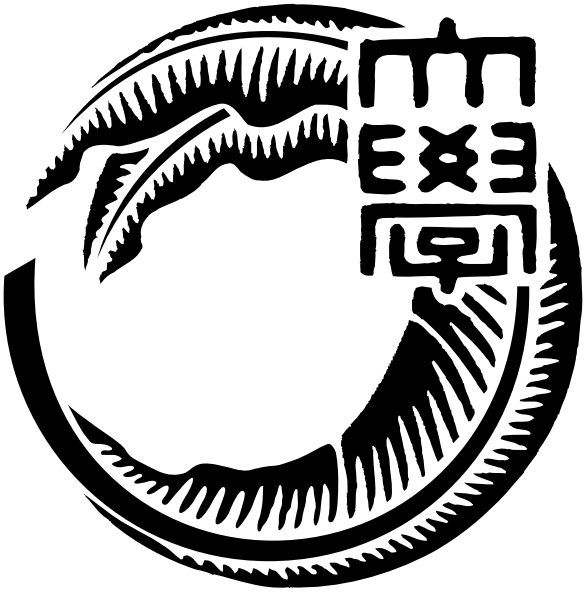
平成30年度 卒業論文

CbC による Perl6処理系



琉球大学工学部情報工学科

155730B 清水 隆博

指導教員 河野 真治

目 次

図 目 次

コード目次

第1章　現在のPerl6処理系

現在開発が進んでいるプログラミング言語に Perl6 がある。Perl6 は設計と実装が分離しており、 現在の主要な実装は Rakudo と呼ばれている。Rakudo は Perl6 のサブセットである、NQP と呼ばれる言語を中心に、 記述されている。NQP 自体はプロセス仮想機械と呼ばれる、 言語処理系の仮想機械で実行される。Rakudo の場合実行する仮想機械は、Perl6 専用の処理系である MoarVM、 Java 環境の JVM が選択可能である。

Rakudo はインタプリタの起動時間及び、 全体的な処理時間が他のスクリプト言語と比較して、 非常に低速である。また、 実行環境である MoarVM の実装事態も複雑であり、巨大な case 文が利用されているなど、 見通しが悪くなっている。

当研究室で開発しているプログラミング言語に、 Continuation Based C (CbC) がある。CbC は C と互換性のある言語であり、 関数より細かな単位である、 CodeGear を用いて記述することが可能となる。CbC では各 CodeGear 間の移動に、 環境などを保存せず次の状態に移動する軽量継続を用いている。軽量継続を用いる事が可能である為、 C 言語におけるループや関数呼び出しを排除する事が可能となる。

現在までの CbC を用いた研究においては、 CbC の言語処理系への応用例が少ない。スクリプト言語処理系では、 バイトコードから実行するべき命令のディスパッチの際に switch 分や gcc 拡張のラベル goto などを利用している。これらは通常巨大な switch-case 文となり、特定の C ファイルに記述せざるを得なくなる。CbC の場合、 この case 文相当の CondeGear を生成する事が可能である為、 スクリプト言語処理系の記述に適していると考えられる。

MoarVM は C 言語で記述されており、 C と互換性のある言語であれば拡張する事が可能となる。CbC は C と互換性のある言語である為、 MoarVM の一部記述を CbC で書き換える事が可能となる。

CbC[?] はこの Code Gear と Data Gear を単位を用いたプログラミング言語として開発している。

CbC は軽量継続による遷移を行うので、継続前の Code Gear に戻ることはなく、状態遷移ベースのプログラミングに適している。

また、当研究室で開発している Gears OS[?] は Code Gear、 Data Gear の単位を用いて開発されており、CbC で記述されている。

本研究では CbC を用いて Perl6 の実行環境である、 MoarVM の改良を行う。

第2章　Perl6

2.1　Perl6の概要

Perl6 は現在開発が勧められているプログラミング言語である。スクリプト言語 Perl5 の次期バージョンとして当初は開発されていたが、 現在では互換性の無さなどから別言語として開発されている。

Perl6 は仕様と実装が分離されており、 現在はテストスイートである Roast が仕様となっている。実装は歴史的に様々なものが開発されており、 Haskell で実装された Pugs、Python との共同実行環境を目指した Parrot などが存在する。Pugs や Parrot は現在は歴史的な実装となっており、 開発は行われていない。現在の主要な実装である Rakudo は、 Parrot と入れ替わる形で実装が進んでいる。Rakudo は Parrot 時代に考案された、NQP(NotQuitPerl) を用いて Perl6 を実装し、 NQP は VM によって評価される。Rakudo の VM は Perl6 専用の VM である MoarVM、 Java 実行環境である JVM、 Javascript が選択可能である。JVM、 Javascript の動作環境は MoarVM と比較して実装されていない処理などが多数あり、 現在は MoarVM が主流な VM として利用されている。MoarVM は C 言語で実装されており、 NQP は NQP 自身で記述されている。

2.2　NQP

NQP とは Rakudo における Perl6 の実装に利用されているプログラミング言語である。NQP 自体は、 Perl6 のサブセットとして開発されている。基本文法などは Perl6 に準拠しているが、 変数を束縛で宣言する。インクリメント演算子が一部利用できない。Perl6 に存在する関数などが一部利用できないなどの制約が存在する。

実際に NQP で記述したコードを に示す。

Perl6 は NQP で記述されている為、 Perl6 における VM は NQP の実行を目標として開発されている。NQP 自体も NQP で記述されており、 NQP のビルドには予め用意された MoarVM などの VM バイトコードによる NQP インタプリタが必要となる。MoarVM を利用する場合、 MoarVM の実行バイナリである moar に対して、 ライブラリパスなどを予め用意した NQP インタプリタのバイトコードに設定する。設定はオプションで与える事が可能であり、 moar を実行することで NQP のインタプリタが起動する。NQP のビルドには、 この NQP インタプリタをまず利用し、 NQP 自体のソースコードを入力して与え、 ターゲットとなる VM のバイトコードを生成する。このバイトコードは NQP ソー

スコードから生成されたNQPインタプリタのバイトコードであり、次にこのバイトコードをライブラリとしてmoarに与え、再びNQPをビルドする。2度ビルドする事により、ソースコードからビルドされたVMバイトコードでNQP自身をビルドする事が出来たため、処理系自身をその処理系でビルドするセルフビルドが達成出来たことになる。2度目のビルドの際に生成されたNQPインタプリタの事を小文字のnqpと呼び、これがNQPのインタプリタのコマンドとなる。

2.3 Rakudo

RakudoとはNQPによって記述され、MoarVM、JVM、Javascript上で動作するPerl6の実装である。Rakudo自体はNQPで記述されている箇所と、Rakudo自身で記述されている箇所が混在する。

ビルド時にはNQPインタプリタが必要となる為、ソースコードからビルドする際は予めNQPインタプリタであるnqpをビルドする必要が存在する。

図??にこの様子を表した。

Code 2.1: 環境付き継続

```
1  __code cs(__code (*ret)(int, void*), void *env){
2    /* C0 */
3    goto ret(1, env);
4  }
5
6  int funcB(){
7    /* B0 */
8    goto cs(__return, __environment);
9    /* B1 (never reached). */
10   return -1;
11 }
12
13 int funcA(){
14   /* A0 */
15   int retval;
16   retval = funcB();
17   /* A1 */
18   printf("retval␣=␣%d\n", retval);
19   /* retval should not be -1 but be 1. */
20   return 0;
21 }
```

このように、環境付き継続を用いることでC、CbC間の処理の移動が可能になる。

謝辞

本研究の遂行，また本論文の作成にあたり、御多忙にも関わらず終始懇切なる御指導と御教授を賜わりました河野真治准教授に深く感謝したします。また、本研究の遂行及び本論文の作成にあたり、数々の貴重な御助言と細かな御配慮を戴いた伊波立樹さん、比嘉健太さん、並びに並列信頼研究室の皆様に深く感謝致します。

最後に、有意義な時間を共に過ごした情報工学科の学友、並びに物心両面で支えてくれた両親に深く感謝致します。