6K-07

利便性向上のためのコンパイラの検討

吉川 慎太郎 †1　和泉 諭 †1
†1 仙台高等専門学校

1　はじめに

プログラミング言語 C++ では，メタプログラミングの 1 つであるテンプレートメタプログラミングが広く使われている．The Rust Programming Language 日本語版によると，「メタプログラミングは，書いて管理しなければならないコード量を減らすのに有用で，これは，関数の役目の一つでもあります.」[1] とある．しかし，テンプレートメタプログラミングにはプログラムの可読性が低下する問題がある．また，コンパイルに失敗した際のエラーメッセージが煩雑になりやすい．

本研究では，前述の問題を改善しコンパイラの利便性を高めることを目的とし，C++ をベースに新たなメタプログラミング環境を追加した拡張言語の設計と，拡張言語から C++ のソースコードへ変換するトランスパイラの開発を行う．

2　関連研究

2.1　Artisan

異機種混在が進む今日，アーキテクチャの詳細を隠蔽する抽象化と，アーキテクチャの特性を最適化して利用するという，一見矛盾する機能を提供する設計ツールへの需要が高まっている．

Artisan [2] は，様々な最適化を Python でプログラミング可能にし，C++ のソースコードから最適化された C++, FPGA コードを生成する処理を自動化する．これによりアプリケーションの開発者は最適化の問題を意識することなくアルゴリズムの実装を行うことができる．さらに，ハードウエアとドメインの専門家はアルゴリズムを意識することなく，最適化とマッピングの実装を行うことができる．

2.2　プログラミング言語 Rust

本研究で開発するトランスパイラに類似する機能を有するプログラミング言語として Rust [1] がある．これには宣言的マクロと手続き的マクロがある．Rust におけるマクロとは，コード生成を行うメタプログラミングの機能を意味する．ソースコード内にマクロ名とカッコを記述すると，カッコで囲まれたトークン列を別のトークン列へ変換する．

宣言的マクロは，トークン列をパターンマッチングで任意のトークン列へと変換する機能を持つ．手続き的マクロはこれを別に書いたプログラム上で行うことができ，より複雑な処理が行えるようになる．引数がトークン列，戻り値がトークン列の関数を呼び出すことで変換を行い，任意のエラーメッセージの出力ができる．

これらの機能は Rust の言語機能として提供されるため，プログラムの可読性を維持したまま型やロジックを生成できる他，読みやすいエラーメッセージを出力できるという特徴がある．これら特徴を参考にし開発を行う．

3　提案

C++ のソースコードをコンパイラによって処理する際，通常プリプロセッサによって事前にコードの変換が行われる．特にプリプロセッサのマクロ機能は任意のトークン列に置き換えができる．このマクロ機能に注目し，任意の機能を持つマクロを開発者が実装ができるプロトタイプを検討する．

プロトタイプの構成を図 1 に示す．C#により実装されたコード生成プログラムはプリプロセッサによって処理されたソースコードを受け取り，新たに生成した C++ のソースコードをコンパイラへ渡す．

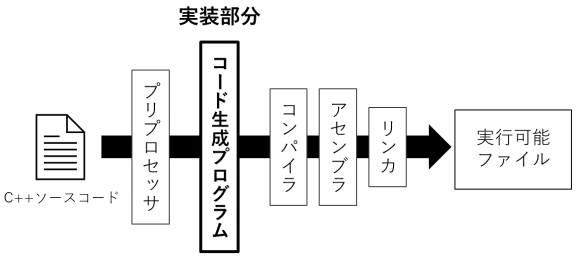

図 1: プロトタイプの構成

A Study on the Meta-Programming Environment for Improving Usability
Shintaro KIKKAWA†1, Satoru IZUMI†1,
†1National Institute of Technology, Sendai College

4 動作例

プロトタイプの動作例を図 2 に示す．生成には正規表現と C#スクリプトを用いる．正規表現は置き換え対象 (__DM_GENERATE()) の認識と引数 ("HELLO", "World") の抽出を行い，該当する部分の置き換えをする．置き換え後の文字列 ("Hello, World") の生成はスクリプトが行う．引数の値を文字列として受け取り，処理やエラーチェックをした後に生成した文字列を返り値として出力する．この一連の変換処理を入力のソースコード全体に行うことでコードの生成とする．

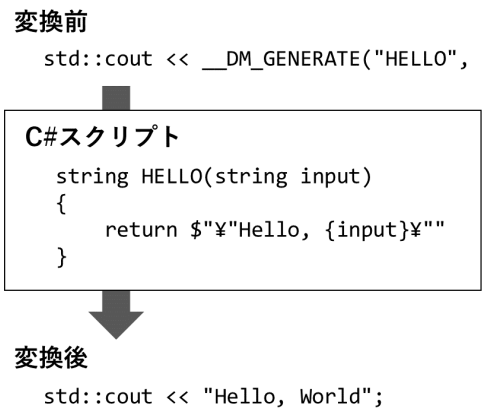

図 2: プロトタイプの動作例

5 実験

プロトタイプシステムを用いた実験を行った．具体的にはコンパイル処理をした際のエラー時と正常時の実行結果を比較した．実験結果を図 3 と図 4 に示す．TEST マクロの引数が空の場合，コンパイル時に引数を空にできないという読みやすいエラーを表示することができた．TEST マクロの引数に文字列が入っている場合は日付と時刻を付加した文字列を自動で生成し，実行可能ファイルを出力することができた．

C#のスクリプトで入力値のチェックやエラーメッセージの出力を比較的簡単に行えるため，このシステムはテンプレートメタプログラミングを使った場合と比べ，エラーメッセージの煩雑さを一部解消することができた．

6 おわりに

開発したプロトタイプでは，エラーメッセージの煩雑さを解決することができた．課題として，テンプレートメタプログラミングのような型システムを使った条件分岐はできない点，ロジックを記述するプログラムとコード生成を行うプログラムの 2 つに分かれており扱いづらい点が挙げられる．

今後は C++ の型システムを用いた処理ができるよう，C++ の文法を解釈するパーサーを組み込むことや，ロジックとコード生成を 1 つのプログラムとして書ける拡張言語の仕様の検討に取り組みたい．

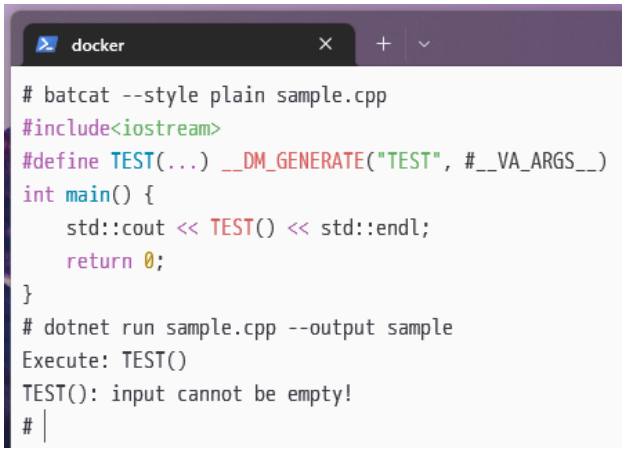

図 3: エラー時の動作結果

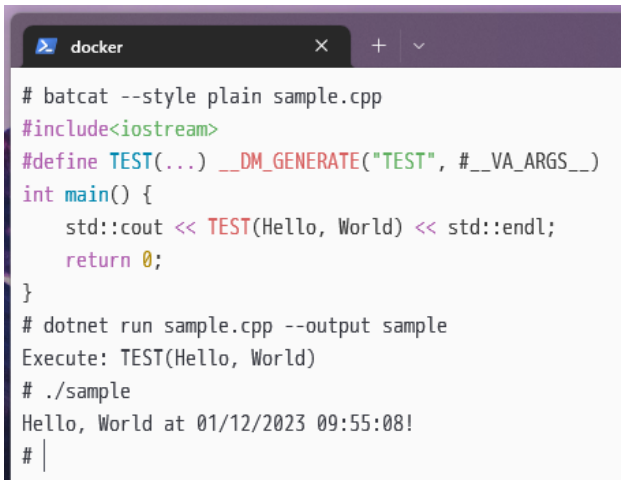

図 4: 正常時の動作結果

参考文献

[1] Steve Klabnik, C. N.: マクロ – The Rust Programming Language 日本語版，，入手先 〈https://doc.rust-jp.rs/book-ja/ch19-06-macros.html〉（参照 2023-01-11）．

[2] Vandebon, J., Coutinho, J. G., Luk, W. and Nurvitadhi, E.: Enhancing high-level synthesis using a meta-programming approach, *IEEE Transactions on Computers*, Vol. 70, No. 12, pp. 2043–2055 (2021).