

低レイヤ言語の動作理解を支援する ブラウザ上で動作可能な仮想マシンの設計

情報科学専攻 55組 1番 岡本八仁

目次

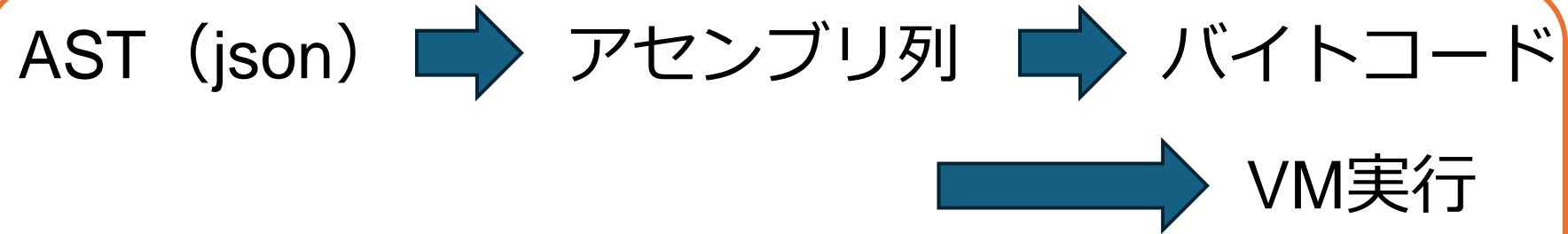
1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

はじめに

1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

1. はじめに

概要



これらを全て**ブラウザ上で実行し**、
スタックの内容を可視化できるシステムを作成した。

1. はじめに

動機

- アセンブリ
- バイナリ

といった低レイヤな環境では, **動作の理解が難しい**
(内部状態を把握できない)

可視化することで理解を支援する

提案手法

1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

2. 提案手法

動作の流れ

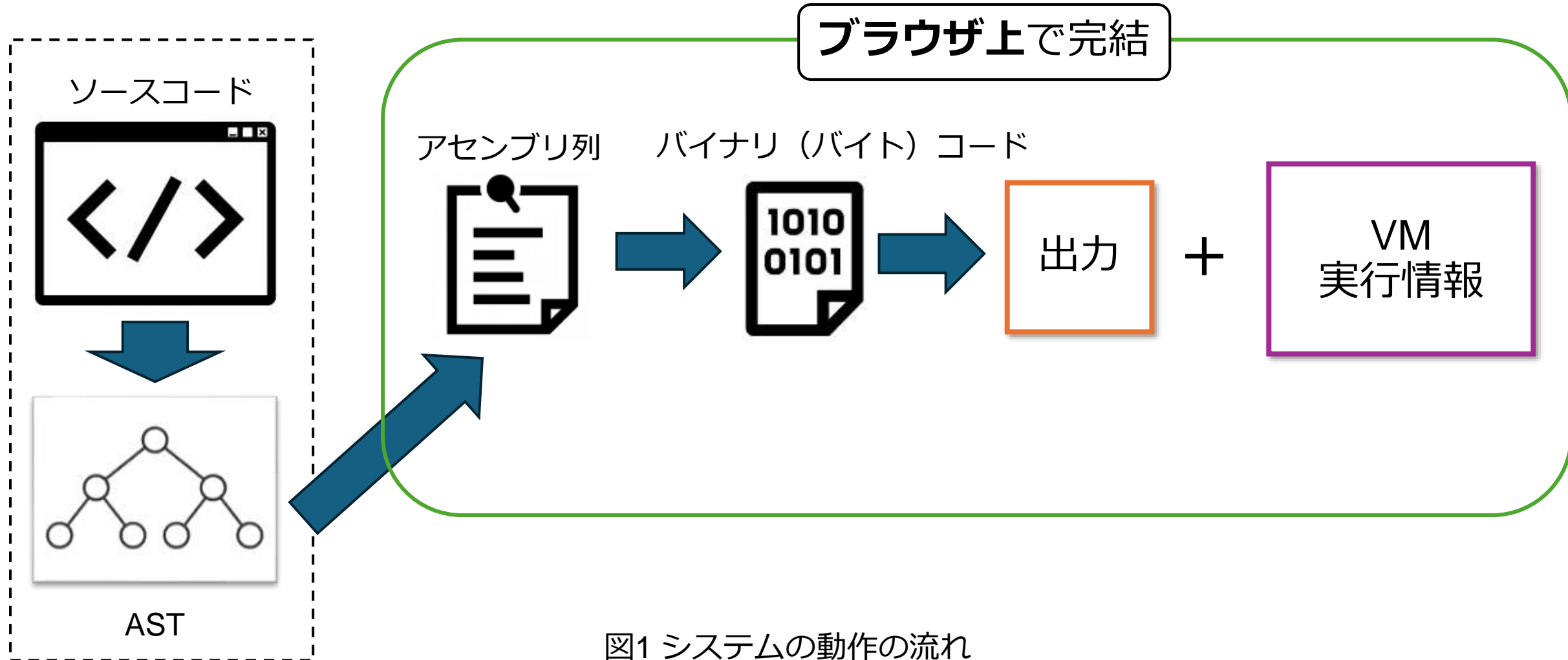
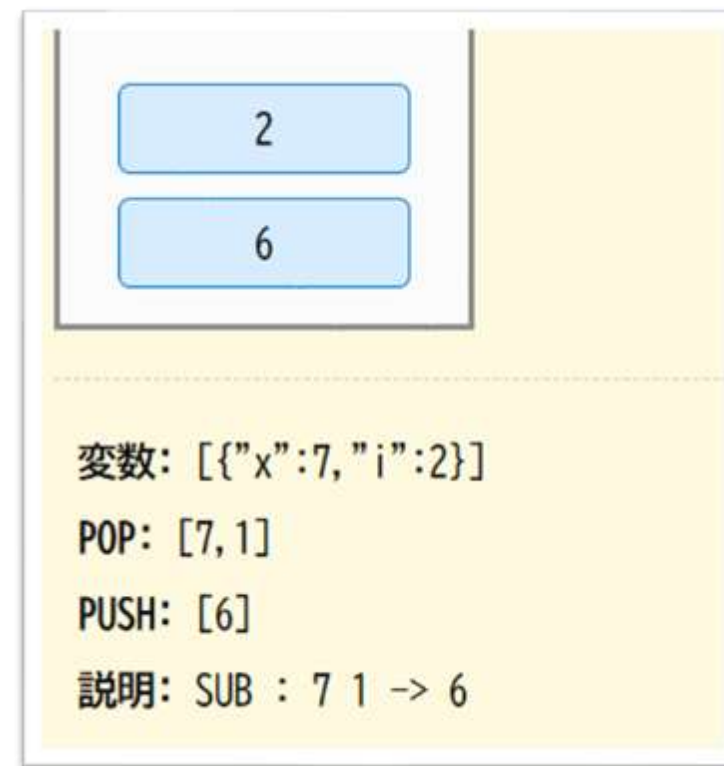


図1 システムの動作の流れ

2. 提案手法

VM実行情報

- VMの実行1ステップごとに
 - スタックの中身
 - 変数情報
 - スタックのpop/pushを記録



可視化することで理解を支援する

2. 提案手法

アセンブリ列→VM情報

アセンブリ列をクリック ➡ VM実行情報を表示

- ・ ループ
- ・ 関数

などで複数回処理が行われる場合は
実行ステップ数を指定可能



デモ

1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

設計

1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

4. 設計

設計方針

VMで実行させるプログラミング言語として**Ruby**[1]を想定

- ・ ASTの読み取り：Prism[2]により生成されたjsonファイルが必要
- ・ 制御文, 関数：Rubyの文法に従う

[1] Ruby:オブジェクト指向スクリプト言語 Ruby, <https://www.ruby-lang.org/ja/>

[2] prism, <https://github.com/ruby/prism>

4. 設計

対応可能な構文 (Ruby)

- 各種演算
 - 四則演算, 論理演算
- 変数
 - 数値, 文字列
 - **配列**
- 制御文
- 関数
 - 標準出力
 - 自作関数の宣言/呼び出し
 - 再帰関数

4. 設計

実行形態

- 本システムは**ブラウザ上で実行可能**
 - 即座に使用できる → 初学者にも優しい
 - プラットフォームに依存しない → 実行がしやすい
 - ファイルが送信されない → 安全である

4. 設計

仮想マシンの設計

スタックマシンとして設計を行った

(vs.レジスタマシン)

- ・オペランドが少なく, 操作の対象がわかりにくい
- ・常にスタックの値を意識する必要がある

等の理由から, 動作の理解が**より難しい**と考えたため
(可視化することの恩恵が大きい)

実装

1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

5. 実装

ラベル

- 制御文 (for, while, if)
- 関数呼び出し

では, Jump命令とともに「ラベル (飛び先)」を用いる

アセンブリ→バイトコード変換時:

1回目: **ラベルのアドレス**を記録

2回目: Jump命令のオペランドを**ラベルのアドレス**に変換

2パスアセンブリ方式を採用

5. 実装

数値・文字列

- 数値：2バイト（リトルエンディアン）

- int
- ラベル
- 変数名
- 関数名

} すべて数値で管理

- 文字列：1文字1バイト（utf-8エンコード）
→日本語・空白には対応できていない

5. 実装

(自作) 関数

再帰関数：スコープごとに環境（変数の値を記録したもの）
を**スタック**として持つ

関数呼び出し：

AST→アセンブリ列への変換時に

(配列名：引数の数) のセットを記録しておく

 VM実行の際に引数の数だけスタックからpop

5. 実装

VM状態の可視化

VMの実行1ステップごとに

- ・スタックの内部状態
- ・変数の値
- ・アセンブリ列の場所

などを記録 → アセンブリ列の番号から検索・表示

5. 実装

開発環境

- 開発言語：TypeScript
 - ブラウザで実行可能なため
- ビルドツール：Vite[3]
 - フロントエンド開発を効率化
- 構文解析：ANTLR[4]



[3] Vite, <https://ja.vite.dev/>

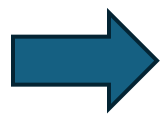
[4] ANTLR, <https://www.antlr.org/>

おわりに

1. はじめに
2. 提案手法
3. デモ
4. 設計
5. 実装
6. おわりに

今後の展望 (1/2)


- TS + ANTLRで自前のRuby-parserを作る
 - Prism用に書いたコードと対応させるのが大変で今回は断念



ブラウザ上でRubyコードを入力したら
即座に変更が反映されるようにする

今後の展望 (2/2)

- UI部分を改善する
 - React/Vueなどを用いてUX向上を目指す

 実行の様子を自動再生などできるようにする

参考文献

[1] Ruby:オブジェクト指向スクリプト言語 Ruby, <https://www.ruby-lang.org/ja/>

[2] prism, <https://github.com/ruby/prism>

[3] Vite, <https://ja.vite.dev/>

[4] ANTLR, <https://www.antlr.org/>

- GitHub:rubyVM-browser, <https://github.com/Hachijin-Okamoto/rubyVM-browser>

ご清聴ありがとうございました

1. はじめに (pp. 3-5)
2. 提案手法 (pp. 6-9)
3. デモ (p.10)
4. 設計 (pp. 11-15)
5. 実装 (pp. 16-21)
6. おわりに (pp.22-24)