4N-3

コンパイラフロントエンド生成系

舞田純一 †　　中井央 ††
筑波大学　　筑波大学

1　序論

コンパイラフロントエンドとは、コンパイラの内、字句解析器、構文解析器、意味解析器等のマシン非依存部の総称で、最適化、コード生成等のバックエンドを実行する前段階である。

コンパイラの作成には多大な労力が必要であり、この労力の削減のため、一般には形式的な記述 (仕様) からコンパイラの実装を生成するツールが用いられている。

字句解析器、構文解析器の自動生成に関する研究は実用となるレベルまで進んでおり、lex [1]、yacc [2] といったツールが広範に用いられている。意味解析器の自動生成に関しては属性文法等の研究がされているが、型検査等についてはサポートが少なく、十分に実用化されているとは言い難い。

本研究では、型検査に特化した機能を持つ意味解析器自動生成系を中心とした、コンパイラフロントエンド全体の自動生成のための生成系について考察し、実際に利用可能なツール群を作成した。

2　システムの概要

2.1　特徴

本システムは以下のような機能を持つ。

- 型推論規則による型推論、型検査
- 記号表への記号の登録、参照
- 記号表のスコープ管理
- Visitor パターンによる多パス意味解析

本システムは型を主体として設計され、基本的な型理論を基礎理論とし、型式、型変数 (総称、非総称)、型推論規則、単一化、代入等を用いることで型検査を簡潔に記述できるようにした。これらの概念については参考文献 [3] [4] を参照されたい。また、そのために必要な型システム、記号表の実行時ライブラリを用意している。

本システムでは、Visitor パターン [5] による構文木に対する巡回によって意味解析を実行する。そのため、型推論規則と構文木の対応付けや型式に対するバックパッチ等の仕組みを用意した。

2.2　構成

本システムはオブジェクト指向スクリプト言語 Ruby[6] で作成されており生成するコードもまた Ruby コードである。システムはプリプロセッサと意味解析器生成系から構成され、Racc[7] という字句解析器、構文解析器生成系と協調する形でコンパイラフロントエンドを生成する。

プリプロセッサは、与えられた記述から Racc 用の文法定義と本システムが処理する意味解析器生成系用の記述を分離する。そして、分離した文法に対し構文木を作成するアクションを構文規則毎に挿入する。また、構文木を作成するために必要なノードを表すクラスの実装コードを生成する。

意味解析器生成系では、形式的記述から意味解析器を生成する。意味解析器生成系に与える記述には、型の定義、ノードへの型の設定、型と型の等価検査、スコープの管理等を記述できる。

生成されたコンパイラフロントエンドは、構文解析器により生成された構文木に対し、意味解析器が巡回することで意味解析処理を行う。

Compiler Frontend Generator
†Junichi Maita・University of Tsukuba
††Hisashi Nakai・University of Tsukuba

3 記述例

3.1 型の定義

多くの静的型付のプログラミング言語にはあらかじめ用意されている型が存在する。これにはintやfloat等の基本型やポインタ等がある。また、基本型に適用できる関数(演算子を含む)が存在する。

例えば、基本型にint、floatがあり、それらが互いに代入、加算可能であるとすると、本システムへの記述は以下の様になる。

```
def NumType
  { int , float }
  ?+ : NumType * NumType -> MAX()
  ?= : NumType * NumType -> ARG1()
```

int、floatは、加算、代入の関数に関し関連があるのでNumTypeとしてグループ化し定義している。グループ化された型は記述順により型のサブセットとして順序付けられる。この例ではint<floatと順序付けされる。

また、それらの型に対する加算、代入関数を定義している(?は特殊文字をエスケープする)。この中で、MAX()は関数の引数群の型の中で最も大きいものを表し、ARGn()はn番目に現れる引数を表す。この例では引数がint × floatなら、+の返り値はfloatになり、=の返り値はintになる。

3.2 変数宣言

変数宣言は以下の様に記述される。上2行は構文規則であり、下2行%で始まる行は意味規則である。意味規則の初めの.p1は、パスを示すラベルである。$で始まる識別子は構文規則のそれと対応する。

この記述では、二重登録防止のためidentが表す文字列の記号が記号表に登録されているか調べ、存在しなかったらtypeの型で記号を記号表へ登録している。

```
var_decl :
  type IDENT- ';'
  % .p1: ! env $ident
  %    - env $ident := $type
```

3.3 演算

演算(ここでは加算)は以下の様に記述される。

これは、引数の型がexprの型× termの型、返り値の型が任意の何か(型変数T)である関数+が定義されているか検査し、検査の結果が成功であればこのノードの型を型変数Tとしている。

```
expr :
  expr '+' term
  % .p1: env "+" == $expr * $term -> @T
  %    - @T
```

4 結論

本研究では型を主体としたコンパイラフロントエンド生成系を研究し、簡潔に型に関する事項を記述できる意味解析器生成系を中心としたシステム及びそれにより生成されたコンパイラフロントエンドが用いる実行時ライブラリを開発した。また、本システムの適用例として手続き型言語としてはC言語のサブセット、型推論を行う関数型言語としてはMLのサブセットの意味チェッカを作成した。

今後の課題としてはオブジェクト指向に対応するためのレコード型の拡張や中間コード生成への対応等が挙げられる。

参考文献

[1] M. E. Lesk. Lex—A lexical analyzer generator. Technical Report CS-39, AT&T Bell Laboratories, Murray Hill, NJ, USA, 1975.

[2] Steven C. Johnson. Yacc: Yet another compiler compiler. In *UNIX Programmer's Manual*, Vol. 2, pp. 353–387. Holt, Rinehart, and Winston, New York, NY, USA, 1979. AT&T Bell Laboratories Technical Report July 31, 1978.

[3] 米澤明憲, 柴山悦哉. モデルと表現. 岩波書店, 1992.

[4] Bertrand Meyer. プログラミング言語理論への招待. アスキー, 1995. 酒匂寛 訳 二木厚吉 監訳.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1994.

[6] まつもとゆきひろ, 石塚圭樹. オブジェクト指向スクリプト言語Ruby. アスキー, 1999.

[7] 青木峰郎. Rubyを256倍使うための本 無道編. アスキー, 2001.