意味解析器の自動生成系にむけて

Towards a Generator of Semantic Analyzer

多幡　充*　桑原 寛明†　國枝 義敏‡

1　はじめに

コンパイラ生成系を利用してコンパイラフロントエンドを作成する場合，字句解析部・構文解析部は現状でも機械的に生成できるが，意味解析部は通常は手作業で作成する必要がある [1]．例えば Yacc を用いて作成する場合，各構文規則に対して意味解析処理のための C 言語のコードをアクション部に記述する必要がある．SableCC ではまず目的のプログラミング言語の構文記述から，構文記述に対応する構文木のクラスと構文木の走査用クラスが生成される．この構文木を走査しながら意味解析を行うため，ユーザは構文木上の各ノードでのアクションとして Java のコードをクラス内に追記する必要がある．

そこで本研究では，コンパイラ生成系においてこれまで手作業で作成していた意味解析を行うコードを自動的に生成し，コンパイラ全体の作成にかかる手間を軽減することを目的とする．このときユーザが作成する意味記述はコンパイラ生成系に依存しない形とし，ユーザは任意のコンパイラ生成系を選択できることを目指す．

2　意味解析器生成系

プログラミング言語によらず共通的に必要な意味解析処理として名前の定義と参照の対応付けがある．その処理は記号表への名前の登録と検索からなる．一方，記号表へ登録される情報やスコープルールに基づいて記号表を検索する方法は言語ごとに異なる．意味解析器生成系を実現するためには，意味解析処理において対象言語に依存する部分と依存しない部分を明らかにする必要がある．

本研究では，記号表の準備，スコープの定義，名前の登録と検索を言語に共通な処理とみなし，これらを記述するための意味解析プリミティブを表 1 の通り定義する．意味解析プリミティブを用いて，記号表の内容やスコープルールなど言語ごとに異なる情報や，名前の登録や検索など各構文規則において行われるべき処理を記述する．各意味解析プリミティブに対し具体的なコードへの変換も定義しているため，コンパイラ開発者は意味解析プリミティブを適切に組み合わせて，構文規則に対応するアクション部へ宣言的に記述するだけで目的の意味解析部が得られる．

表 1　意味解析プリミティブ

declareTable	記号表を作成する．
declareColumn	記号表の要素を定義する．
addColumn	記号表に要素を追加する．
scopeEnter	スコープの階層を変更する．
register	名前の文字列表現や型などを記号表に登録する．
search	スコープ規則に従い，記号表から名前を検索する．
return	構文木の親ノードへ情報を渡す．

*Mitsuru Tabata, 立命館大学大学院 理工学研究科

†Hiroaki Kuwabara, 立命館大学 情報理工学部

‡Yoshitoshi Kunieda, 立命館大学 情報理工学部

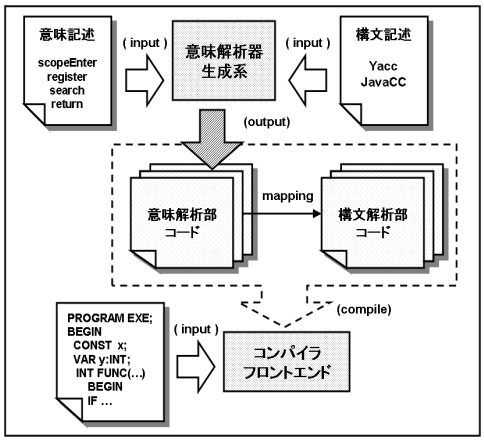

図 1　意味解析器生成系の概要

```
1 /* 記号表の定義 */
2 declareTable ( CONST_TABLE, VAR_TABLE, FUNC_TABLE );
3 declareColumn  type  (INTEGER, CHAR);
4 addColumn [VAR_TABLE]  type;
5 addColumn [FUNC_TABLE] type;
6 scopeEnter ( $block );
7 %%
8 program   : PROGRAM  IDENTIFIER
9   { register[ FUNC_TABLE ](name:$IDENTIFIER.string); }
10              ';'  block  '.'
11              ;
12
13 block     : constdcl vardcl func_list _BEGIN stmt_list END
14            ;
15
16 vardcl    :
17              | VAR  IDENTIFIER  ':' basictype ';'
18   { register[ VAR_TABLE ](
19       name: $IDENTIFIER.string, type: $basictype.type ); }
20              ;
21 basictype : INTEGER { return(type:INTEGER); }
22              | CHAR      { return(type:CHAR); }
23              ;
```

図 2　Yacc を用いた意味記述のコード例

図 1 に意味解析器生成系の概要を示す．本生成系は意味記述及び構文記述をもとに，任意のコンパイラ生成系に対応する意味解析部のコードを自動的に生成する．ユーザはこの意味解析部のコードとコンパイラ生成系が生成した構文解析部のコードを合わせてコンパイルすることで目的のコンパイラフロントエンドを得ることができる．本生成系が対象とするプログラミング言語は手続き型言語とし，スコープルールは多くの手続き型言語で採用されている入れ子型ブロック構造を対象とする．

図 2 に実際に Yacc を用いて，Pascal のサブセットの一種である KPL と呼ばれる手続き型言語の意味記述の例を示す．{} で囲まれた Yacc のアクション部に記述されたプリミティブは，本生成系によりそれぞれ意味解析を行うコードに変換される．例えば，図 2 の 18 行目では登録のプリミティブ register を利用しているが，ここは図 2 の 2 行目で宣言された記号表 VAR_TABLE に名前と型の情報を登録するコードに変換される．また，図 2 の 21 行目では親ノードに情報を渡すプリミティブ return を利用しており，ここは型の情報を渡すコードに変換される．

3　今後の課題

コンパイラにおける意味解析処理の自動生成を目的として，意味解析処理を記述するためのプリミティブを定義した．定義した意味解析プリミティブを用いてプログラミング言語 KPL を例に，その意味解析処理をコンパイラ生成系 Yacc のアクション部中に自動生成できることを確認した．しかし，これだけでは意味解析器生成系として機能的に不十分であり，以下の点は今後の課題である．

- デフォルトで記号表に登録できる情報は名前の文字列表現のみである．そこで，ユーザ自身が記号表を拡張し，参照するための枠組みを用意する．なお，現状の意味記述では記号表に要素を追加するためのプリミティブを用意しているが，追加した要素を参照したり，親ノードに渡すなどの操作を想定していない．
- 上記のユーザによる拡張とは別に，記号表に登録できる情報に名前の型を追加し，汎用の型検査を行うための意味解析プリミティブを用意する．
- 構文記述から独立的に記述された意味解析プリミティブをそれぞれ対応する構文木ノードへマッピングするための枠組みを用意する．

参考文献

[1] 亀山裕亮, 山下義行, 中井央, 田中二郎. コンパイラのための意味解析器生成系. 日本ソフトウェア科学会第 18 回大会論文集，2001.