

携帯電話向けのマイグレーション可能な Scheme 言語処理系

松崎 泰裕[†], 並木 美太郎^{††}

[†] 東京農工大学大学院情報工学専攻 ^{††} 東京農工大学大学院共生科学技術研究院

本論文では、携帯電話の Java VM 上で動作する Scheme 処理系の開発について述べる。本処理系では携帯電話向け API を提供し、Scheme 言語での携帯電話向けアプリケーションの開発を実現する。また、Scheme 上のオブジェクトを携帯や計算機間でマイグレーションすることに対応する。処理方式にツリートランバース型インタプリタを採用して実装を行った結果、処理系のサイズは約 55 KB、実行時メモリは約 230 KB となった。今後の課題は、携帯電話のモバイル性を活かす分散処理やマイグレーションによる情報処理などの応用システム開発である。

Scheme Interpreter for Cellular Phones with Migration

Yasuhiro MATSUZAKI[†] Mitaro NAMIKI^{††}

[†] Dept. of Computer and Information Sciences, Tokyo University of Agriculture and Technology

^{††} Graduate school of Engineering, Tokyo University of Agriculture and Technology

This paper describes development of a Scheme interpreter running on Java VM for cellular phones. The interpreter provides API for GUI and network functions on cellular phones and makes mobile programmers to develop programs using Scheme on cellular phones. The interpreter also enables the migration of Scheme objects to another phones. The interpretation and execution of scheme programs are traversing the program tree of Scheme codes. The size of this interpreter is 55 KB and the amount of using memory is 230 KB. Application systems such as distributed processing or information system with the migration of continuations are prospected.

1 はじめに

近年の携帯電話は Java VM を搭載しており、プログラマーは携帯電話上で動作するアプリケーション (以下 AP) を自由に開発できる。しかし、使用できる言語は Java に限られている。このことは Java 言語以外のプログラマーに対して携帯電話向け AP 開発の敷居を高めていると言える。また、既存 AP を携帯電話へ移植する際にはプログラミング言語とプロファイルを越えた移植が必要となり、多大な手間が生じている。また同じ Java 言語であっても、キャリアによってプロファイルが異なり、AP は各キャリアに依存するものとなっている。

さらに携帯電話の普及により携帯端末は一人一台持つものとなっているが、それら複数の端末に存在する情報やハードウェア資源を利用するシステムの開発においては、状況に応じてサーバーや携帯端末間でマイグレーションを行う方式での開発が挙げられるが、携帯電話の Java 言語ではマイグレーションへの対応がされていない。

これらの問題を解消するために、本研究では携帯電話の Java VM 上で動作するマイグレーションに

対応した Scheme 言語処理系の設計と実装を行う。

2 既存の言語処理系と問題点

携帯電話向け AP を Java 言語以外で開発するために、携帯電話の Java VM 上で動作する言語処理系がいくつか存在する。また、PC の Java VM 上で動作する言語処理系も存在する。以下に、既存の言語処理系をいくつか紹介し、その問題点を挙げる。

2.1 soyBasic

soyBasic¹⁾ は、NTT DoCoMo の i アプリとして動作する BASIC 環境である。処理対象の言語として BASIC のサブセット言語を採用しており、対話実行や行番号による実行をサポートする。携帯電話上での入力を重点に考えられており、編集状況や文脈などから自動的に識別子の候補を表示し、コード補完を行う。

この処理系は、携帯電話向け AP の開発言語として BASIC を提供する点では優れていると言える。しかし携帯電話の通信機能の利用などには対応しておらず、マイグレーションについても非対応である。提供されている処理系のプロファイルは、Web アプレットと i アプリ (DoJa) のみである。

2.2 JAKLD

JAKLD²⁾は、Java アプリケーション組み込み用の Lisp ドライバである。Java アプリケーションへの組み込みを想定して開発されているが、単体の Scheme 処理系としても利用可能で、NTT DoCoMo の i アプリ版も公開されている。処理対象の言語として IEEE Scheme を採用しているが、準拠していない点として次の3点が挙げられる。

- 継続は、脱出にのみ利用できる
- 末尾呼び出しの最適化は行われない
- 文字列は immutable であり、既存文字列の変更ができない

この処理系の問題点として、まず携帯電話向け API が提供されていない点が挙げられる。携帯電話の特徴である液晶画面へのグラフィック表示やネットワークへのアクセスといった機能にアクセスする API が提供されておらず、そのような機能を用いた AP を開発することができない。通信機能が提供されていないため、他の端末へのマイグレーションも不可能である。

別の問題点として、継続の使用の制限や、末尾呼び出しの最適化に対応していない点も挙げられる。Scheme は数少ない継続を扱える言語の一つであり、既存のプログラムで継続を使用しているものも多い。また Scheme は関数型言語であり、末尾呼び出しの最適化を前提とした末尾再起が積極的に使われるため、末尾呼び出しの最適化は必須と言える。

3 本研究の目標

前章までに述べたとおり、携帯電話向け AP の開発言語は主に Java 言語に限られており、他の言語の処理系も存在するものの、言語上の制限や、携帯電話向けの考慮がされていないなどの問題点がある。

この問題を解決するために本研究では JavaVM 上で動作する Scheme 言語処理系を開発する。本処理系により Scheme 言語を用いて携帯電話向け AP を開発する環境を提供し、既存の Scheme プログラム資源を有効活用する。Scheme の言語仕様は R5RS³⁾に準拠する。本処理系では多くの携帯電話で動作することを指すため、一部の携帯電話で対応していない実数演算とファイル出力には対応しない。

本処理系は、現在の多くの携帯電話で採用されている DoJa プロファイルおよび MIDP、そして Java 2 Second Edition を搭載した一般の計算機において動作するものとする。また、携帯電話上の Java VM で

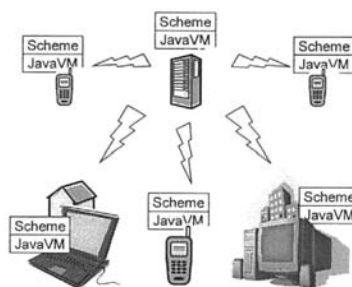


図 1 マイグレーションイメージ図

は JAR パッケージのサイズが制限されており、DoJa-4.0 プロファイルでは 30~100 KB⁴⁾、MIDP を採用する S!アプリでは 50~256 KB⁵⁾と制限が厳しい。したがって本処理系の JAR パッケージに処理するプログラムを格納する場合を考慮し、本処理系の動作に不可欠な核部分のサイズは 60 KB 以下を目標とする。

本処理系では、携帯電話の液晶画面やネットワーク機能を扱うための手続きを提供し、GUI、グラフィックやネットワークを用いたプログラムを Scheme によってプログラミング可能とする。また、継続を含めた Scheme オブジェクトのマイグレーションに対応し、携帯電話のモバイル性を活かした分散処理や、ネットワークを通じた携帯電話や計算機間のマイグレーションをするモバイルエージェントシステム (図 1 参照) などに応用する。

4 本処理系の概要

本処理系の全体構成は図 2 のとおりである。本処理系では、Scheme プログラムをあらかじめ計算機上で作成して本処理系の JAR パッケージへ格納して実行する方式の他に、携帯電話上で Scheme プログラムを入力し、そのまま実行する方式をサポートする。ただし、今回の実装は図 2 の点線部とし、実行する Scheme プログラムは JAR パッケージへ格納しておくものとする。携帯電話上での Scheme プログラムの入力や対話のためのエディタは今後の課題とする。

Scheme 言語処理系において Scheme プログラムを処理する方式には表 1 に挙げる 3 通りがある。現在の携帯電話向け Java VM では、動的に生成した Java バイトコードを実行することができないため、ネイティブコンパイル方式は使用できない。本研究では携帯電話の Java VM のサイズ制限を考慮し、処理系のサイズが小さくできるツリートラバース型イ

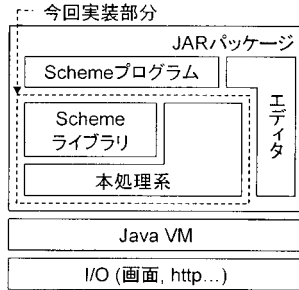


図 2 本処理系の全体構成

表 1 Scheme プログラムの処理方式

方式	実装	処理速度	サイズ
ネイティブコンパイル	複雑	高速	大きい
Virtual Machine	複雑	中速	大きい
ツリートランスバース	単純	低速	小さい

インタプリタ方式を採用する。

また本処理系では携帯電話の Java VM のサイズ制限を考慮し、本処理系ではガベージコレクションの処理は記述せず、Java VM によるガベージコレクションを利用する。

5 評価器の設計

本章では、インタプリタの核である評価器の設計について述べる。

5.1 データ表現

Scheme のデータ構造を Java で保持するためには、S 式を表わすクラスを定義し、そのクラスを基底として拡張してドット対を表わすクラスや各アトムを表わすクラスを定義する必要がある。各アトムの区別には Java の instanceof 演算子を用いて、クラスの識別を行って区別する。

本処理系におけるデータ表現クラスは図 3 のとおりである。本処理系ではサイズを削減するために、Java の組み込みクラスである Object クラスで S 式を表わす。Object クラスはすべての Java クラスの基底であり、この Object クラスにより S 式の構造を表現することが可能となる。また、各アトムの Java での表現・保持方法には、できるかぎり Java の標準クラスを用いる。

真偽値、文字、数値は、それぞれ Java 標準のプリミティブ型の配列で保持する。配列としているのは S 式を表わす Object 型と部分型関係とするためである。Java には標準クラスとして各プリミティブ型を保持するためのクラスが存在するが、インスタンス

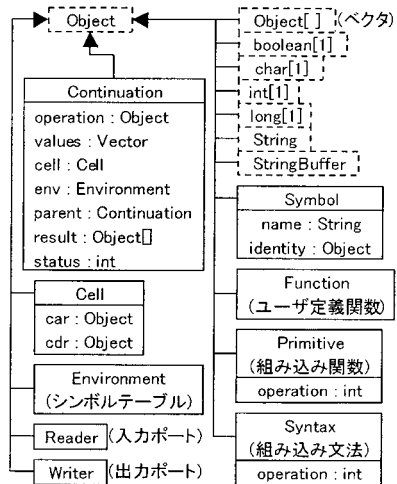


図 3 データ表現クラス

の生成やアクセスは配列の方が高速である。

文字列は、Java の標準クラスである String クラスと StringBuffer クラスで表わす。Java で文字列を扱う String クラスは、一度インスタンスを生成すると書き換えが不可能である。書き換えが必要な場合にはインスタンスを生成し直す必要があり、速度面で大幅な低下が発生する。そこで、Java において文字列を生成する途中に使われる StringBuffer クラスを用いて、書き換え可能文字列を表わす。

5.2 メインループ

ある式の評価中に別の式の評価が必要となった場合、再帰呼び出しを用いると処理される側のプログラムのコンテキスト情報がインタプリタのスタックに退避され、インタプリタを単純な構造にすることができる。Scheme には、継続をファーストクラスオブジェクトとして扱う機能があり、任意のタイミングでのコンテキスト情報をコピーや復元が必要となる。Java VM ではこのコンテキスト情報が格納されているスタックにアクセスすることは許されおらず、インタプリタのメインループにおいて再帰呼び出しを用いることはできない。

この問題に対応するために、再帰呼び出しを用いずにループのみで処理をし、処理される側のプログラムのコンテキスト情報をインタプリタのスタックではなくヒープ領域に確保してインタプリタ自身で管理する。本処理系では Continuation クラスを定義し、そのインスタンスで各式の評価におけるコンテキスト情報を保持する。このクラスにより Continuation

クラスで保持するコンテキスト情報は次のとおりである。

- 命令
行うべき処理を表わす命令である。S 式の先頭を評価して得た、構文や手続きが格納される。
- 評価済み要素の結果
- 未評価残り要素
- 環境
- 次の継続
現在の評価が終わった後に処理すべき継続、つまり呼び出し元への参照を保持する。

5.3 継続

継続は、前節で述べたコンテキスト情報である。本処理系では Continuation クラスのインスタンスを生成し、前節で述べた各コンテキスト情報を保持する。

Scheme のプログラムによって継続の取得が要求された場合には、この Continuation クラスのインスタンスをコピーして別のインスタンスを生成し、それを評価結果とする。ここでコピーが必要となるのは、要求の後に実行される継続と後で使われる継続の干渉を防ぐためである。

5.4 シンボルテーブル

Scheme の変数束縛は、シンボルに対して値を束縛するという形をとる。トップレベルでは、同じシンボル名であれば値は同じ場所へと格納される。ラムダ式により局所的な変数束縛が行われた場合は、束縛されたシンボルに対する値は新しい環境へと格納され、それ以外は元の環境が参照される。

この束縛を実現するために、個々の環境を表わす Environment クラスを定義し、親となる環境への参照を保持するようにする。各 Environment ではハッシュテーブルを持ち、シンボルに対する値を保持する。シンボルに対する値を取得する場合は、一番末端の子となる環境でまずハッシュテーブル走査を行い、見つからなければ親の環境を順に参照する形となる。

一般的にはシンボル名で走査をし、見つかった一番内側の値をその束縛の値とすれば良い。しかし Scheme ではマクロを使用でき、マクロを使用した場合には同じシンボル名でより外側の値を要求される場合がある。

この問題に対応するために、ハッシュテーブルの走査に用いるキーには、シンボル名ではなく、名前

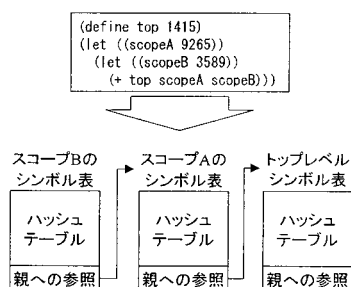


図 4 シンボルテーブル

と別に持つ識別オブジェクトを用いる。この方法により、同じ名前であっても、内側と外側の環境を区別して走査することが可能となる。

5.5 組み込み構文と手続き

Scheme の仕様書で定義されている構文や手続きは、評価器のコード中に組み込み構文や組み込み手続きとして実現する。

構文は Syntax クラス、手続きは Primitive クラスを定義し、メンバとして各構文や手続きを識別するための数値を持つ。そして各構文や手続きに応じてユニークな数値を保持したインスタンスを作り、その構文の名前であるシンボルに束縛しておく。評価時にはその数値を用いて switch 文で分岐を行い、各処理を行う。

5.6 ユーザ定義手続き

Scheme では図 5 のように引数と本体を指定した手続きをプログラム中で定義することができる。

評価器では、手続きを作り出す構文が評価された時に、Function クラスのインスタンスを生成し、それを評価結果とする。Function クラスでは、次の情報を保持する。

- 固定長の各引数を束縛するシンボルのリスト
- 可変長の残りの引数を束縛するシンボル
- 関数本体
- 環境
- 手続き名

6 マイグレーション

本処理系では、Scheme 上の任意のオブジェクトのマイグレーションに対応する。この機能ではデータの転送を容易にすると共に、継続をネットワークを通して他の携帯電話や計算機へ転送し、転送した先でその継続を起動することを実現する。

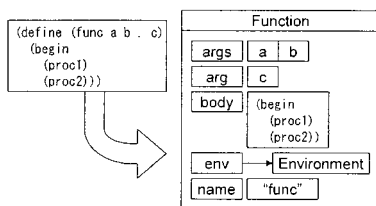


図 5 ユーザ定義手続き

Scheme 上でマイグレーションを行う場合は、まずマイグレーション元の端末で http 通信を用いてサーバーへオブジェクトを送信し、次にマイグレーション先の端末で http 通信を用いてオブジェクトを受信する。現在の携帯電話では、携帯電話間で直接通信することができないため、このようにサーバーを中継する必要がある。

オブジェクトを http 通信を用いて転送する際には、オブジェクトを http 通信で扱えるバイト列に変換する必要がある。本処理系では、基本的にはオブジェクトの各メンバを再帰的にシリアル化することによってマイグレーションを実現する。循環オブジェクトに対応するために、シリアル化の際には入出力順にオブジェクトに番号を付け、既出のオブジェクトはその番号で参照する。以下に各オブジェクトのマイグレーション対応について述べる。

6.1 グラフィックス描画

画面関係の I/O のマイグレーションには、次の二通りがある。一つは入出力処理のみをマイグレーションし、実際の表示や入力はいマイグレーション元で行う方法である。もう一つは、表示や入力に関してもマイグレーション先で行う方法である。

現在の携帯電話の通信は自発的にしか行えず、前者の方法を実現するためにはマイグレーション元で定期的に描画要求をポーリングをする必要がある。定期的なポーリングによる通信料の増大や通信の遅延を考え、本研究では後者の方法を採用する。マイグレーションが要求された場合には、現在表示されている内容をシリアル化して転送し、マイグレーション先で表示することに対応する。

グラフィックス描画で使われるキャンパスは、任意の図形の描画が可能であり、その実体はビットマップである。この内容をマイグレーションするには、画面上の各点に表示されている色を転送する必要がある。しかしそのデータ量は膨大であるため、転送時には圧縮をする必要がある。そこで DoJa では転送時に JPEG 形式で圧縮をし、マイグレーション先

でそれを展開して描画することで対応する。しかし MIDP には JPEG 形式で圧縮するための機能が無いため、MIDP で JPEG への対応は保留とし、非圧縮で送ることとした。

6.2 GUI コンポーネント

GUI コンポーネントのマイグレーションに関しても、グラフィックス描画と同様にマイグレーション先で表示するものとする。

マイグレーションが要求された際には、パネル上に追加されている各 GUI コンポーネントの一覧を取得し、コンポーネントの種類とそのコンポーネントに入力されている内容をシリアル化する。但し、DoJa ではパネル上のコンポーネント一覧を取得できないため、Scheme プログラムで GUI コンポーネントの追加処理を行う際に一覧のコピーを作成しておき、マイグレーション時にはこの一覧を参照する。

6.3 HTTP 通信

現在の携帯電話における HTTP 通信には大きな制限があり、接続中に送受信するデータを操作することができない。送信するデータは接続する前にすべてを設定する必要があり、接続をするとあらかじめ指定しておいたデータが送信される。受信データは接続中に順次処理することはできず、すべてのデータを受信し終わって接続が終了してから処理が可能となる。また、HTTP 通信は同時に複数行うことはできない。これらの制限により、本処理系で HTTP 通信オブジェクトのマイグレーションが発生しうるタイミングは接続を開始する前と接続が終了した後の二つに分かれる。

接続が開始される前にマイグレーションが要求された場合には、接続すべき URL と設定済みの送信すべきデータをシリアル化して転送する。マイグレーション先では新たに ByteArrayOutputStream を作成してそのストリームへ設定済みのデータを書き込み、Scheme プログラムが続きを出力できるようにする。

接続が終了した後にマイグレーションが要求された場合には、受信したデータのうち、未だ処理がされてないデータをストリームからすべて読み出して転送する。マイグレーション先では転送されたデータを元に ByteArrayInputStream を作成し、Scheme プログラムが続きを読み込めるようにする。

7 API の設計

携帯電話の機能を操作するための API は、Java で提供されるものと同様のものを提供する。ただし

Java と違い、Scheme はオブジェクト指向ではないため、作成されたオブジェクトを引数に渡す形で各オブジェクトに対する操作を行う。また、各 Java プロファイルの差異を本処理系で吸収し、Scheme プログラムに対してはプロファイルに依存しない共通の API を提供する。

7.1 GUI コンポーネント

GUI コンポーネントについては、DoJa および MIDP 両者で共通して提供されるものを本処理系では提供する。GUI コンポーネントを扱うための API は、表 2 のとおりである。

まず最初に Panel を作成し、その上に TextField などのアイテムを配置する。作成した Panel や、後に述べる Canvas などは、show 手続きで実際に画面に表示する。バックグラウンドで作成した複数の Panel や Canvas を切り替えながら表示することも可能である。Panel や Canvas には画面下部に表示されるソフトキーを設定可能である。ソフトキーが押下されると、setKeyPressed 手続きで指定しておいた Scheme の手続きが起動される。

7.2 グラフィック

グラフィック描画を行う API は、DoJa および MIDP における Canvas, Image, Graphics クラスにアクセスする API とする。グラフィック描画のための API は、表 2 のとおりである。

画面へ描画を行うには、まず描画キャンバスとなる Canvas を作成し、その Canvas の Graphics を取得して描画を行うか、または描画処理を記述した手続きを Canvas-setPaint で描画要求時に起動されるように設定する。

7.3 HTTP 通信・マイグレーション

HTTP 通信を行う API は、DoJa および MIDP における HttpURLConnection インターフェースにアクセスする API とする。HTTP 通信のための API は、表 2 のとおりである。

HTTP 通信を行うには、まず Connector-open 手続きに URL を指定して http 接続オブジェクトを作成する。次にそのオブジェクトを用いてメソッドや送信するデータを指定し、http-connect 手続きを呼ぶことで実際に通信が行われる。http-connect には通信が終了した時に起動されるべき手続きを指定する。通信が終了したら、http-getInputPort で取得した入力ポートから受信データを取得する。

本処理系では通信の終了を手続きの起動で通知することにより、通信中に処理がブロックされること

表 2 API 一覧

手続き名	説明
show	画面に Panel, Canvas を表示
setSoftLabel	ソフトキーを設定
make-Panel	Panel を作成
Panel-add	Panel に Item 追加
Panel-setKey...	キー押下時の手続きを指定
make-TextField	TextField 作成
TextField-getText	文字列を取得
TextField-setText	文字列を設定
make-Canvas	Canvas を作成
Canvas-getGraphics	Graphics を取得
Canvas-setPaint	描画時に起動する手続きを指定
Canvas-setKey...	キー押下時の手続きを指定
Canvas-repaint	再描画を要求
Image-createImage	Image を作成
Image-getGraphics	Graphics を取得
Graphics-drawImage	Image を描画
Graphics-drawLine	線を描画
Graphics-setColor	描画色を指定
Connector-open	URL から http 接続を作成
http-setRequestMethod	GET/POST メソッドを指定
http-getOutputPort	送信データを設定
http-connect	通信終了時に起動する手続きを指定して、実際に接続する
http-getInputPort	受信データを取得
read-object	入力からオブジェクト復元
write-object	出力ポートへシリアルライズ

なく、他の処理を行うことができる。また Scheme で定義される手続きには、ユーザ定義関数の他に継続も含まれており、本処理系ではコールバック関数の起動だけではなく、継続の切り替えをもって通信の終了時の処理を行うことができる。

Scheme 上のオブジェクトのマイグレーションを行う場合には、HTTP 通信 API で取得した人出力ポートに対して read-object 手続き、または write-object 手続きを用いてオブジェクトのシリアルライズを行う。

8 実装と評価

8.1 処理系のサイズ

実装を行った結果の、本処理系の最終的な JAR パッケージのサイズを確認し、他の処理系と比較した。結果は表 3 のとおりとなった。

本処理系のサイズは Scheme で書かれたライブラリなどを含めた JAR パッケージで約 55 KB となった。これは、一般的な携帯電話のサイズ制限である 100 KB⁴⁾ の約半分であり、処理する Scheme プログラムを JAR パッケージに埋め込む十分な余地が残っている。また JAKLD と比較して、約 15 KB の増加となった。これは携帯電話向け API を追加した上でのサイズであり、携帯電話向け API を省いた純粋な Scheme としての本処理系は約 45 KB である。この

表 3 処理系のサイズ

対象環境	本処理系	JAKLD
DoJa	55 KB	40 KB
MIDP	55 KB	-
PC	45 KB	36 KB

```
(let ((var 123))
  (call/cc
    (lambda (c)
      (define http
        (Connector-open "http://hoge/?put"))
      (http-setRequestMethod http "POST")
      (define out (http-getOutputPort http))
      (write-object c out)
      (close-output-port out)
      (http-connect http)
      (http-close http)))
    var)
```

図 6 マイグレーション送信プログラム

結果より、継続への対応による処理系サイズへの影響は小さいと言える。本処理系では継続への対応をした上でサイズを既存処理系と同等に抑えることができ、また携帯電話向け API を追加した上で約 15 KB の増加に抑えることができた。

8.2 マイグレーションの評価

本処理系の応用の可能性を確かめるために、継続のマイグレーションの動作を確認した。確認に使用したプログラムは図 6、図 7 のとおりである。図 6 のプログラムは、call/cc により捕獲した継続を HTTP 通信によりサーバーへ送信する。サーバーは受信した継続を一時的にファイルへ保存する。図 7 のプログラムは、HTTP 通信により保存された継続をサーバーから受信し、その継続を起動する。マイグレーションされる継続は、var という変数の値を表示するという処理である。

A 社の携帯電話端末において図 6 のプログラムを実行し、その後 B 社の携帯電話端末において図 7 のプログラムを実行したところ、var の値が表示され、マイグレーションが正しく行われたことが確認できた。

本処理系のこの機能により、携帯電話間をマイグレーションする情報処理などの応用システム開発が期待できる。また本処理系は PC 上の Java VM でも動作するため、携帯電話間に限らず、サーバーや一般の PC などへのマイグレーションも期待できる。

8.3 実行時メモリの評価

携帯電話 2 台、および PC 上で本処理系を動作させ、メモリ使用量を測定した。測定直前に Scheme 上から Java の GC を起動し、その後に全メモリ容量

```
(define http
  (Connector-open "http://hoge/?get"))
(http-setRequestMethod http "GET")
(http-connect http)
(define in (http-getInputPort http))
(define cont (read-object in))
(close-input-port in)
(http-close http)
(cont)
```

図 7 マイグレーション受信プログラム

表 4 メモリ使用量

環境	起動時	リスト作成時
A 社携帯	233 KB	21 KB
B 社携帯	221 KB	21 KB
C 社携帯	232 KB	21 KB
PC	280 KB	16 KB

と使用量を取得することでメモリ使用量を算出した。JAKLD などの他の処理系については、メモリ使用量を取得する API がいないため測定を行っていない。

結果は表 4 のとおりとなった。リスト作成時とは、car 部に空リストを格納した長さ 1000 のリストを作成した時の消費メモリである。本処理系の消費メモリは起動時で約 230 KB、リスト作成時で 20 KB 程度となっており、これは数 MB⁶⁾ のヒープメモリを搭載している最近の携帯電話において問題ない使用量である。

8.4 処理速度の評価

携帯電話 2 台、および 3GHz の CPU を搭載した PC 上で実行速度を測定した。比較のために、B 社の携帯電話においては JAKLD²⁾ についても測定を行った。また PC 上では SISC⁷⁾ と Kawa⁸⁾ についても測定を行った。比較に用いたコードは、Gabriel ベンチマーク⁹⁾ およびループ速度を計測するための図 8 に示すコードである。ただし、Gabriel ベンチマークは PC 向けであり、携帯電話上では現実的な時間で処理できないため、携帯電話上では tak のみを測定した。また JAKLD と Kawa は継続と末尾呼び出しの最適化に対応していないため、一部の動作しないコードは測定できていない。

結果は表 5、表 6 のとおりとなった。ループは 1 ループあたりの時間である。本処理系は、携帯電話向けとして問題ない速度を確保できている。B 社携帯電話上では一部のコードで JAKLD より遅い結果が見られるが、PC 上では同等、または逆に速い結果が見られる。これは携帯電話の Java VM の性能によるものと考えられる。

PC 上においては、JAKLD と同等の結果が得られた。本処理系では継続や末尾呼び出しの最適化に対

```

(define (loop-only n)
  (if (> n 0) (func (- n 1)))))
(define (func0) #t)
(define (loop-func n)
  (func0)
  (if (> n 0) (func (- n 1)))))
(define (loop-cont n)
  (define i n)
  (define con #f)
  (call/cc (lambda (c) (set! con c)))
  (set! i (- i 1))
  (if (> i 0) (con)))
(define (loop-do n)
  (do ((i n (- i 1)))
      ((= i 0))))

```

図 8 ループプログラム

表 5 携帯電話での実行速度

コード	A 社携帯	B 社携帯		C 社携帯
	本処理系	本処理系	JAKLD	本処理系
tak	316.2s	189.2s	26.89s	221s
loop-only	2.170ms	746 μ s	-	1.230ms
loop-func	3.287ms	1.468ms	-	1.961ms
loop-cont	2.250ms	764 μ s	-	1.366ms
loop-do	2.431ms	849 μ s	76 μ s	1.324ms

応した上で同等の速度を実現できたと言える。SISC や Kawa と比較すると速度の差が見られるが、これは Scheme プログラムの処理方式の違いによるものと考えられる。しかし継続を用いたループにおいては SISC よりも多少速い結果が得られており、ツリートラバース方式によって Virtual Machine 方式と同等の処理速度を実現できている。コンテキストのコピーや復元が頻繁に発生する継続ループでは Virtual Machine 方式での利点である最適化が難しく、コンテキストのコピーや復元という処理のみにおいては二つの処理方式で大きな差は無いためである。

9 おわりに

本論文では、携帯電話向け Scheme 言語処理系の設計と実装について述べた。本処理系により携帯電話上での Scheme プログラムの実行が可能となった。本処理系の互換性により既存の Scheme プログラムの活用が期待される。また継続などの Scheme 上のオブジェクトのマイグレーションに対応したことにより、携帯端末間や計算機間でマイグレーションを行う AP の開発が可能となった。

今後の課題は、携帯電話における二次記憶であるスクラッチパッドやレコードストアへの対応、携帯電話上で Scheme プログラムを入力するための実用的なエディタの開発、そして本処理系によって可能となった継続のマイグレーションを用いた情報処理や分散処理などの応用システムの開発である。

表 6 PC 上での実行速度

コード	本処理系	JAKLD	SISC	Kawa
cpstak	786ms	-	101ms	-
ctak	1.601s	13.56s	898ms	3.380s
deriv	1.905s	1.189s	333ms	69ms
destruct	25.64s	24.76s	3.766s	2.031s
div(ite)	12.92s	8.360s	1.188s	328ms
div(rec)	11.82s	10.69s	1.625s	344ms
earley	3.568s	-	1.141s	105ms
fib	12.15s	17.50s	3.516s	1.313s
hanoi	5.724s	7.563s	1.214s	311ms
nqueens	491ms	534ms	63ms	34ms
puzzle	8.812s	10.05s	1.625s	563ms
tak	406ms	491ms	80ms	16ms
takl	3.687s	4.89s	714ms	22ms
takr	436ms	502ms	91ms	70ms
travinit	10.11s	8.204s	2.640s	469ms
travrun	42.95s	38.59s	10.19s	890ms
loop-only	3.739 μ s	-	0.890 μ s	-
loop-func	4.890 μ s	-	1.390 μ s	-
loop-cont	4.818 μ s	-	6.500 μ s	-
loop-do	4.510 μ s	3.812 μ s	0.875 μ s	0.359 μ s

参考文献

- 1) 長慎也: soyBasic, <http://basic.kake.info.waseda.ac.jp/> (2005).
- 2) 湯浅太一: Java アプリケーション組み込み用の Lisp ドライバ, 情報処理学会論文誌, Vol.44, No.SIG4, pp.1-16 (2003).
- 3) R. Kelsey, W. Clinger and J. Rees, editors: The revised5 report on the algorithmic language Scheme. In Higher-Order and Symbolic Computation 11(1), pp. 7-105 (1998).
- 4) NTT DoCoMo: i アプリコンテンツ開発ガイド for DoJa-4.x, <http://www.nttdocomo.co.jp/> (2006)
- 5) ソフトバンクモバイル: S!アプリ開発ガイド MIDP 1.0 対応端末編, <http://developers.softbankmobile.co.jp/> (2006).
- 6) NTT DoCoMo: i アプリ対応端末の情報, <http://www.nttdocomo.co.jp/> (2006).
- 7) Scott G. Miller and Matthias Radestock: SISC for Seasoned Schemers, <http://sisc.sourceforge.net/> (2006).
- 8) Per Bothner: The Kawa language framework, <http://www.gnu.org/software/kawa/> (2006).
- 9) Richard P. Gabriel: Performance and Evaluation of Lisp System, MIT Press in Computer Science, MIT Press, Cambridge, MA (1985).