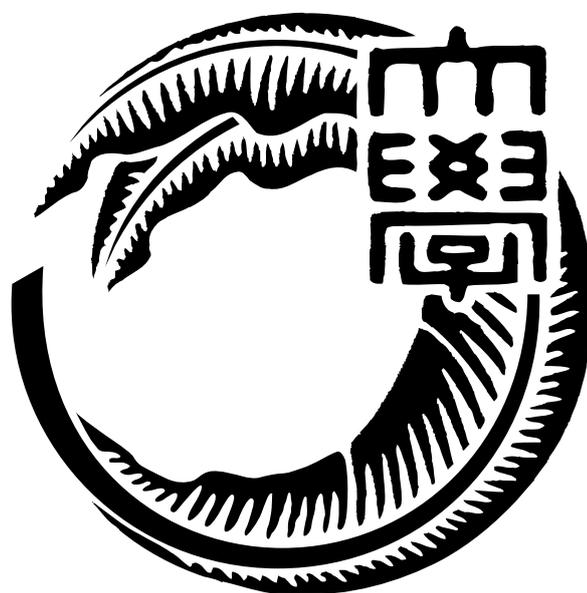


平成30年度 卒業論文

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

第 1 章 現在の Perl6 処理系	1
第 2 章 Perl6	2
2.1 Perl6 の概要	2
2.2 Rakudo	2
2.3 MoarVM	3
2.4 NQP	3

目 次

コード目次

2.1 Rakudo の実装の一部	2
2.2 フィボナッチ数列を求める NQP のソースコード	3
2.3 NQP が加算命令を生成する箇所	4

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。Perl6 そのものはスクリプト言語として実装されており、漸進的型付け言語である。言語的な特徴としては、独自に Perl6 の文法を拡張可能な Grammer、Perl5 と比較してオブジェクト指向言語としての機能の強化などが見られる。

2.2 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 で実装されている箇所と、Perl6 で実装されている箇所が混在する。これらは拡張子によって区別され、NQP は .nqp、Perl6 は .pm6 が拡張として設定されている。実際に NQP で実装されている箇所の一部を Code 2.1 に示す。

Code 2.1: Rakudo の実装の一部

```
1 use Perl6::Grammar;
2 use Perl6::Actions;
3 use Perl6::Compiler;
4
5 # Initialize Rakudo runtime support.
6 nqp::p6init();
7
8 # Create and configure compiler object.
9 my $comp := Perl6::Compiler.new();
10 $comp.language('perl6');
11 $comp.parsegrammar(Perl6::Grammar);
12 $comp.parseactions(Perl6::Actions);
13 $comp.addstage('syntaxcheck', :before<ast>);
14 $comp.addstage('optimize', :after<ast>);
15 hll-config($comp.config);
16 nqp::bindhllsym('perl6', '$COMPILER_CONFIG', $comp.config);
```

Rakudo を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。Rakudo のビルド時にはこの nqp と、nqp が動作する VM を設定として与える必要がある。この両者を指定しない場合、ビルド時に動的に NQP、MoarVM をソースコードをダウンロードし、ビルドを行う。

2.3 MoarVM

MoarVM とは Rakudo 実装で主に使われる仮想機械である。Rakudo では Perl6 と NQP を実行する際に仮想機械上で実行する。この仮想機械は OS レベルの仮想化に使用する VirtualBox や qemu と異なり、プロセスレベルの仮想機械である。Rakudo ではこの仮想機械に MoarVM、Java の仮想機械である JVM(JavaVirtualMachine) が選択可能である。MoarVM はこの中で Rakudo 独自に作成された仮想機械であり、現在の Rakudo プロジェクトの主流な実装となっている。

MoarVM は C 言語で実装されており、レジスタベースの仮想機械である。MoarVM は NQP や Perl6 から与えられた MoarVM バイトコードを評価する。

2.4 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。歴史的には Perl6 の主力実装が Parrot であった際に開発され、現在の Rakudo に引き継がれている。Rakudo における NQP は、Parrot 依存であった実装が取り払われている。

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。

NQP のコード例を Code2.2 に示す。

Code 2.2: フィボナッチ数列を求める NQP のソースコード

```
1  #! nqp
2
3  sub fib($n) {
4      $n < 2 ?? $n !! fib($n-1) + fib($n - 2);
5  }
6
7  my $N := 29;
8
9  my $t0 := nqp::time_n();
10 my $z := fib($N);
11 my $t1 := nqp::time_n();
12
13 say("fib($N) = " ~ fib($N));
14 say("time = " ~ ($t1-$t0));
```

Perl6 は NQP で実装されている為、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 で実装されており、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。実際に NQP 内部で入力として与えられた NQP から加算命令を生成する部分を Code2.3 に示す。

Code 2.3: NQP が加算命令を生成する箇所

```
1 $ops.add_hll_op('nqp', 'preinc', -> $qastcomp, $op {
2   my $var := $op[0];
3   unless nqp::istype($var, QAST::Var) {
4     nqp::die("Pre-increment can only work on a variable");
5   }
6   $qastcomp.as_mast(QAST::Op.new(
7     :op('bind'),
8     $var,
9     QAST::Op.new(
10      :op('add_i'),
11      $var,
12      QAST::IVal.new( :value(1) )
13    )))
14 });
```

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。この2度目のビルドで、ソースコードからビルドされた VM バイトコードで NQP 自身をビルドした事になる。処理系自身をその処理系でビルドする事をセルフビルドと呼び、NQP はセルフビルドしたバイナリを利用する。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。

参考文献

- [1] 唐鳳. Pugs: A perl 6 implementation.
- [2] ThePerlFoundation. Perl6 documentation.
- [3] ThePerlFoundation. Perl 6 design documents.
- [4] ThePerlFoundation. Roast – perl6 test suite.
- [5] ParrotFoundation. Parrot.
- [6] ThePerlFoundation. Nqp opcode list.
- [7] ThePerlFoundation. Nqp - not quite perl (6).
- [8] 大城信康, 河野真治. Continuation based c の gcc 4.6 上の実装について. 第 53 回プログラミング・シンポジウム, 1 2012.
- [9] 徳森海斗, 河野真治. Llvm clang 上の continuation based c コンパイラの改良. 琉球大学工学部情報工学科平成 27 年度学位論文 (修士), 2015.
- [10] 並列信頼研究室. Cbc.gcc.
- [11] 並列信頼研究室. Cbc.llvm.
- [12] Jonathan Worthington. Rakudo and nqp internals.
- [13] Jonathan Worthington. Rakudo and nqp internals - day1.
- [14] Anton Ertl. Threaded code.
- [15] Kaito TOKUMORI and Shinji KONO. Implementing continuation based language in llvm and clang. *LOLA*, 7 2015.
- [16] 光希宮城, 優桃原, 真治河野. Gears os のモジュール化と並列 api. Technical Report 11, 琉球大学大学院理工学研究科情報工学専攻, 琉球大学大学院理工学研究科情報工学専攻, 琉球大学工学部情報工学科, may 2018.
- [17] 笹田耕一, 松本行弘, 前田敦司, 並木美太郎. Ruby 用仮想マシン yarv の実装と評価. 情報処理学会論文誌プログラミング (PRO) , 2 2006.

- [18] James R. Bell. Threaded code. *Commun. ACM*, Vol. 16, No. 6, pp. 370–372, June 1973.
- [19] Ian Piumarta and Fabio Riccardi. Optimizing direct threaded code by selective inlining. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI '98, pp. 291–300, New York, NY, USA, 1998. ACM.
- [20] Mike Pall. The luajit project.

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りました河野真治准教授に深く感謝いたします。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も深く感謝致します。

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。