豊田工業高等専門学校　情報工学科
Department of Information and Computer Engineering,
National Institute of Technology, Toyota College

卒業論文
Graduation Thesis
February 22, 2024

コンパイラ学習者のための Yacc 用言語サーバの開発

稲垣研究室　　石部 鳳空

あらまし　これは豊田高専情報工学科の卒業論文用 LaTeX テンプレートです．提供される docx ファイルとほぼ同様の見た目になるようにフォントや文字間隔や余白などを調整しました．LuaLaTeX 処理系の使用を想定しています．

キーワード　LaTeX, Word，豊田高専，卒業論文，テンプレート

1.　背景

1.1.　Yacc とは

Yacc は高専や大学のコンパイラの講義やコンパイラ開発者，研究者などに広く使用されている一般的な LALR(1) パーサジェネレータである．Yacc ファイルには終端記号と非終端記号を定義し，それらを参照しながら BNF で構文規則と対応する意味動作を記述する．作成したファイルを Yacc で処理すると，内部でオートマトン (実体としては構文解析表である) とスタックを生成し，構文解析関数がこれらを操作して構文解析を実行するため，ユーザはこの関数を呼び出すだけで構文解析が実行できる．

コンパイラの授業の演習などで Yacc を採用している例として，豊田高専専攻科情報科学専攻の「コンパイラ」[1] や岡山大学工学部情報系学科「コンパイラ」[2]，電気通信大学情報理工学域「言語処理系論」[3] がある．またプリンストン大学のコンパイラの講義の教科書である『最新コンパイラ構成技法』[4] では構文解析器を開発するために ML-Yacc を使用している．

1.2.　Yacc の周辺ツールの現状

Yacc の周辺ツールについて，理解の補助のためのツールとして Leon Aaron Kaplan による”yaccviso − a tool for visualizing yacc grammars”[5] や楠目勝利らによる「コンパイラにおける構文解析過程の視覚化」[6] が提案されている．これらのツールは Yacc ファイルに記述した文法を概観したり，Yacc が出力した構文解析器の動作を可視化したりする際に利用される．また，コーディング時の補助ツールとして，シンタックスハイライトを施すもの [7] や静的解析を行うもの [8] が存在する．しかし，これらは Visual Studio Code (VSCode) 拡張として提供されているソフトウェアであり，Vim や Emacs などの他の主要なエディタでは同等の支援機能が得られない．特に静的解析によるコーディング支援を提供するものは [8] のみであるが，これは現時点 (2024 年 2 月 8 日時点) で 73,000 回以上インストールされており，Yacc コーディング時の静的解析による支援機能の提供への需要が存在することが分かる．

図 1: 言語サーバを利用したコーディング支援

2.　問題と解決法

セクション 1.2 で述べたように，現時点では Yacc コーディング時の静的解析によるコーディング支援機能の提供を行うソフトウェアは VSCode 上でのみ動作し，他の主要なエディタでは同等の支援機能が得られない．

そこで本研究では，現在多くの主要なプログラミング言語の実装が存在する Language Server Protocol (LSP)[9] の仕様に則った言語サーバを Yacc 用に開発する．この言語サーバを利用することで，ユーザは使い慣れたエディタでコーディング支援機能を使うことができ，Yacc コーディング時の負担が軽減する．さらに，新しいエディタが現われた場合でも，エディタ開発者は LSP に則ってクライアントを開発するだけで Yacc コーディング時の支援機能をユーザに提供することができる．

特定のエディタでのみ記述ができる言語は潜在的なユーザグループを排除しているという指摘 [12] がある．そのため LSP を用いて多様なエディタに対応することで，Yacc ユーザやエディタ開発者の負担軽減だけでなく，言語の普及に貢献することもできると考える．なお，現在 Yacc よりも一般に使用されている Bison 3.8.1 の仕様を元に静的解析器を実装するが，Bison は Yacc との上位互換性のあるソフトウェアのため，便宜上 Yacc と表現している．

3.　LSP とは

LSP は 2016 年に Microsoft が発表したプロトコルで，エディタと言語サーバ間の通信方法を規定している (図 2)．現在は多くの主要なプログラミング言語が公式，非公式を問わず言語サーバを持っており [10]，プログラミング言語のコーディング環境を整備する際に

図 2: LSP 登場による言語とエディタの関係の変化 [11]

複数の拡張やプラグインをインストールする必要がないだけでなく，言語クライアントが存在するエディタならばどのエディタでも利用することができることが強みである．

近年，Go や TypeScript, Rust といった新しいプログラミング言語が注目を集め利用者が増えているが，エディタや IDE などの開発者がそれぞれのプログラミング言語用に補完や定義ジャンプ，ホバー時のヒント表示などの多くの支援機能を追加するには多大な労力を要する．従来は開発ツール毎に支援機能を開発する必要があり，M 個の言語に対して N 個のエディタが存在し，その実装の数は M×N であった．しかし LSP の登場により，M 個の言語サーバと N 個のクライアントのみで同様の機能が実現できるようになり，実装の数は M+N となった．このようなことから，LSP は言語のユーザとエディタ開発者双方の負担を軽減することができる．

本研究では LSP のバージョン 3.17 に準拠する．

4. Yacc 用言語サーバの開発

4.1. 開発概要

本研究では Yacc 用言語サーバを開発と複数の言語クライアントの設定を行う．また言語サーバについて，LSP に則ってリクエストやレスポンスを処理したり，ドキュメントを読み書きしたりするサーバとしての機能だけでなく，静的解析に基づく支援機能をを提供するために静的解析器も実装する．つまり，開発は主に 3 つのプログラムからなり，それは言語サーバと言語クライアント，静的解析器である．

表 1 に開発環境を示す．主要なエディタはすでに言語クライアントが実装済みであり，開発者は設定ファイルへの追記のみで対応できることが多いが，VSCode は言語クライアントのパッケージを用いて軽微な実装を行う必要があった．表 1 中の言語クライアントは VSCode の言語クライアントである．

表 1: 開発環境

	言語サーバ	言語クライアント
プログラミング言語	OCaml 4.14.0	TypeScript 5.3.3
ビルドシステム	dune 3.7.2	tsc 5.3.3
ランタイム	OCaml 4.14.0	Node.js v21.6.1
OS	Arch Linux x86_64	
CPU	Intel i7-10710U	

4.2. 言語クライアントの実装・設定

VSCode で動作する言語クライアントを VSCode 拡張として実装した．そこでは Microsoft が開発している vscode-languageclient という npm パッケージを使用し，VSCode の拡張機能としての設定や言語サーバとの接続方法などの設定を記述した．また，Vim と Emacs でも動作されるために，Yacc ファイルを開いた状態で言語サーバと通信を行うように設定ファイルに記述を追加した．

4.3. 言語サーバの通信

LSP では JSON-RPC を用いて通信をするが，通信方式は指定されていない．しかし LSP においては，同一のコンピュータでクライアントとサーバが動作することが通常であるため，今回は同一コンピュータ上で標準入出力による通信を確立することとした．実際に VSCode の言語クライアントのパッケージでは標準入出力だけでなく，パイプ通信やソケット通信，IPC 通信が選択できるが，Vim や Emacs の言語クライアントのプラグインの多くが標準入出力での通信のみをサポートしている．

LSP の仕様に則って，ドキュメント (ファイル) はクライアントではなくサーバ側のハッシュ表で管理することとした．ドキュメントが開かれると，クライアントはサーバにドキュメントの内容を含むドキュメント情報を送信する．このリクエストを処理する際に，サーバがドキュメント情報をハッシュ表に格納する．これ以降のドキュメントの編集では，差分のみがサーバに送信されるため，差分をハッシュ表に格納されている情報に適用することで，サーバはクライアントが開いているドキュメントと同期するように実装した．

4.4. 言語サーバのライフサイクル

言語サーバのライフサイクルを図 3 に示す．初期化と起動という 2 つの状態を保持し，これらの状態に応じてリクエストを受理してレスポンスを返却するか，状態を遷移させてエラーレスポンスを返却するかなどを判断する．

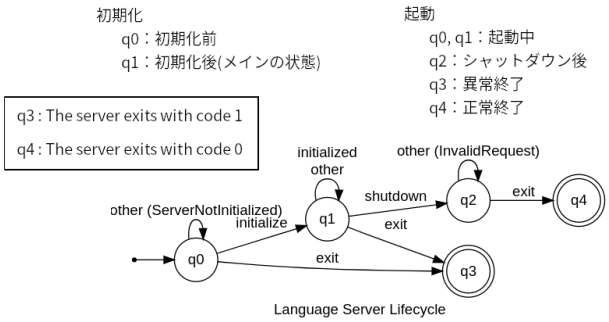

図 3: 言語サーバのライフサイクル

LSP の通信でやりとりされるメッセージには，request, response, notification の 3 種類がある．request は response を必要とし，notification は response を必要としないものである．クライアントとサーバのどちらからも，どのメッセージでも送受信できる．クライアントは支援機能に関する通信を行う前に，initialize request を送信する．これに対してサーバは response を返し，クライアントは initialized notification を送信する．この一連の通信でクライアントとサーバはそれぞれの capabilities を送り合う．これによって，双方はそれぞれが対応している支援機能などについて知ることができる．これが完了すると，サーバはクライアントから送信された補完や定義ジャンプなどの支援機能に関わる request を受理するようになる．

4.5. Yacc の静的解析器

Bison 実装の字句解析器と構文解析器の部分を ocamllex, Menhir を用いて OCaml への移植を行った．これによって Bison の仕様に則った字句解析器と構文解析器を得られた．また，構文解析の結果の出力として抽象構文木を得るように実装をすることで，静的解析が可能となった．

完全な入力を想定している通常のコンパイラと異なり，言語サーバは多くの場面で不完全なソースコードが入力として渡される．抽象構文木を走査して型検査や変数の定義と使用の検査などを行うため，どのような入力であっても構文木を得ることができなければ十分なコーディング支援を提供することが困難である．そのため，不完全なエラー回復をして抽象構文木を得る必要がある．エラー回復 (修復) 戦略としては，特殊な error 記号を使用した局所的なエラー回復や，Burke-Fisher エラー修復などが考えられるが，前者は対応できる状況に限界があり，後者は 2 つのスタックを使ってトークンを操作しなければいけないため実装が複雑になるという問題がある．そこで本研究では Menhir の Inspection API を利用することとした．Inspection API は LR 構文解析器のオートマトンやスタックの状態にアクセスすることができるため，比較的汎用なエラー回復が実現できる．

静的解析の流れを図 4 に示す．今回，実装した部分を青色の文字で表している．

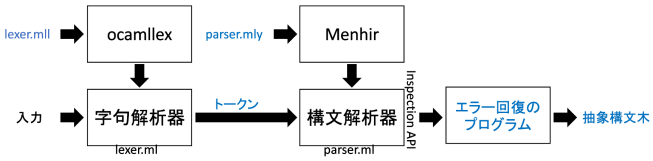

図 4: 静的解析の流れ

5. 実行例

コード診断を VSCode, Vim, Emacs で行っている様子をそれぞれ図 5, 図 6, 図 7 に示す．ここでは「%」という入力が不正であることと，予期しない位置 EOF があることを構文解析の過程で検出し，LSP の textDocument/didSave メソッドによってエディタに報告している．

図 5: VSCode でのコード診断実行例

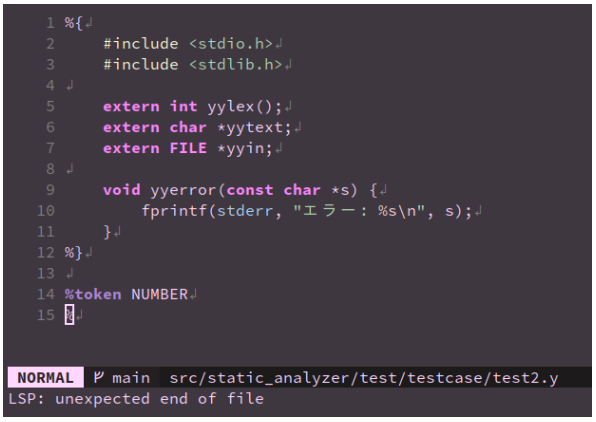

図 6: Vim でのコード診断実行例

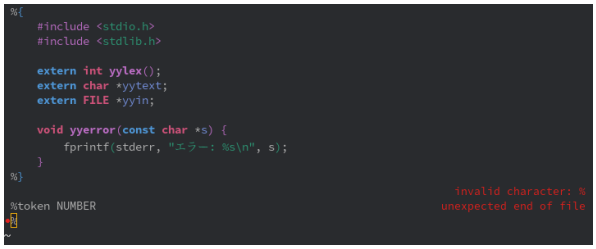

図 7: Emacs でのコード診断実行例

6. 考察

VSCode と Vim, Emacs の言語クライアントを使用し，今回作成した言語サーバを実行するように設定したことで，異なるエディタで同様の支援機能が容易に得られることが確認できた．また，Bison の仕様や実装を元に字句解析器と構文解析器を作成し，構文解析器の出力として抽象構文木を構築することで，静的解析を可能にした．今回は不完全な入力に対して可能な限りエラー回復を試みて抽象構文木を構築し，その過程で見つかった構文エラーなどを LSP の textDocument/diagnostic メソッドによってクライアントに送信している．

7. 今後の課題

今回記述したパーサジェネレータのファイルは Bison のものの移植のため，error 記号によるエラー回復を前提としており，文法に対応する意味動作に副作用がある場合が多いが，エラー回復のパターンを増やしたり，より安全にエラー回復を行ったりするためには，意味動作を副作用の無いものに変更する必要があると推察する．また，意味解析器を実装し，コード補完や定義ジャンプなどサポートする機能を増やして利便性を高めていく．その後，支援機能に関する計算を並列化するなどして効率化を図りたい．

このように問題を解決したり機能を追加したりすることで，言語サーバを十分実用に耐え得るものにしていき，OSS として公開しユーザの獲得や継続的な開発を実現したい．

8. 謝辞

本研究の遂行にあたり，指導教員として多大なご指導を賜りました稲垣宏教授 (豊田工業高等専門学校情報工学科)，ご多忙のなか相談に応じてくださりご助言を頂いた内山慎太郎氏 (豊橋技術科学大学大学院工学研究科 情報・知能工学専攻 博士後期課程 応用数理ネットワーク研究室) に感謝申し上げます．

文　献

[1] 高専 Web シラバス，“コンパイラ”，https://syllabus.kosen-k.go.jp/Pages/PublicSyllabus?school_id=23&department_id=25&subject_code=95018&year=2017&lang=ja, 2023 年 10 月 18 日閲覧

[2] Nobuya WATANABE，“コンパイラ（Compilers）”，http://www.arc.cs.okayama-u.ac.jp/~nobuya/lecture/compiler/, 2023 年 10 月 18 日閲覧

[3] 電気通信大学 シラバス Web 公開システム，“シラバス参照”，http://kyoumu.office.uec.ac.jp/syllabus/2023/31/31_21124118.html, 2023 年 10 月 18 日閲覧

[4] Andrew W. Appel，Modern compiler implementation in ML，Cambridge University Press，New York, Cambridge，2008.（アンドリュー・W・エイペル 神林靖・滝本宗宏 (訳), 最新コンパイラ構成技法, 翔泳社, 東京, 2020.）

[5] Leon Aaron Kaplan，“yaccviso − a tool for visualizing yacc grammars”, 2006.

[6] 楠目 勝利, 佐々 政孝．“コンパイラにおける構文解析過程の視覚化”, 全国大会講演論文集, 第 55 回, ソフトウェア科学・工学, pp448-449, 1997.

[7] Visual Studio Marketplace，“VSCode-YACC”，https://marketplace.visualstudio.com/items?itemName=carlubian.yacc, 2023 年 10 月 19 日閲覧

[8] Visual Studio Marketplace，“Yash”，https://marketplace.visualstudio.com/items?itemName=daohong-emilio.yash, 2023 年 10 月 6 日閲覧

[9] Official page for Language Server Protocol，“Language Server Protocol Specification - 3.17”，https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/, 2023 年 10 月 19 日閲覧

[10] Official page for Language Server Protocol，“Implemantations”，https://microsoft.github.io/language-server-protocol/implementors/servers/, 2023 年 10 月 19 日閲覧

[11] Language Server Extension Guide | Visual Studio Code Extension API，https://code.visualstudio.com/api/language-extensions/language-server-extension-guide, 2024 年 2 月 8 日閲覧

[12] Bünder, H. “Decoupling Language and Editor - The Impact of the Language Server Protocol on Textual Domain-Specific Languages.” International Conference on Model-Driven Engineering and Software Development, no.10.5220/0007556301290140, pp.129-140, Prague, Czech Republic, 2019.

[13] npm，“vscode-languageclient”，https://www.npmjs.com/package/vscode-languageclient, 2023 年 10 月 19 日閲覧

[14] Menhir Reference Manual (version 20231231)，“Inspection API”，https://gallium.inria.fr/~fpottier/menhir/manual.html#sec64, 2024 年 1 月 28 日閲覧