなくてもインストールし利用できること．メディアセンターにインストールしメンテナンスすることは，技術的および手続き的に負担が大きい．一般の学生でも，処理系の最新バージョンをダウンロードしてセンターのマシンで利用できることが望ましい．
5. 小さいこと．データ通信料金を考えると，短時間でダウンロードできるほうがよい．また，他人

の PC を借りている場合は，ディスクに負担をかけないものが欲しい．

なお，あくまでも学生の演習用処理系であるから，必ずしも高性能である必要はない．SICP が出版された 1996 年当時と現在とでは計算機の性能は比較にならないほど向上している．多少性能が劣る処理系であっても，10 年以上前に解けた演習問題なら解けるはずである．

2.4 演習用プログラムの共通化

SICP の後半では，遅延評価，非決定的計算，論理プログラミングなどのパラダイムに基づいた評価器が紹介されている．本文では，これら評価器を Scheme で定義していくが，その前に，これら評価器を使ったプログラム（例えば，非決定的計算を利用して簡単なパズルを解くプログラム）の作成練習を通して，それぞれのパラダイムを理解する．作成したプログラムが正しく動作するかどうかは，実際に各パラダイムの評価器を使って確認する．

それぞれの評価器を構成する諸関数は，機能の説明にあわせて本文のあちこちに散らばっている．学生が評価器を使うためには，これらを寄せ集めてきて，一つのシステムとしてまとめる必要がある．Scheme プログラミング初心者の学生たちにとっては，困難な作業である．実際，評価器を自分で構成できなかったために，プログラムをテストしないで提出するケースが多い．演習用には，それぞれのパラダイム用の評価器をあらかじめ教員側で準備し，学生に提供することが強く望まれる．

SICP の解答を掲載した Web ページのなかには，プログラム動作確認用の評価器も同時に提供していることがある．学生がそのような Web ページを参照することが望ましくないことは言うまでもないが，仮に評価器をダウンロードしても，学生の手元の処理系で直ちに利用できるとは限らない．評価器にバグがあることもあるし，特定の処理系に依存したコードのこともある．図 3 に，実際にそのような Web ページからダウンロードした評価器の一部を掲載する．

この関数は，非決定的計算の評価器の一部で，処理系の組み込み関数を，非決定的計算の評価器においても利用できるように設定するものである．（コメントは筆者が追加したものであり，適宜省略とインデントを行っている．）この中で，`error` は R6RS で採用された関数で，R5RS には含まれない．逆に，`inexact->exact` は R5RS で定義されていたが R6RS では削除されている．`runtime` は処理系固有の関数らしい．このように，特定の処理系での動作を前提としているようで，手元の処理系で試してみると，未定義の組込み関数があるためにエラーを起こすことがある．パラダイムのプログラミングをしようとする学生が，このような問題に直面して，適切に評価器を修正できるとは考えにくい．

```
(define primitive-procedures
  (list
    (list '* *)
    ...
    (list 'eq? eq?)
    (list 'error error) ;; new in R6RS
    (list 'list list)
    ...
    (list 'round round)
    (list 'runtime runtime) ;; not in R6RS
    (list 'set-car! set-car!)
    ...
    (list 'integer? integer?)
    (list 'inexact->exact inexact->exact)
      ;; was in R5RS but not in R6RS
    (list 'even? even?)
    ))
```

図 3　評価器を構成する組込み関数一覧

この例で明らかなように，演習のための評価器を学生に提供するときは，そのコードが，学生の使っているどの処理系においても，修正なしに直ちに利用できることが重要である．これを，演習用プログラムの共通化と呼ぶことにする．

3 JAKLD

以上の考察から，今回，Java で書かれた処理系 JAKLD をベースにし，SICP 演習に必要な機能を追加することによって，理想的な演習環境を提供することを目指した．Java で記述した処理系であれば，Java VM さえインストールされていれば，どの計算機のどの OS でも直ちに利用できることが期待され

る．処理系を jar ファイルの形にまとめておけば，特別なインストール作業は不要であり，管理者権限がなくてもダウンロードして利用できる．SICP 演習用追加機能でもっとも手間のかかりそうな図形言語のサポートは，Java のグラフィック機能を有効に利用すれば，比較的簡単に実現できそうである．さらに JAKLD の場合は，Java のクラスライブラリを徹底的に利用することによって，処理系自体がコンパクトに出来ており，ダウンロード時間やディスクの占有領域をあまり気にする必要がない．

3.1 JAKLD の概要

JAKLD は，Java アプリケーションに組み込んで使うことを目的として開発した Lisp ドライバである．その設計にあたっては，特に次の項目を重視した．

1. Lisp 処理系の実装ノウハウを持たない Java プログラマにも機能の追加・削除・変更が容易に行えること．
2. Java で開発したソフトウェア部品を扱うための機能を容易に組み込めること．
3. コンパクトな実装であること．
4. 高度な Lisp プログラム開発支援ツールを備える必要はないが，デバッグのために最低限必要な機能は備えること．
5. 高性能である必要はないが，性能が極端に悪くないこと．

項目 1 から，処理系記述言語は必然的に Java になる．Lisp の組込み関数を Java で容易に定義できれば，Java の部品を利用するためのインタフェースは容易に構築できるので，項目 2 も満たすことができる．また，Java の豊富なクラスライブラリを利用すれば，コンパクトかつ許容範囲の性能を有する処理系の実現が期待でき，残りの三つの項目も満たすことができる．

JAKLD の組込み関数は，きわめて理解しやすいように記述されている．その一例として，組込み関数の `car`，`cons`，`set-car!`の定義を図 4 にあげる．Lisp の言語仕様を理解している Java プログラマにとっては，これらの定義に説明はまったく不要であろう．

```
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;
}
```

図 4　JAKLD における組込み関数の定義例

処理系をコンパクトにするために，JAKLD では Java 実行系の持つ諸機能と豊富なクラスライブラリを有効に利用している．たとえば次の機能を，処理系実装に直接利用した．

- メモリ管理とごみ集め
- 標準的なクラスで，Lisp のデータ型としてそのまま利用できるもの
- 入出力関係の諸機能
- 例外処理機能
- reflection 機能

これらを有効に利用することによって，実装用のクラスはわずか 13 個にとどまり，ソースコードは約 3,500 行，100 K バイト程度に収まっている．また，Java コンパイラが生成するクラスファイルのサイズも，合計 74K バイト程度と，きわめてコンパクトな処理系を実現している．JAKLD はフリーソフトとして Web で公開されており [19]，いくつかの研究プロジェクト（例えば [13]）で言語機能の試験実装などのために利用されてきた．また，コンパクトな実装であるために，携帯電話端末で動作するバージョンもある [20]．

3.1.1 言語仕様

JAKLD は，IEEE Scheme [2] [4] のほぼフルセットをサポートしている．IEEE Scheme に準拠していないのは，次の 3 点である．

- 継続（continuation）は，それを生成した `call/cc` 関数がリターンした後は呼び出せない．つまり escape procedure [5] として機能し，Common Lisp [12] における catch&throw のような非局所的脱出には利用できるが，コルーチンを実現することはできない．これは，Java の実行

時スタックをヒープに退避するための処理系非依存の方法がないためである．

- 末尾再帰（tail-recursive）呼出しの最適化は行わない．これも，Java の実行時スタックを直接操作する方法がないためである [16].
- 文字列は immutable であり，既存の文字列中のある文字を別の文字で置き換えることはできない．これは，文字列を Java の `String` オブジェクトで表現しているためである．`String` オブジェクトをラッピングするクラスを定義すれば，mutable な文字列は容易に実現できるが，そのための処理系の肥大化や実行時オーバヘッドに見合うだけの価値があるとは思えない．

関数には，Subr（組込み関数），Lambda（ユーザ定義関数），Contin（継続）の 3 種類があり，それぞれ同じ名前の Java クラスによって定義されている．前述のように JAKLD の継続は escape procedure として機能するので,「継続を呼び出す」ための処理は,「例外を投げる」ための処理と本質的に同じになる．そこで，`Contin` を例外クラス（`RuntimeException` のサブクラス）と定義することによって，Java の機能を効果的に利用して処理系をコンパクトに抑えている．Java は多重継承を許さないので，3 種類の関数を統一的に扱うための `Function` クラスは，三つのクラスが実装（implements）するインタフェースとして定義している．

3.1.2 インタープリタ

処理系の改造が容易に行えるように，JAKLD はコンパイラ方式ではなく，S 式（評価の対象となる Lisp オブジェクト）を解釈しながら実行するインタープリタ方式を採用している．インタープリタは，`eval` というメソッドで実装されている．

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

S 式と環境（environment，局所変数の束縛情報）とを受け取り，与えられた環境の下で S 式を評価し，その結果を返す．`eval` への第 1 引数と評価結果は，任意の Lisp オブジェクト（`Object`）である．

S 式が特殊形式であれば，その cdr（先頭要素を除いた残りのリスト）と環境とを引数として，特殊形式を実行するための Java メソッドを呼び出す．これらのメソッドは，前述の組込み関数 `car` の定義と同様の，直感的に理解しやすい形式で定義されている．たとえば，条件分岐を行う `if` 式は，図 5 のように実装されている．このメソッド定義は，`if` 式

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

の実行を，その仕様に従って忠実に記述したものである．まず `eval` を再帰的に呼び出して条件 c を評価し，結果が真（`Boolean.FALSE` 以外）であれば，再度 `eval` を呼び出して e_1 を評価する．条件 c の評価結果が偽であれば，e_2 を評価する．e_2 は省略可能であり，省略された場合は空リストを返す．Java 言語の `null` は，Lisp データにはなりえず，この定義のように，引数が与えられなかったことを示すような場合に用いられる．なお，メソッド名の “`Lif`” は，特殊形式名と同じ “`if`” としたいところだが，“`if`” は Java の予約語であり識別子としては使えないので，“`Lif`” としている．

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

図 5 JAKLD における特殊形式 `if` の定義

特殊形式と，それを実装するメソッドをリンクするには，`defSpecial` というメソッドを使う．`if` 式の場合であれば次の式を実行する．

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

`Eval` クラスで定義されている `Lif` というメソッドが，特殊形式の `if` 式を実装し，`if` 式は少なくとも 2 引数を受け取り，省略可能な引数をもう一つ受け取ることができることを表している．`defSpecial` の第 4 と第 5 のパラメータは非負整数であり，特殊形式が最低限必要とする引数の個数と省略可能な引数の個数をそれぞれ指定する．`defSpecial` への最後のパラメータは，特殊形式が任意個の引数を受け取れるかどうかを表す．`if` 式の場合はたかだか三つしか引数を受

け取れないので，このパラメータに対する実引数は，false である．

組込み関数と，それを実装するメソッドをリンクするには，def というメソッドを使う．例えば，関数 car の場合は，次のように指定する．

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

def への引数の意味は，defSpecial への引数とまったく同じである．

関数呼出しの実行は，環境を受け渡す必要がないことを除けば，特殊形式の実行とほとんど同じである．唯一の相異点は，S 式中の引数をそのまま受け渡すのではなく，それらを評価した結果を受け渡す点である．JAKLD は，引数の評価と受け渡しのために，古典的な evlis 方式を採用している．すなわち，引数を評価した結果を 1 本のリストとして関数に受け渡す．受け渡されたリストは，次節で述べる方法によって実装用メソッドに受け渡される．

3.1.3 関数呼出し

前述のように，JAKLD は，関数として組込み関数，ユーザ定義関数，継続の 3 種類をサポートしている．これらを実装する Java クラスの Subr，Lambda，Contin は，それぞれのインスタンスを関数として呼び出すためのインスタンス・メソッド

```
public Object invoke(List args)
```

をクラスごとに定義している．ここで args は，呼び出し時にインタープリタが生成した引数リストである．以下本節では，Subr クラスの invoke（Subr.invoke），つまり組込み関数の呼出し機構について解説する．

Java のメソッドとして実装された組込み関数を，Lisp 処理系から呼び出すために，Java の提供する reflection 機能を利用している．前節で述べたように，組込み関数の初期化には，def を使う．

```
def(cs, mt, fn, nreq, nopt, auxp)
```

は，まず *cs* という名のクラスで定義されている *mt* という名の public メソッドを検索する．次に，Subr クラスのインスタンスを生成し，その中に，見つかったメソッド（Method クラスのインスタンス）と，def への残りの引数を格納する．この Subr オブジェクトが，*fn* という名の関数データを表現する．最後に，*fn* という名の記号を（もし存在していなければ）生成し，その値スロットに，生成した Subr オブジェクトを格納する．

Java の reflection 機能では，Method オブジェクトとして取り出したメソッド M を呼び出すためには，M が受け取る引数の個数と同じ長さの配列（以下，「引数配列」とよぶ）を用意し，実引数を順に格納して受け渡さなければならない．このように受け渡された実引数が，M の定義中の引数の型と整合するかどうかは，reflection 機能が自動的に検査する．この検査に合格してはじめて，メソッド M が呼び出される．

4 機能追加

前節で述べたように，JAKLD は IEEE 機能のうち，一級継続と mutable な文字列をサポートしていない．しかしこれらがなくても，SICP の演習をする上で問題はない．一級継続を生成する call/cc も，文字列中の文字を別の文字で置き換える string-set! も，SICP では使われていないからである．一方，末尾再帰の最適化については SICP で明確に説明があるために，SICP 演習用処理系には不可欠な機能である．

第 2 節で議論した「必要な機能」については，変数の true，false，nil は単に定義を追加すればよいし，関数の random は簡単に定義できる．define 式が記号 ok を返すようにするのも簡単である．したがって，SICP 演習用として JAKLD を拡張するための主要な作業は，末尾再帰の最適化とグラフィック機能の追加だけということになる．

4.1 末尾再帰の最適化

前述のように，JAKLD は Java の実行時スタックを直接操作できない．直接操作できなくても，Scheme コードを中間コードに変換し，それを専用のインタープリタで実行することによって末尾再帰の最適化を実現することはできる．しかしそのためには処理系を大幅に変更する必要がある．そこで今回は，トランポリンによる実装を採用した．ある関数 f から別の関数 g を末尾再帰的に呼び出す場合に，f の戻り値のかわりに g と実引数 $e_1, \ldots, e_n$ の情報を f の呼出し元に返し，f の呼出し元が g を呼び出す．これによって f のためのスタックフレームがポップされた状態で g を呼

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

図 6　変更した if の定義

び出すことができ，実質的に末尾再帰の最適化を行ったことになる．本格的な実装と比べると性能面で劣るが，演習用の処理系なので十分現実的な解である．

関数呼出しが末尾再帰であるかどうかを判断するために，特殊形式の定義には，それが末尾再帰位置で呼び出されたかどうかを示すフラグ `tailp` を追加の引数として受け渡すように変更した．例えば，図 5 の `if` 式の定義は図 6 のようになる．e_1 または e_2 を評価するときに，それらが末尾再帰位置にあるかどうかは，`if` 式自体が末尾再帰位置かどうかに依存する．そのために，これらの式の評価には，`Lif` が受け取った `tailp` を受け渡している．条件式 c を評価するときには，必ず評価結果が必要になるので，従来通りの評価を行う．`eval` はこのようにオーバーローディングされており，

```
eval(cond, env)
```

は，

```
eval(cond, env, false)
```

と等価である．

`eval` が関数呼出し式に遭遇すると，従来通り関数の特定と実引数の計算を行う．もし末尾再帰位置であれば，関数と実引数リストを格納した特殊なオブジェクト（Call オブジェクトと呼んでいる）を生成してリターンする．逆に末尾再帰位置でなければ，通常通り関数を呼び出し，戻り値が Call オブジェクトかどうかを調べる．Call オブジェクトでなければ通常通りリターンするが，Call オブジェクトであれば格納されている関数および実引数を取り出して再度呼出しを行う．この処理を，戻り値が Call オブジェクトでなくなるまで繰り返す．

実際の処理系では，トランポリンを必要とする箇所は `eval` だけでなく，“`=>`” 構文を伴った `cond` 式や，`map` や `call-with-input-file` などの高階関数などでも必要となる．JAKLD の場合は 10 数カ所あった．トランポリンのコードを 1 カ所にまとめるために，3 種類の関数（Subr，Lambda，Contin）を統一する `Function` をインターフェイスから抽象クラス（abstract class）に格上げし，トランポリンを必要とするすべての関数呼出しは `Function` で定義された `invoke` メソッドを呼び出すよう変更した．そして，`Subr`，`Lambda`，`Contin` の三つのクラスは，`Function` のサブクラスとした．この変更によって，`Contin` を例外クラスとすることができなくなったので，「継続を呼び出す」際には，Contin オブジェクト自体ではなく，特殊な例外オブジェクトを投げるよう修正した．

4.2　端末機割り込み

Scheme には，C 言語の for や while に相当する繰り返し構文がない．繰り返しは，末尾再帰呼出しを使って記述する．末尾再帰呼出しの最適化が行われないときは，繰り返しが無限に続くとスタックがオーバーフローするので，プログラムの実行は自動的に停止して，コントロールはトップレベルに戻る．このために，従来の JAKLD には無限ループを停止するための端末機割り込み（通常はキーボードからコントロール C を打つ）が備わっていなかった．しかし末尾再帰の最適化を行うようになったので，端末機割り込みが必要となった．

端末機割り込みの機能は，標準の Java 仕様には含まれていないが，調べたところ，多くの Java VM では，Unix と同様の割り込みハンドラが備わっており，その仕様も Sun Microsystems 社が非標準として提供しているもの（sun.misc.SignalHandler）に準拠しているようである．この機能を使って端末機割り込みを実装した．端末機割り込みのシグナルを処理するハンドラを設定しておき，割り込みがあったらハンドラが専用のフラグ（割り込みフラグ）を立てる．JAKLD のインタープリタがときどきこの割り込みフラグをチェックすることによって割り込みがあったことを知る．

基本的に無限ループを回避するための割り込みであり，無限ループは末尾再帰呼出しによって起きるので，割り込みフラグのチェックはトランポリンのところで行えばよい．トランポリンは処理系全体で 1 カ所に集約されているので，そこでチェックする．関数呼出し（末尾再帰であろうとなかろうと）のたびに割り込みフラグのチェックを行うことになるが，インタープリタ方式の実行なので，チェックの頻度はちょうどよさそうである．

4.3 グラフィックス

グラフィック機能をどこまでサポートするかは大きな問題であるが，今回は，TUTScheme/Tk を数年間使っていた経験から，同様のグラフィック機能を提供し，講義担当者や TA の意見を取り入れて改善していくことにした．提供するグラフィック機能は，全面的に Java のクラスライブラリを使い，SICP 演習に使い易いような API を JAKLD 側で用意した．ソースコードにして 300 行，11K バイト程度のコーディングとなった．

図形言語を使用するには，まず

```
(start-picture)
```

とする．これによって図形言語の初期化が行われて描画用のウインドウが開く．描画ウインドウのサイズなどは `start-picture` へのオプショナル引数で指定できる．学生は JAKLD と対話しながら描画用コマンドを投入し，その作用は直ちに描画ウインドウに反映される．

描画できるオブジェクトには，折れ線，多角形，画像の三種類がある．折れ線と多角形は，SICP にあるとおり頂点を指定することによって作成する．画像はファイルからデータを読み込むことによって作成する．オブジェクトはアフィン変換による変形をして描画ウインドウに描き出すことができる．描画ウインドウの内容は，いくつかの画像フォーマットでファイルとして保存することができる．Java にはこれらを実装するために必要なライブラリがそろっており，仕様が理解できれば比較的短期間で実装できる．

グラフィック機能の大部分を Java のライブラリに委ねているために，詳細な仕様は Java の仕様や実装に依存することになる．例えば，画像データとして読み込むことのできるフォーマットは Java VM が読み込むことができるものに限られ，それ以上でもそれ以下でもない．どのようなフォーマットを扱えるかは，Java VM やそれが動作する計算機環境に依存するので，JAKLD が扱えるフォーマットもそれに依存する．BMP，JPEG，PNG などはどのような環境でもほぼ使えるようであるが，正確なところは，各環境で試してみるしかない．

Java のグラフィック機能に精通している者にとっては，上記の機能の実装は自明なので，以下，実装についてごく簡単に紹介するにとどめる．図形言語が起動されると，描画内容を記憶するバッファを一つ生成する．実際の描画操作はこのバッファに対して行われ，一定時間間隔で描画ウインドウに複写される．複写専用のスレッドを用意し，それが定期的に起き上がって複写を行う．描画内容をファイルに描き出す場合は，バッファの内容をファイルに保存すればよい．

5 おわりに

SICP 演習の問題点は以前から気づいていたが，去る 7 月，前期の SICP 講義が終わるころに学生たちと雑談していて，教員側が把握（想像）していたよりはるかに学生が演習の環境で困っていることが分かった．しかし，演習用処理系の開発に多大な時間を費やす余裕がなかったので，何とか簡単にしかも効率よく開発できる方法はないものかと考え，Java で記述された処理系を改造することを思いついた．筆者は Java のグラフィック機能や割り込み機能など，今回の拡張機能についての知識がほとんど無かったが，関連する書籍を生協で買いあさったり，Web で資料収集することによって，なんとか当初の目的を達成するものが夏休み明けごろに完成した．1 年生後期の SICP 講義で使えるよう，メディアセンターにインストールし，筆者の Web ページからもダウンロードできるように公開している [19].

Java のライブラリを徹底的に利用することによって短時間で作業を終えることができたが，その反面，Java の仕様に制約を受けることになってしまった．これは仕方がないことではあるが，Java VM の動作が

計算機環境によって異なっていることがあり，処理系の使い方に微妙な相違が生じている．とはいえ，当初の目標であった，「同一仕様で多様な計算機環境で動作すること」はほぼ達成できたと自負している．

処理系のサイズについては，今回の拡張によって，クラスファイルが 3 個追加され，ソースコードは約 700 行，30K バイト増加した．また，Java コンパイラが生成するクラスファイルのサイズは 8K バイト増加した．増加したとはいえ，その量はわずかであり，依然としてコンパクトな処理系であることに変わりはない．処理系全体をまとめた jar ファイルはわずか 46K バイトである．

もともとの JAKLD もそうであったが，処理系のプログラムはきわめて簡単にできているので，今後，演習を通して改善すべき箇所があれば簡単に修正できると思われる．

謝辞

SICP 講義の前半を担当されている奥乃博先生，TUTScheme/Tk を開発された電気通信大学の小宮常康先生，湯淺研究室のスタッフおよび学生のみなさん，これらの方々には，現状認識，仕様策定，実装方法などについて有益なコメントをいただきました．ここに感謝の意を表します．

参 考 文 献

[1] Harold Abelson and Gerald Jay Sussman: *Structure and Interpretation of Computer Programs, 2nd Edition*, MIT Press, Cambridge, MA, 1996.
[2] William Clinger and Jonathan Rees: Revised^4 Report on the Algorithmic Language Scheme, MIT AI Memo 848, MIT, 1991.
[3] 小宮常康: TUTScheme/Tk の処理系, http://www.spa.is.uec.ac.jp/~komiya/download/
[4] *IEEE Standard for the Scheme Programming Language*, IEEE, 1991.
[5] Takayasu Ito and M. Matsui: *A Parallel Lisp Language PaiLisp and Its Kernel Specification*, Lecture Notes in Computer Science 441, pp.22–52, Springer-Verlag, 1995.
[6] R. Kelsey, W. Clinger, and J. Rees: Revised^5 Report on the Algorithmic Language Scheme, *Higher-Order and Symbolic Computation*, Vol.11, No.1 (1998).
[7] 京都大学工学部シラバス（情報学科）：アルゴリズムとデータ構造入門, http://syllabus.t.kyoto-u.ac.jp/syllabus/2008/91150.html.
[8] 京都大学工学部シラバス（情報学科）：プログラミング言語, http://syllabus.t.kyoto-u.ac.jp/syllabus/2008/90170.html
[9] The MIT Press: *Structure and Interpretation of Computer Programs, 2nd Edition*, http://mitpress.mit.edu/sicp/.
[10] MIT/GNU Scheme, http://www.gnu.org/software/mit-scheme/
[11] Michael Sperber, R. Kent Dybvig, Matthew Flatt, and Anton van Straaten: Revised^6 Report on the Algorithmic Language Scheme, http://www.r6rs.org/final/html/r6rs/r6rs.html, 2007.
[12] Guy L. Steele Jr.: *Common Lisp the Language*, Second Edition, Digital Press, 1990.
[13] 高野保真, 岩崎英哉: Improving Sequence を第一級の対象とする Scheme コンパイラ，第 8 回プログラミングおよびプログラミング言語ワークショップ論文集, pp.153–162, 2006.
[14] Tcl Developer Xchange, http://www.tcl.tk/
[15] 和田英一 訳: 計算機プログラムの構造と解釈　第二版, ピアソン・エデュケーション, 2000.
[16] 山本晃成，湯淺太一: 末尾再帰の最適化と一級継続を実現するための JVM の機能拡張, 情報処理学会論文誌, 42 巻 SIG 11(PRO 12) 号, pp.37-51, 2001.
[17] 湯淺研究室: TUTScheme, http://www.yuasa.kuis.kyoto-u.ac.jp/~komiya/tus_intro.html
[18] 湯淺太一：Java アプリケーション組込み用の Lisp ドライバ, 情報処理学会論文誌. vol.44, No.SIG 4 (PRO 17), pp.1-16, 2003.
[19] 湯淺太一：JAKLD ダウンロードページ, http://www.yuasa.kuis.kyoto-u.ac.jp/~yuasa/jakld/index-j.html, 2006.
[20] 湯淺太一，鵜川始陽: 携帯電話で使える i アプリすぷ, コンピュータソフトウェア, Vol.24, No.4, pp.109-122, 2007.