第1章 コンパイラ理論の基本を集中レッスン

構文解析や意味解析を体験

宮田 賢一

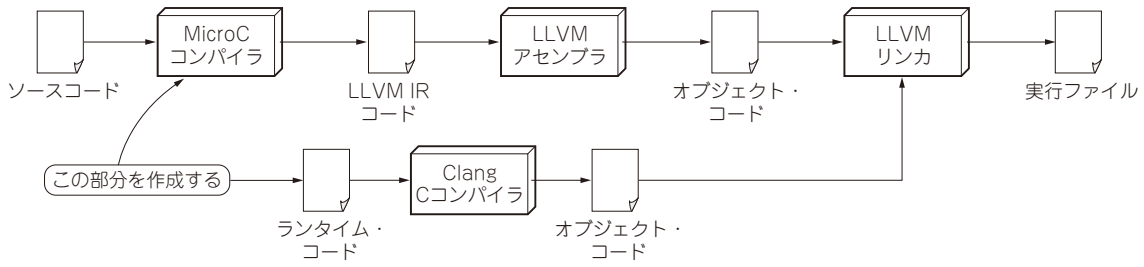

図1 本稿で作成するコンパイラ・システム全体の流れ

コンパイラのフロントエンド部

本実験では，実際にコンパイラを作成します．作成するのはフロントエンド部とし，バックエンドとしてLLVMを利用することにしました．これによりLLVMが持つ豊富な最適化機能をそのまま適用できます．また，フロントエンド部はコンパイラ理論の基本部分が多く含まれるため，コンパイラの学習にも適していると思います．

● 全体の構成

コンパイラ・システムの全体の流れを**図1**に示します．本実験用にC言語のサブセットである小さな言語MicroC言語を設計します．MicroC言語コンパイラは，ソース・プログラムからLLVM IRコードを生成するものとします．LLVM IRコードは，LLVMに付属のアセンブラでオブジェクト・コードに変換できます．さらにこのオブジェクト・コードを実行可能とするために，C言語でランタイム・コードを作成しておき，MicroCのオブジェクト・コードとランタイムのオブジェクト・コードをリンクし，実行ファイルとします．

● MicroCコンパイラの構成

図2にこの実験で作成するコンパイラ内部の構成を示します．フロントエンドとして必要な字句解析，構文解析，意味解析，LLVM IR生成部を作ります．

字句解析と構文解析（パーサ）は，Python Lex-Yacc（PLY）を使って字句定義，構文定義から自動生成するものとします．コンパイラ内部では，ソースコードを3アドレス・コードというシンプルな命令表現に変換して各種解析処理を実行した後，LLVM IRに変換します．

● Python Lex-Yacc（PLY）について

PLYは字句解析と構文解析を処理するPythonコードを自動生成してくれるツールです（`http://www.dabeaz.com/ply/`）．UNIXでは古くから標準コマンドして用意されていた`lex/yacc`や`lex/yacc`のGNU版である`flex/bison`と同等の処理をPythonで実装したものです．基本的な記法や内部のアルゴリズムは`lex/yacc`を踏襲したものになっています．つまり字句解析は正規表現により定義し，言語の文法はLALRによる構文解析処理を行います．

● LLVMとLLVM IRについて

LLVMは，多様なプログラミング言語とターゲットCPUに対応可能なコンパイラ基盤であり，内部ではLLVM IRという中間コードを使用しています．別の見方をすると，LLVM IRは仮想的なマシンで動作するアセンブリ言語のコードとも言え，LLVM IRの仕様に従って直接LLVM IRコードを記述することも可能です．