IPA

2021 年度 未踏 IT 人材発掘・育成事業 採択案件評価書

1. 担当 PM

竹迫 良範（株式会社リクルート データプロダクトユニット ユニット長）

2. クリエータ氏名

Jantakorn Passawee（株式会社はてな）

3. 委託金支払額

616,075 円

4. テーマ名

Go の資産を再利用できるコンパイラ基盤

5. 関連 Web サイト

なし

6. テーマ概要

本プロジェクトでは、Go Assembly と呼ばれる Go コンパイラ内部で使われるアセンブリコードをターゲット言語するコンパイラ基盤「GGVM」を提案する。既存のプロジェクトである LLVM に対して、GGVM は豊富な Go のライブラリ資産を再利用できるといった優位性がある。

7. 採択理由

本提案は LLVM や WASM のようなコンパイラ基盤を Go Assembly ベースで実現するという野心的なプロジェクトである。過去 Perl コミュニティで Parrot というレジスタベースの仮想マシンが提案されたことがあり、Perl6 や Python をサポートするために PVM、PASM、PIR、PBC、IMCC、PIRC など様々なレイヤーでの実装が作られたが、実行速度の問題もありメジャーになることはできなかった。最初の性能要件として実行速度が重要視されて作られた Go Assembly をベースにコンパイラ基盤を開発することが出来れば、Parrot で実現できなかったことが今の時代でできるかもしれない。コンパイラ共通基盤は COINS など学術系でも古くから長く研究されている分野でもあるため、世代を超えて、アカデミアとインダストリーが融合してプログラミング言語の基盤技

術と実装が進化していくことを期待したい。

8. 開発目標

Go コンパイラの特徴としてソースコードを様々なアーキテクチャで動くシングルバイナリとしてクロスコンパイルできることが挙げられる。これはコンテナのサイズを小さくしたいモダンなWebアプリケーションアーキテクチャと相性がよく、実装言語として採用される理由に一役買っている。他にも Go には以下の優位性がある。

- コンパイラが生成したバイナリの実行速度のパフォーマンスが良い
- libc に頼らずとも syscall を呼べる様なとても豊富な標準ライブラリ
- goroutine による並行処理の簡潔さ
- 静的型付き言語流行に依る脱 LL（Ruby, Python）トレンド

しかしその一方で、Go 自体は以下の課題もある。

- ジェネリクスを始めとした高級な型システムの欠如
- immutability
- 関数型言語にあるパターンマッチなどの高級な言語機能
- コード生成器が Go の AST と結合していて別のコンパイラフロントエンドから利用しにくい

これから新しいプログラミング言語を自作する人に対して、LLVM より軽量かつ、ネイティブバイナリに Go のランタイムと豊富な標準ライブラリの資産が利用できるコンパイラ基盤 GGVM を提供することを目標とする。これにより、自作言語実装において、LLVM を使うか、もしくは自力で対象アーキテクチャのアセンブリを生成するか、の 2 択だけではなく、第 3 の選択肢として GGVM が候補に加わることで、かつて Java の資産を再利用した言語 Kotlin や Scala が登場したようなことが Go 言語周辺で行われることを期待する。

9. 進捗概要

本プロジェクトで実装したコンパイラ基盤 GGVM のアーキテクチャを図 1 に示す。利用者の入出力は Input と Binary で、内部入力はそれぞれ矢印の上のオブジェクトになる。実装に用いた言語は Rust で、内部実装を安定化させるために Go のバージョンは 1.17.0 で固定した。

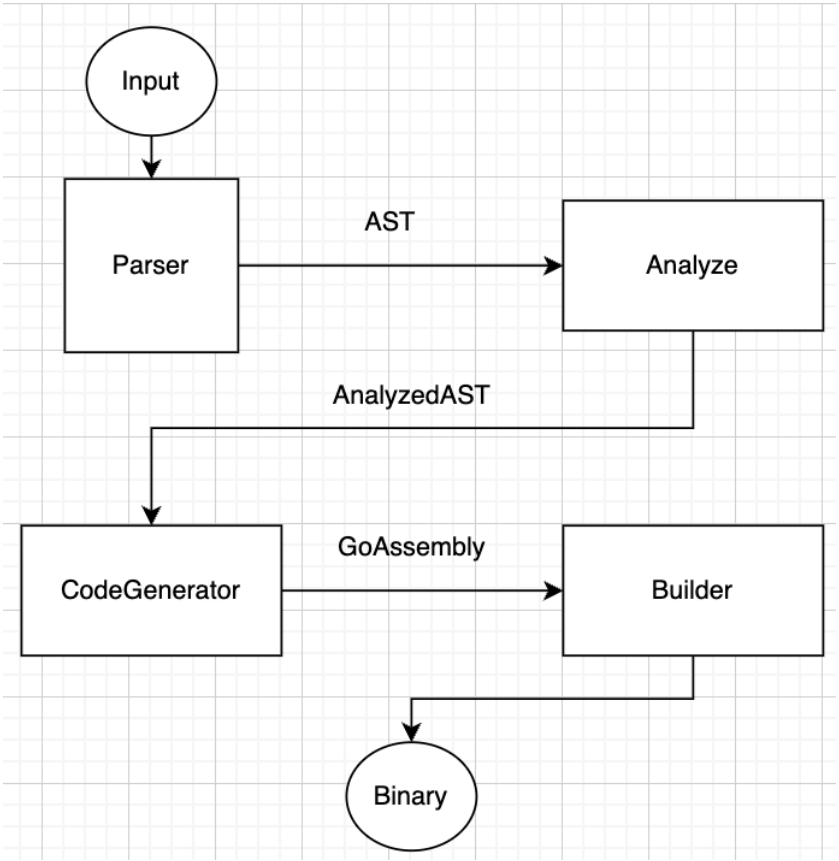

図 1：GGVM のアーキテクチャ

開発した GGVM は図 2 で示す BNF で定義された GGVM-IR を入力文字列として受け取り、内部で Go Assembly を経由して、最終的に実行可能なバイナリを生成する。

```
<Program> := <Func>
<Func> := func <FunctionName>() <Type> { <Statement> }
<Statement> := <LocalStatement> | <InstructionStatement>
<LocalStatement> := local <LocalIdent> <AddInstruction>
<InstructionStatement> := <AddInstruction> | <RetInstruction> | <CallInstruction>
<AddInstruction> := add <Type> <Operand> , <Operand>
<CallInstruction> := call <Callee>
<RetInstruction> := ret <Type> <Operand>
<Type> := int
<Operand> := <Var>
<Var> := %<Ident>
<FunctionName> := $<Ident>
<Callee> := <LocalIdent>
<LocalIdent> := %<Ident>
<Ident> := <Alpha>(<AlphaOrNumeric>)*
<AlphaOrNumeric> := <Alpha> | <Numeric>
<Alpha> := 'A' ... 'Z'
<Numeric> := '1' ... '9'
```

図 2：GGVM-IR の BNF

現状の実装では、一つの関数をコンパイルしてバイナリを実行できることを確認できた。

10. プロジェクト評価

途中 Go Assembly／ランタイムの解析に想定以上の時間がかかってしまいプロジェクトの進捗が滞ってしまう時期があったが、最終的に本プロジェクトで開発したコンパイラ基盤 GGVM は、Go の標準ライブラリの関数のラッパを main.go に埋め込み、GGVM が生成する Go Assembly からそのラッパを呼び出すことで間接的に自作言語から Go の標準ライブラリの関数を再利用できることを実現した。

11. 今後の課題

最低限のコンセプト実証までたどり着いたが、本プロジェクトが目指すコンパイラ基盤の目標の実現にはまだ数年以上かかる見込みであり、最初のスタートラインに立った状態である。コンパイラ基盤そのものもまだ機能が足りないため、加算以外にも減算、乗算、除算、ラベル、JUMP、LOAD、STORE、LEAQ、条件分岐、文字列対応、Go のスライス対応、ポインタ対応、構造体対応、goroutine 対応など多くの機能を実装する必要がある。そうして開発したコンパイラ基盤 GGVM を利用した自作プログラミング言語の実装の実例を示す必要がある。