Rubyを動作可能な仮想マシンの設計と実装

情報科学専攻 55組 1番 岡本八仁

目次

1. システム概要
2. デモ
3. 設計
4. 実装
5. おわりに

システム概要

1. システム概要
2. デモ
3. 設計
4. 実装
5. おわりに

動作の流れ（1/2）

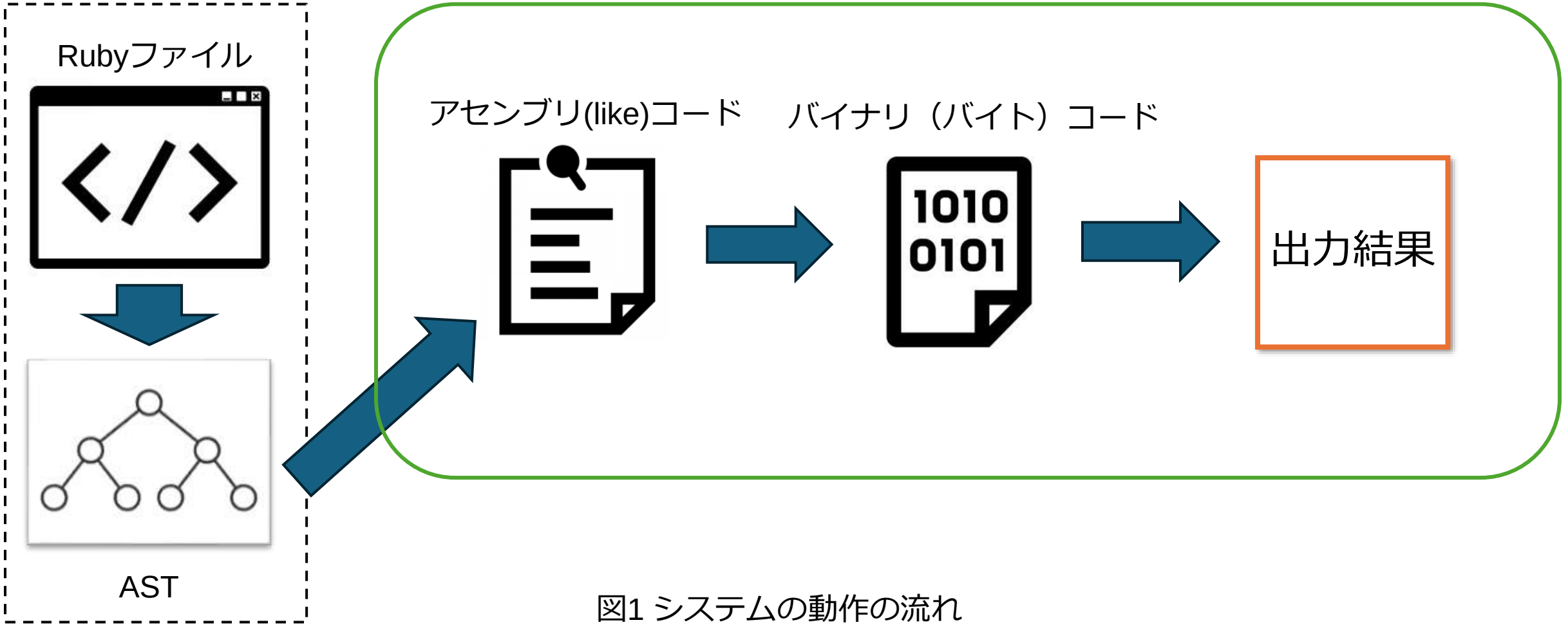


図1 システムの動作の流れ

動作の流れ（2/2）

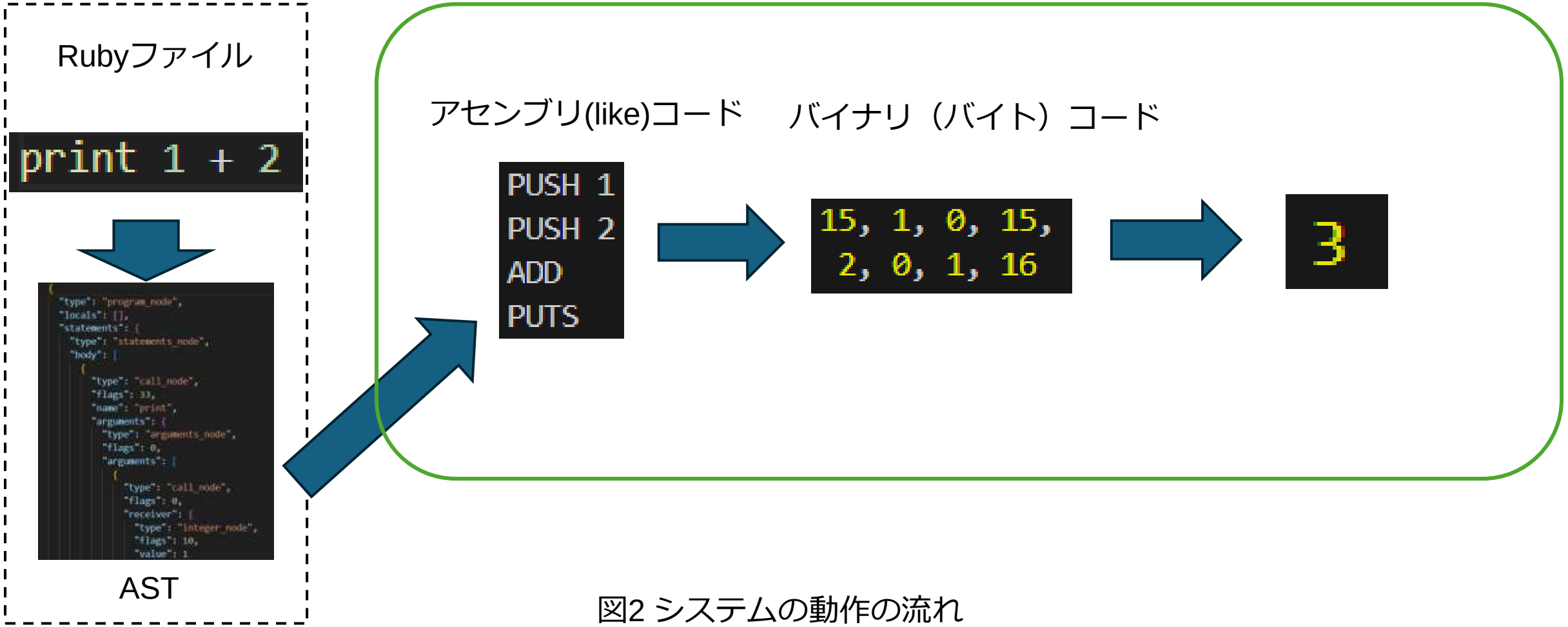


図2 システムの動作の流れ

デモ

1. システム概要
2. デモ
3. 設計
4. 実装
5. おわりに

素数判定

素数判定を行いたい数字を変数として宣言

・素数なら”Prime”

・素数でなければ”NotPrime”

と出力

（簡略化のため$O(\sqrt{N})$ではなく$O(N)$）

選定理由

- 変数
- for
- if
- 比較
- 出力

これらを一度に使用できるため

2. デモ

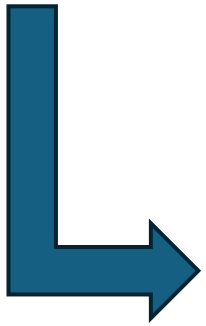
実演

設計

1. システム概要
2. デモ
3. 設計
4. 実装
5. おわりに

仮想マシンの設計方針（1/2）

スタックマシン vs レジスタマシン



- **実装量が比較的少ない**
  - オペランドを指定する場面が少ない
- ユースケース的に最適化や高速化を行う必要がない

仮想マシンの設計方針（2/2）

スタックマシン vs レジスタマシン

・コードの保守性や再利用性が高まる
・最適化がしやすい
・複雑な命令を扱いやすい

大規模なプロジェクトでは**レジスタマシン**の方が優れている
（JVMをレジスタマシンにJITする研究[1]もある）

[1] Byung-Sun Yang et al., "LaTTe: a Java VM just-in-time compiler with fast and efficient register allocation," 1999 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00425), Newport Beach, CA, USA, 1999, pp. 128-138, doi: 10.1109/PACT.1999.807503.

対応可能な構文

- 四則演算
- 比較演算
- 変数（代入・参照）
- if文
- for文/while文
- puts/print（文字列も可）
- exit

参考：Ruby 3.4 リファレンスマニュアル[2]

[2] https://docs.ruby-lang.org/ja/3.4/doc/index.html

対応不可能な構文

- クラス
- 一部の演算子
- コメント
- 一部の制御文
- メソッド（宣言/呼び出し）
- 標準入力

参考：Ruby 3.4 リファレンスマニュアル[2]

[2] https://docs.ruby-lang.org/ja/3.4/doc/index.html

実装

1. システム概要
2. デモ
3. 設計
4. 実装
5. おわりに

開発環境

- 使用言語：TypeScript
- npmで管理

（GitHub上にコードを公開[3]しています）

[3] https://github.com/Hachijin-Okamoto/rubyVM

AST→アセンブリ

- 再帰関数でASTを順番に見ていく

- 定数として分割しているので，
命令文の名称変更は容易

```ts
export const ASSEMBLY = {
  // 四則演算
  ADDITION: "ADD",
  SUBTRACTION: "SUB",
  MULTIPLICATION: "MUL",
  DIVISION: "DIV",
  REMAINDER: "REM",
  POWER: "POW",

  // 比較演算
  GREATER: "GT",
  LESS: "LT",
  GREATER_EQUAL: "GTE",
  LESS_EQUAL: "LTE",
  EQUAL: "EQ",
  NOT_EQUAL: "NEQ",

  // 変数
  ASSIGNMENT: "STORE",
  REFERENCE: "LOAD",

  // ジャンプ命令
  JUMP: "JUMP",
  JUMP_IF_FALSE: "JIF",

  // その他
  OUTPUT: "PUTS",
  FUNCTION_CALL: "CALL",
  NUMBER: "PUSH_NUM",
  STRING: "PUSH_STR",
  END: "HALT",
} as const;
```

constants.ts

アセンブリ→バイトコード（1/2）

1. ラベル（ジャンプ命令に使用）をアドレスに置き換える
2. 各命令文をバイナリに置き換える

という作業を行う**2パスアセンブリ**構成

アセンブリ→バイトコード（2/2）

・オペコード：1バイト
・オペランド：2バイト（リトルエンディアン）

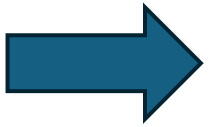
前のバイトから値を格納していく
（例：28 → 0x1c 0x00）

※文字列は1文字1バイト（UTF-8エンコード）

仮想VM

- 現在のカウンタ
- スタック
- バイトコード
- レジスタ（変数の値保管用）

を持つクラスを作成

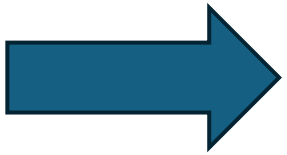


作成したバイトコードを渡して動作させる

おわりに

1. システム概要
2. デモ
3. 設計
4. 実装
5. おわりに

まとめ

Rubyファイル  抽象構文木

アセンブリ風コード

バイトコード

実行結果　と動作するシステムを作成した.

感想

- 実装がかなり重かった
  - スタックマシンを選んだのにも関わらず
- バイトコード層でのバグを取り除くのにとても時間がかかった

参考文献

[1] Byung-Sun Yang et al., "LaTTe: a Java VM just-in-time compiler with fast and efficient register allocation," 1999 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00425), Newport Beach, CA, USA, 1999, pp. 128-138, doi: 10.1109/PACT.1999.807503.

[2] Ruby:Ruby3.4 リファレンスマニュアル, https://docs.ruby-lang.org/ja/3.4/doc/index.html

[3] GitHub:rubyVM, https://github.com/Hachijin-Okamoto/rubyVM

目次

以下，補足スライド