A-028

自己拡張可能な構文解析器生成系における構文解析手法の選択機能の実現
Implementation of Selecting Parsing Method on Self-Extensible Parser Generator

舞田 純一 † **中井 央** † **佐藤 聡** † **長谷川秀彦** †

Jun'ichi Maita Hisashi Nakai Akira Sato Hidehiko Hasegawa

1. 序論

既存の生成系を用いてコンパイラフロントエンドを作成する場合、新たな機能を追加、評価するには、その生成された構文解析器または生成系自体を直接改造する必要があったが、これには生成される構文解析器または生成系の内部構造を熟知する必要があり、決して容易なものではなかった。

そのため、我々はオブジェクト指向に基き、純粋な構文解析機能のみをもつ構文解析クラスに対して、ユーザの目的に応じて機能を付加できる構文解析器の構成方法を提案した [1]。この構成方法では、Decorator パターン [2] および Template Method パターン [2] を用い、構文解析器に対する付加機能をデコレータとして実現することで、構文解析器本体を直接改造することなく様々な機能を容易に付加できる。

しかし、いわゆるアクションや AST の構築といった付加機能を利用したい場合、それらの付加機能には生成される構文解析器に依存した部分がある (例えば、一般にある構文解析器のアクションはその構文解析器のみで用いられるはずである)。そのため、生成系の側もそれに必要な入力を受け取り、その構文解析器のための付加機能を生成する必要がある。この観点から、我々は生成系自体を拡張可能とし、必要とする任意の入力を処理可能とする構文解析器生成系を実現した [3][4]。

本生成系では、いわゆるアクション機能、CST 構築機能、AST 構築機能、字句解析機能などに必要となる入力記述に対して本生成系を対応させるための機能を、拡張 (生成系用の機能モジュール) として本生成系に個別に追加出来るようになっている。これにより、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、本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。本生成系により生成された構文解析器は、前述の構成法を用いて機能付加が可能となっており、特定の機能が実装されたデコレータを利用することで、利用者の要求に合わせ、機能を取捨選択できるようになっている。前述の拡張機能は基本的にデコレータを生成することで、生成される構文解析器への機能付加を実現する。また、この機能付加機能により、生成された構文解析器に対し、任意の誤り回復やデバッグ情報表示機能などを持たせることができる。

一方で、本生成系を用いた場合、生成される構文解析器の構文解析法は現状では LALR(1) 構文解析法に限定されていた。しかし、構文解析の手法・クラスも多様であり、利用者の要求に合わせたものが使用できることが望ましい。

†筑波大学
University of Tsukuba

```
 1 %class Sample based_on ...
 2 %extend Lexer ('depager/lex.rb')
 3 %extend NVAction ('depager/nvaction.rb')
 4 %decorate @NVAction
 5 %%
 6 %LEX{
 7   /\s+/, /\#.*/ { }
 8   /[1-9][0-9]*/ { yield _Token(:NUM, $&.to_i) }
 9   /./           { yield _Token($&, $&) }
10 %}
11
12 #begin-rule
13   expr:
14       term expr_     { "#{_term} #{_expr_}" }
15   ;
16   expr_:
17                      { }
18     | '+' term expr_ { "+ #{_term} #{_expr_}" }
19   ;
20   term:
21       fact term_tail { "#{_fact} #{_term_}" }
22   ;
23   term_tail:
24                      { }
25     | '*' fact term_ { "* #{_fact} #{_term_}" }
26   ;
27   fact:
28       NUM            { _NUM.value }
29     | '(' expr ')'   { _expr }
30   ;
31 #end-rule
32 %%
33 parser = createDecoratedSample
34 r, = parser.yyparse(STDIN)
35 puts r
```

図 1: sample.dr

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、LALR(1) 解析法以外の任意の解析法で構文解析器を生成するための手段を実現し、その実装例として、予測型構文解析表を用いた LL(1) 構文解析法 (以下、単に LL(1) 法と記す) および再帰下降法による構文解析器の生成を実現した。

2. 本システムの概要

本生成系への入力は、宣言部、文法部、拡張部から構成される。宣言部には、生成される構文解析器のクラス名、利用する拡張、利用するデコレータなどを記述する。文法部には、生成する構文解析器の基となる文法を記述する。拡張部は、文法部中にあり、利用する拡張が要求する入力を記述する。

今回実現した、構文解析法の選択機能を利用する場合、宣言部に以下のように記述する。

```
%class <構文解析器クラス名> based_on
  <構文解析法> (<ファイル名>)
```

<構文解析器クラス名> には、生成される構文構文器のクラス名を指定する。<ファイル名> には、構文解析法が定義されているファイル名を指定する。構文解析法には現状では以下のものが選択できる。

- LALR::SingleParser
- LALR::ExtensionParser
- LL::SingleParser
- RDP::SingleParser

LALR::SingleParser では LALR(1) 法による構文解析器を生成する。LALR:: ExtensionParser では LALR(1) 法による本生成系への拡張機能を生成する。本生成系の拡張は、本生成系への入力記述の拡張部に対する処理系 (構文解析器) とみることができ、これを指定することで、本生成系自身を用いて本生成系への拡張を実装することができる。LL::SingleParser では LL(1) 法による構文解析器を生成する。RDP::SingleParser では 再帰下降法による構文解析器を生成する。これら以外にも、利用者が新たに構文解析法を定義することにより、任意の構文解析法を利用することができる。本生成系への入力例を図 1 に挙げる。この例では、1〜5 行目が宣言部であり、6〜32 行目が文法部・拡張部である。1 行目で . . . となっている部分に、前述の構文解析法を指定することができる。この例では、字句解析拡張といわゆるアクション拡張を用いているが、これは構文解析法に依存しないので、構文解析法によって拡張を使い分ける必要がない。

3. 実装

本生成系は宣言部、文法部を処理するため宣言部パーザ、文法部パーザにより構成される。以前は文法部パーザが入力となる文法を処理した後、直接 LALR(1) 構文解析表を構築し、コードを出力していたが、本研究では、様々な構文解析法に対応するために、この部分をモジュール化し、そのモジュールを生成系への入力記述から指定できるようにした。これにより、ある構文解析法に基づいた表構築・コード生成モジュールを実装することで、容易に生成系をその解析法に対応可能にすることができるようになった。この機構を用いて、本研究では今回 LL(1) 法および再帰下降法による構文解析器の生成を実装した。解析法自体に関しては参考文献を参照されたい [5]。

生成された構文解析器は前述の構成法に基づき、デコレータによる機能付加が可能である必要がある。この構成法では任意個のデコレータオブジェクトが、構文解析器オブジェクトを装飾する。そして構文解析器オブジェクトが特定の処理を行うタイミングに合わせて、デコレータオブジェクトのメソッドがフックされる。このときメソッドは装飾する外側から内側に向かって順に呼び出される。これにより、デコレータは付加機能を実現する。デコレータのメソッドがフックされる、特定の処理を行うタイミングとは、シフト、還元、受理、エラーのそれぞれの前後である。これはボトムアップ型の構文解析法を前提としたものであるが、LL 法などのトップダウン型の解析法では、シフトや還元といった処理は直接には存在しない。そのため、LL(1) 法や再帰下降法の構文解析器の実装では、スタックへの終端記号のプッシュや新たな先読みの直前をシフトの代替とし、ある生成規則の処理の終了を還元の代替とした。これにより、従来の LALR 法による構文解析器と拡張レベルでの互換性をある程度保つことに成功した。

デコレータは、構文解析法に依存するものと依存しないものが存在する。例えば、解析スタックの状況を表示するデコレータはそれぞれの解析法により全く構造が異なるが、AST 構築機能や CST 構築機能は還元時アクション機能の派生として実装されているため、還元時アクション機能のみがそれぞれの解析法に対応して実装されれば、そのまま利用できる。このため、本生成系では、多くの拡張・デコレータが LALR(1) 法、LL(1) 法、再帰下降法に関わらず利用できる。

4. 結論

本研究により、さまざまな構文解析法を容易に生成系に追加し、評価することができるようになった。また、本生成系が標準で用意する様々な拡張機能を、可能であれば流用することで、新たな解析法においてもある程度実用的なコンパイラ生成系として本生成系を用いることができる。

今後の課題としては、EBNF による文法記述への対応やより多くの解析法モジュールの実装、またトップダウン型の解析法に対してより利用しやすいように拡張機能を改良することなどが挙げられる。

参考文献

[1] 佐竹力, 中井央. オブジェクト指向に基づいた構文解析器構成法の提案. 情報処理学会論文誌：プログラミング, Vol. 45, No. SIG.12 (PRO 23), pp. 25–38, 2004.

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

[3] 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. 情報処理学会論文誌：プログラミング, Vol. 47, No. SIG.16 (PRO 31), pp. 1–9, 2006.

[4] 中井研究室 Hiki. 拡張可能構文解析器生成系. http://nakai2.slis.tsukuba.ac.jp/hiki/.

[5] A. V. Aho, R. Sethi, and D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, Mass., 1986.