コンパイラにおける記号表処理部の自動生成系

4L−7

亀山裕亮[†]　中井央[†††]
山下義行[††]　中田育男[†††]

1. はじめに

コンパイラにおける字句解析器や構文解析器などの生成系の研究は多く行なわれているが (LEX[2],YACC[1], PCCTS[3] など)、記号表処理器の生成系に関する研究はこれまでほとんど行なわれていない。そのため記号表処理部はコンパイラの製作者が手書きで作成していた。

記号表処理の研究としては Pei らの提案した記号表ライブラリ[4] がある。4) ではスタックやハッシュといったよく使われるデータ構造をライブラリとして用意することで、様々なプログラミング言語で再利用可能な記号表ライブラリを提供している。

この方法ではコンパイラ製作者は対象とするプログラミング言語のスコープ規則を考慮した表のデータ構造を選択しなければならない。

しかし実際にはどの処理系においても記号表に対する操作は主に登録と検索だけであるので、実際の記号表のデータ構造を隠蔽しインタフェースのみを製作者に提供すれば、記号表処理部の製作の手助けになる。

そこで本論文では対象とするプログラミング言語の識別子や型の種類、スコープ規則などの記述 (以下記号表記述と呼ぶ) から記号表処理部を生成するシステムを提案する。生成される記号表処理部は登録と検索のインターフェースを持っていて、設計者は構文解析器記述の中からこれらを使用することで記号表のデータ構造を意識することなく登録、検索といった操作を行うことができる。本システムは Java[5] で記述された記号表処理器のプログラムを出力する。

Automatically generation of Symbol Table Manipulator for Compilers
† 筑波大学工学研究科
Doctoral Program in Engineering,University of Tsukuba
email: kame@lang.esys.tsukuba.ac.jp
†† 筑波大学機能工学系
Institute of Engineering and Mecanics and Systems,University of Tsukuba
††† 図書館情報大学
University of Library and Infomartion Science

本論文で提案するシステムの構成を図 1 に示す。

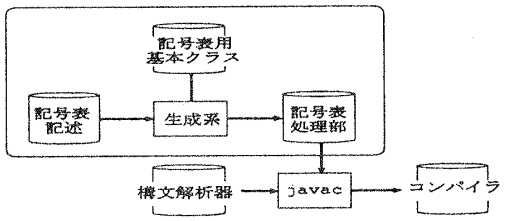

図 1　システムの構成

2. 記号表記述

記号表記述はキー宣言部、型宣言部、スコープ宣言部の 3 つの部分で構成されている。それぞれの部分で宣言するもの及びその役割を以下に示す。

キー宣言部　表を探索する際にキーとなる識別子の種類名 (変数や定数、型など) を宣言する。

型宣言部　意味チェックのために必要な、対象としているプログラミング言語に存在する型を、次の 4 つに分類して宣言する。

- **基本型**　論理型や数値型などの他の型から構成されることがない最も単純な型。
- **構造型**　既存の型から構成される型。配列、レコードなど。
- **関数型**　プログラムとして定義される関数や手続きの型。
- **抽象データ型**　C++や Java のクラスのような、データ構造とそのデータ構造を操作・解釈するための基本的な手続きをひとまとめにした型。

スコープ宣言部　名前の宣言と使用の対応を決定するために、対象としているプログラミング言語に存在するスコープに名前を付けて宣言する。

本システムでは汎用性の高い操作関数やメンバを持った基本クラスを用意しておき、記号表記述における処理系によって異なる部分は基本クラスから派生させて作る。この基本クラスについて以下で簡単に説明する。

記号表クラス　記号表全体を統轄するクラス。主にス

コープの管理を行う。

要素クラス　表の1つのエントリを表すクラス。メンバとして名前の綴、キー、型の情報を持つ。すべての表の要素はこのクラスから派生する。

型クラス　型の情報を持ったクラスで以下の4つのクラスを派生する。

- **基本型クラス**
- **構造型クラス**
- **関数型クラス**
- **抽象データ型クラス**

スコープクラス　スコープを操作するクラス。スコープ内の宣言を保持するメンバ変数及び表への登録用のメソッドや、スコープから出る処理を行うメソッドを持つ。

3. 構文解析器記述中での使用例

記号表クラスを構文解析器の中でどのように使用するのかをJavaのコンパイラを例として取り上げ、yacc風の構文解析器記述及び記号表に関する意味記述を用いて説明する。

記号表記述のキー宣言部で変数としてvarが宣言されていれば、要素クラスを継承しSYM_varクラスが作られる。変数の宣言部ではSYM_varクラスのインスタンスを作成し表へ登録することになる。

```
var_decl := modifier type var_decl_list semi;
{
  //$2 には型のエントリ、
  //$3 には名前のリストが入っている。
  while($3 の要素が無くなるまで)
  {
    Entry entry = SYM_var($3 のひとつの要素,$2);
       //変数のエントリの作成
    insert(entry);
       //表へのエントリの追加
  }
}
```

型宣言部で構造型としてarrayが宣言されていれば、構造型クラスを継承してSYM_arrayクラスが作られる。配列の宣言部ではSYM_arrayクラスのインスタンスを作成する。

```
array_type := primitive_type "[" "]"
{
  //$1 は型のエントリ
  $$ = new SYM_array($1);
      //array 型のエントリを返す
}
```

型宣言部で抽象データ型としてclassが宣言されていれば、抽象データ型クラスを継承してSYM_classクラスが作られる。クラスの宣言部ではSYM_classクラスのインスタンスを作成し、inheritメソッドを使用して親クラスを継承する。

```
class_decl := modifier CLASS ident EXTENDS ident
{
  Entry entry = new SYM_class($3);
       //クラスのエントリの作成
  entry.inherit($5);
       //$5 の親クラスのエントリを継承する
}
```

スコープ宣言部でメソッドの本体のような{···}で囲まれたスコープ(block)が宣言されていれば、SCOPE_blockクラスが作られる。SCOPE_blockクラスのインスタンスを作成することでスコープに入り、EndScope()関数を呼ぶことでスコープから出る。

```
method_decl := method_head
{
  new SCOPE_block();
     //block スコープに入る
}
"{" method_body "}"
{
  EndScope();
     //block スコープを抜ける
}
```

4. まとめと今後の課題

本論文ではコンパイラにおける記号表処理を対象とし、記号表処理器を記号表記述から生成するシステムについて述べた。このシステムを用いることにより、記号表のデータ構造を考えることなく登録、探索といった表の操作を行うことができるようになる。今後の課題として、意味チェックに関する記述を記号表記述に含めることにより、意味チェック処理部も自動生成したいと考える。

参 考 文 献

1) Johnson, S.: Yacc - yet another compiler compiler, Computing science technical report 32, AT&T Bell Laboratories (1975).
2) Lesk, M.: Lex - a lexical analyzer generator,Computing Science, Computing science technical report 39, AT&T Bell Laboratories (1975).
3) Parr, T.: *The Purdue Compiler Construction Tool Set*, version 1.10 release notes edition (1993).
4) Pei-Chi, W., Jin-Hue, L. and Feng-Jian, W.: Designing a Reusable Symbol Table Library, Tech. report csie-93-1010, National Chiao Tung University (1993).
5) 松永健司: Java クイックリファレンス, 株式会社オライリージャパン (1996).