研究テーマ：コンパイラ コンパイラの設計と実装

０７５番　　小寺 広志

１．はじめに

コンパイラの標準的な設計は，以下の手順で行われる．
① 対象言語の字句解析･構文解析を実行する C コードを Lex，Yacc という２つのツールを用いて自動生成する．
② 対象言語の意味解析･コード生成を実行する C コードを人手で設計･実装する．

本研究では，言語として「算術式」を選び，①の工程を，ツールを用いずに設計･実装することを試みた．構文解析法には，適用可能な文法のクラスが LL 構文解析法，演算子順位構文解析法より比較的大きな LR 構文解析法を選び，その中でも最も単純な SLR(1)構文解析法を選んだ．

本研究の目的は，SLR(1)構文解析法の原理の理解と SLR(1)構文解析ルーチンの実装技術の習得にある．これらはソフトウェア設計の基礎として有用と考えられる．

２．課題設定

コンパイラ コンパイラとはコンパイラ設計支援システムのことで，本研究の場合，算術式の文法に従った文字列を SLR(1)構文解析するために必要な，動作表・行先表を自動生成するプログラムに相当する．課題は以下に示す通り 2 つ設定した．
① テキスト[1]を参考とした，SLR(1)構文解析ルーチンの実装．
② 構文木の描画アルゴリズムの考案と実装．

上記の課題で作成するプログラムの仕様は以下に示すように定義した．
入力：算術式の文法に従う有限長文字列
出力：入力文字列の構文が正しい場合 → 構文木の図 ；
　　それ以外 → 構文エラーの検出メッセージ
算術式：（生成規則が優先順位付けされ，曖昧性なし）
　P = { S→E, E→E+T|E-T|T, T→T*F|T/F|F, F→(E)|i }

３．データ構造とアルゴリズム

３．１ SLR(1)構文解析

概ねテキスト[1]に従ったが，解析対象の文法が左再帰を未除去の場合の first 集合の計算には，非終端記号チェックテーブルを導入して一工夫をした．即ち，非終端記号を１度展開(*)した時点で「展開した」という情報をテーブルに記し，そのテーブルを再帰呼び出し関数にコピーして渡し，テーブルに基づく判定で左再帰の無限ループを停止させる．その他の実装手法の説明は割愛する．
((*)展開例： E→E+T，E→E-T，E→T なら，
First(E) = First(E+T) ∪ First(E-T) ∪ First(T))

３．２ 構文木の描画

１）構文木スタックの導入

生成規則Ａ→αにより，(βα，γ)⇒(βＡ，γ)と還元する際，αは文字列βαの postfix であり，スタックを用いて還元操作を実現できる．一方，「還元」と「構文木の増築」は１対１に対応するため，構文木の増築もスタックを用いて実現できる筈である．通常のスタック操作における push は，既に確保してある記憶領域に対してデータを格納する．しかし，構文木は記憶領域を動的確保しながら増築するため，構文木スタックにおける push は「記憶領域確保＋データ格納」を行う．構文木スタックによる構文木のデータ例を図１に示す．

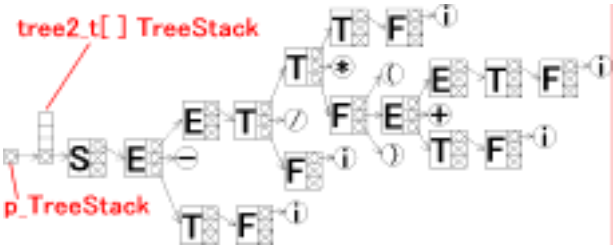

図１　i*(i+i)/i-i$ に対する構文木データ

２）構文木の横型探索

図１の構文木データから，「構文木を描画するために必要な情報」を抽出することを考える．ノード同士のリンク情報を保存するという条件下で，以下の 3 種の情報が必要と考え，queue を用いて構文木データから１世代毎の横型探索を行って抽出することにした（図２）．
① 各ノードの文字　；　② 各ノードにおける枝の数
③ ’i’ の順位（何番目の i かを判断するための数）

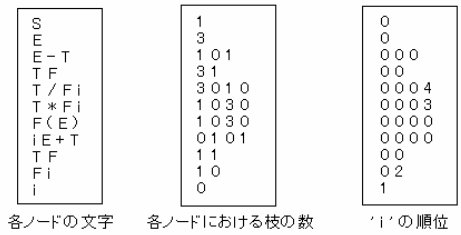

図２　抽出した構文木描画用のデータ族（図１に対応）

３）構文木の描画

図２のデータ族に基づき，各ノードの文字を配置する座標を，Ｓを原点とした相対座標で計算し，各座標を画面隅へとシフトし，ノード同士を線で結び，図３を得る．

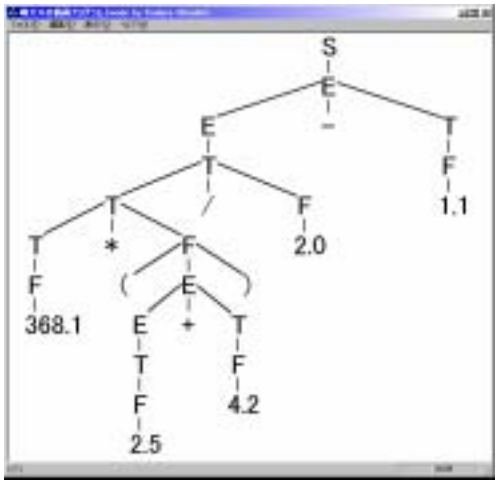

図３　描画した構文木（図１，図２に対応）

４．おわりに

今回，「SLR(1)構文解析」及び「構文木描画」のプログラムを実装した．課題は，適切なエラーメッセージの表示，より複雑な文法への対応である．自主課題研究の感想：プレゼンの時間は 10 分に延長すべきだと思う．

参考文献

[1] 湯淺 太一；「コンパイラ」p.76～112；昭晃堂