A_017

Rubyと拡張可能構文解析器生成系によるCOINSを用いたコンパイラの自動生成

舞田純一[†] 佐藤聡[†] 中井央[†]

Compiler Generation for COINS with Ruby and An Extensible Parser Generator

Jun'ichi Maita,[†] Akira Sato[†] and Hisashi Nakai[†]

1. 序論

著者らは現在、柔軟なコンパイラ開発の実現のため、自己拡張可能な構文解析器生成系[1)2)]を開発している。この生成系では、Yaccでいうところのアクションの機能やCST構築機能、AST構築機能、字句解析器生成機能などを拡張(生成系用のモジュール)として生成系に個別に追加出来るようにし、利用者が自分の利用したい機能を選択してコンパイラを実装できる。また、生成系を用いて利用者が新たな機能を拡張として実現でき、生成系自身を機能強化できる。

一方でこの生成系を用いた場合、現状では意味解析処理やコード生成処理は利用者自身が全て実装する必要があった。

本研究では、この自己拡張可能な構文解析器生成系において、意味解析実装及びCOINSインフラストラクチャ[3)4)]によるコード生成実装を支援するフレームワークを開発し、これを用いてコンパイラを作成する方法を提案する。

生成系はオブジェクト指向スクリプト言語Ruby[5)]を用いて実装され、生成されるコードもRubyであるため、本フレームワークもRubyを用いて実装した。

2. 概要

本フレームワークは以下の機能から構成される。

- 記号表と型検査のための枠組み
- COINSインフラストラクチャによるHIRの構築・コード生成関数

2.1 Rubyコードからの COINSの利用

並列化コンパイラ向け共通インフラストラクチャCOINS(以下単にCOINSと記す)は、高性能なコンパイラバックエンドを構成するために、高水準中間表現HIRと低水準中間表現LIRを用いて構築された木に対する最適化・コード生成を行なうためのライブラリ群を提供する。

COINSには、HIR操作インタフェースが用意されているが、これはJavaにより実装されている。これをRubyコードから利用するため、本フレームワークではRubyJavaBridge rjb[6)]を用いる。

2.2 記号表・型検査

COINSの記号表・型検査機構は、手続き型言語を実装する目的の制限されたものであるため、本ライブラリでは、柔軟なスコープ構造を表現できる記号表と、論理式に基づく型検査フレームワークを独自に実装している。

これらと拡張可能構文解析器生成系のAST拡張を併用することで、例えば型推論[7)]やレキシカルクロージャを容易に実現できる。

2.3 HIRの構築

前述の通り、本ライブラリではCOINSのHIR操作インタフェースを利用してHIRを構築できる。

HIRには以下のノードが用意されており、これらと記号表を組み合わせてプログラムを表す木を構築する。

式　四則演算や比較演算、アドレス演算など
文　制御構造(if、whileなど)やブロックなど
その他　変数や定数など

COINSによるHIRの構築では記号表と記号に関

† 筑波大学
University of Tsukuba

```
Stmt stmt =
  hir.returnStmt(hir.intConstNode(0));
BlockStmt mainBlock =
  hir.blockStmt(null);
mainBlock.addLastStmt(stmt);
Subp mainSubp = sym.
  defineSubp("main".intern(),symRoot.typeInt);
mainSubp.setVisibility(Sym.SYM_PUBLIC);
SymTable lSymTable =
  symRoot.symTableCurrent
    .pushSymTable(mainSubp);
mainSubp.closeSubpHeader();
SubpDefinition mainSubpDef =
  hir.subpDefinition(mainSubp,
     symRoot.symTableCurrent);
mainSubpDef.setHirBody(mainBlock);
((Program)hirRoot.programRoot)
  .addSubpDefinition(mainSubpDef);
```

図 1　java による HIR 構築例

```
define(:subp, 'main', vPUBLIC, tINT) {
  sBLOCK( sRETURN( nCONST(0) ) )
}
```

図 2　Ruby と本フレームワークによる HIR 構築例

する SymRoot、Sym、HIR 木とそのノードに関する HirRoot、Hir などの複数種類のインタフェースを使いこなす必要があるが、本フレームワークではこれらを 1 つのモジュールにラップし、関数的に利用できるようにしている。これにより、HIR の構築を容易にした。

`int main(){return 0;}` という C コードの HIR の構築について、図 1 に java によるコード例 (断片) を、図 2 に本フレームワークによるコード例 (断片) を示す。

3. 適　用　例

図 3 に、本フレームワークによる簡易電卓コンパイラの記述例を示す。この例では生成系のアクション拡張を用い、生成規則の各右辺の{…}はアクション部であり、その中の `val[n]` は右辺の記号の値 (Yacc での `$n`) を表す。ここではアクション拡張を用いたが、より複雑な場合は AST 拡張を用いて抽象構文に対して意味を記述することなどもできる。

この例では、四則演算の生成規則に対応するようにアクションで HIR を構築している。`eOP_ADD`、`eOP_SUB`、`eOP_MUL`、`eOP_DIV`、はそれぞれ加算、減算、乗算、除算のための HIR ノードを構築する関数である。そして、プログラム全体は `p` を左辺とする生成規則のアクションでまとめ、ここでエントリポイントとなる `main`

```
…字句解析部等省略…
p: e
 { define(:subp,'main',vPUBLIC,tINT)\
     { sEXP(val[0]) }
   gen_code()
 } ;
e: e + e { eOP_ADD(val[0], val[1])}
 | e - e { eOP_SUB(val[0], val[1])}
 | e * e { eOP_MUL(val[0], val[1])}
 | e / e { eOP_DIV(val[0], val[1])}
 | NUM   { nCONST(val[0]) } ;
```

図 3　本手法によるコンパイラ記述例

関数を定義し、`gen_code` 関数でコードを生成している。関数定義は `define` 関数を用い、その本体はブロックとして与える。COINS では、関数定義の本体となるのは文を表すノードのみであるため、ここでは `sEXP` 関数を用いて式文を構築している。

4. 結　　論

本フレームワークと自己拡張可能な構文解析器生成系をともに利用することで、COINS を利用するコンパイラを容易に実装することが可能になった。本手法により、pl0' コンパイラを実装したが、これは Java と JavaCC によるものに比べ約 7 割程度の行数で実現できた。また、本手法により、型推論やレキシカルクロージャを扱う言語のコンパイラを実装することが可能なことも確認した。

今後の課題として、現在は Ruby 用ライブラリとして実装されている本フレームワークを自己拡張可能な構文解析器生成系のための拡張として実装すること等が挙げられる。

参　考　文　献

1) 舞田純一, 佐藤聡, 中井央. 機能拡張可能なコンパイラ生成系, 2006. 情報処理学会第 58 回プログラミング研究会.
2) 中井研究室.　拡張可能構文解析器生成系. http://nakai2.slis.tsukuba.ac.jp/hiki/.
3) 中田育男他. 21 世紀のコンパイラ道しるべ・・COINS をベースにして. 情報処理, Vol.47, 4〜7, 2006.
4) coins 開発グループ.　COINS - 並列化コンパイラ向け共通インフラストラクチャ. http://www.coins-project.org/.
5) 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.
6) arton. Rubyjavabridge. http://arton.no-ip.info/collabo/backyard/.
7) 龍田真. 型理論. 近代科学社, 1992.