無制限に変更・拡張が可能なコンパイラの設計・実現方法の提案

7G－4

一杉 裕志　平野 聡　田沼 均　須崎 有康
電子技術総合研究所

概要

プログラミング言語の言語仕様やコンパイル方法等を幅広い範囲で変更・拡張可能なコンパイラを設計・実現する方法について述べる。コンパイラを構成するすべての部品をオブジェクトとして定義し、ユーザプログラムのソース中からこれらの部品を新しい部品と取り替える手段を提供することによって言語仕様の大幅な変更・拡張を可能にする。コンパイラはオブジェクト指向スタイルによって拡張性を強く意識して記述され、追加機能は差分として記述できる。

1　はじめに

従来、言語の拡張機能はマクロなど制限された形のものがほとんどであった。また、言語機能の幅広い拡張を可能とする手法としてリフレクション[1, 2]が提案されているが、主に実行時の振舞いを変更することを目的としているため、コンパイル方法のきめ細かい変更をユーザが指定することは不可能である。

我々は、言語仕様およびコンパイル方法を非常に幅広い範囲で変更・拡張可能な、手続き型言語のコンパイラ Lods の設計を行なっている。Lods は以下のことを目指している。

1. 言語機能の無制限な変更・拡張を許す。すなわち、元の言語仕様に対して上位互換になるような拡張にとどまらず、元の言語仕様の一部を犠牲にして新しい機能を実現することも許す。元の言語とは全く異なった言語に変更することも可能とする。

2. 拡張機能を、元の言語機能に対する差分として記述できるようにする。これにより第三者が拡張機能をライブラリの形で提供可能になる。ユーザは複数の拡張機能ライブラリの中から欲しい言語機能を選んで使うことができる。

3. 局所的な言語機能の変更・拡張が行なえるようにする。例えば、関数単位、構文単位、変数単位など、きめ細かい単位での変更・拡張を許す。これにより、プログラムの各部分ごとに最適な言語仕様を選ぶことができる。

A Proposal of Design and Implementation of Widely Extensible Compilers by Yuuji ICHISUGI, Satoshi HIRANO, Hitoshi TANUMA, Kuniyasu SUZAKI (Electrotechnical Laboratory).

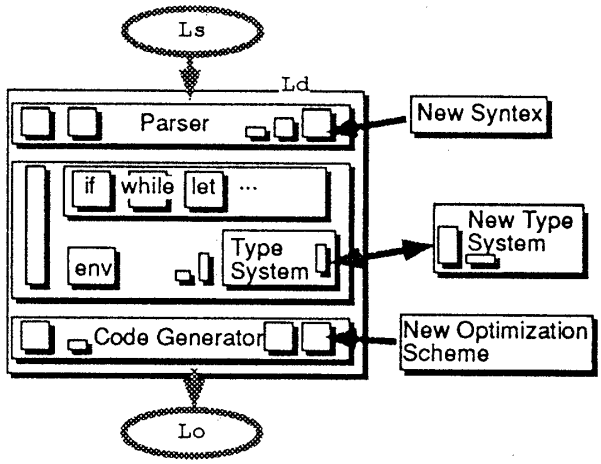

図 1: コンパイラの部品の交換・追加

4. 他の言語の処理系を実現する際に利用できる部品を提供する。例えば型システム、オブジェクト指向言語におけるクラスシステム、並列実行のためのモジュール、ソースレベルデバッガのモジュールなどを、再利用性を意識した形で実現する。これにより、新しい言語の実装が容易になる。

以上の性質によって、言語の進化の速度を早め、新しいプログラミングパラダイム、新しい応用分野に素早く適応できる言語を実現可能にする。

2　言語 Ls,Ld,Lo

Lods では、拡張の対象となる言語を Ls、Ls のコンパイラおよび実行時ルーチンを記述する言語を Ld、コンパイル結果として出力する言語を Lo と呼んでいる。

Ls としては、最初は単純な手続き型言語を想定しているが、機能拡張パッケージを追加することによってオブジェクト指向、並列実行などの機能を持たせることができる。

記述言語 Ld には、再利用性や動的拡張性を意識したシステム記述をサポートするオブジェクト指向言語を用いる。具体的には、仕様宣言機構（後述）、システムを構成する部品を動的に他の部品と交換する機構、古い部品の内部状態を新しい部品に引き継ぐためのプロトコルを規定する手段などを提供する。

目的言語 Lo には、特殊な仮定は置いていない。広い範囲のターゲットマシンに移植可能になるようにコンパイラを

設計する。

3 コンパイラの構造

Ld によって記述される Ls コンパイラは、メインルーチン、各構文の意味定義、変数名や関数名の名前表など多数のオブジェクトから構成される。そして、 Ls のユーザプログラム自身から Ls コンパイラの部品に対して操作を加えたり、動的にユーザが定義したオブジェクトに置き換えたりするための構文を Ls に導入することにより、ユーザによる Ls コンパイラの変更・拡張を可能にする（図1）。

通常、コンパイラは 1) 字句解析・構文解析を行なう部分、2) 構文解析結果から中間コード（3番地コードや rtl など）を生成する部分、3) 中間コードを使って最適化を行ないアセンブリ言語を生成する部分から構成される。 オブジェクト指向言語、データパラレル言語など、最近の高級言語では他の部分と比べて相対的に 2) の部分がはたす役割が大きくなっている。この部分は言語の本質的な特徴を決定すると同時に、実行効率にも大きく影響するため、この部分の拡張性を特に重視し、内部のインターフェースをきめ細かく規定する。例えば、変数環境を実現するオブジェクトのインターフェースを規定することによって変数スコープに関する機能の変更・拡張を可能にする。 1) と 3) の部分に対しても同様に内部のインターフェースを規定する。

4 機能拡張支援ライブラリ

前節で述べた方法によって幅広い範囲の拡張が原理的に可能になるが、拡張が容易に行なえるようにする機能拡張支援ライブラリの提供も同時に必要である。具体的には、拡張者の目的に応じて、以下の 3 つのレイヤにわかれたライブラリを提供する。

1. Ls に関する知識のみを必要とするレイヤ。従来の言語が持つ言語拡張機能、例えば lisp のマクロ、 C++ の演算子のオーバーロードやテンプレートなどに相当する機能を提供する。

2. コンパイラの構造に関する知識を必要とするレイヤ。コンパイラを構成するオブジェクトの部品を直接操作することによって機能拡張を行なう。リフレクティブな言語と同様の手法を用いることにより、ユーザがコンパイラの部品を新しく定義する際に、1 レベル上の Ls が使用可能になっている。したがって、本来の記述言語 Ld の言語仕様については知る必要がない。また、このレイヤではコンパイラが出力する目的言語 Lo のコードはオブジェクトとして抽象化されているため、Lo に対する知識も必要ない。

3. 目的言語 Lo に対する知識も必要とするレイヤ。 Lo の性質にあわせて、きめ細かい機能拡張が可能。ただし、このレイヤを使った拡張機能は、他のターゲットマシンへの移植性は悪くなる。

5 仕様宣言機構

一般に拡張可能システムには次のような問題がある。独立した複数の機能拡張パッケージを同時に取り込んだ時にそれぞれの機能が正しく動かないことがある、という点である。そのような状況が起きないように、複数の拡張パッケージ同士の衝突を半自動的に検出するための機構、仕様宣言機構を提供する。拡張パッケージのプログラマは、機能が正しく働くためには拡張の対象となるシステムがどのような内部仕様を満たしていなければならないか、また拡張することによって内部仕様がどのように変化するかを宣言する。ある拡張パッケージが必要とする内部仕様が他の拡張パッケージによって変更されたことを仕様宣言機構が見つけた場合、警告を発する。

なお、拡張可能なクラスライブラリを設計する方法に関する研究として [3]、またシステムの一貫性を保った拡張を支援するための型システムの提案として [4] があるが、上記の問題に対してはふれていない。また、動作中のシステムの部品の動的な変更は対象としていない。

6 まとめ

言語仕様およびコンパイル方法を幅広い範囲で変更・拡張可能なコンパイラを設計・実現する方法について述べた。現在は Ls コンパイラとその記述言語 Ld の細かい部分の設計を行なっているところである。今後は、プロトタイプシステムを実現するとともに、様々な言語機能の拡張パッケージを記述してみることによって設計上の問題点等を明らかにしていく予定である。

謝辞

本研究の一部はＲＷＣ計画の一環として「超並列システムアーキテクチャに関する研究」で行なわれたものである。関係各位に感謝いたします。

参考文献

[1] Smith, B. C., “Reflection and Semantics in Lisp”, In *Conference Record of ACM POPL '84*, pp. 23–35, 1984.

[2] Maes, P., “Concepts and Experiments in Computational Reflection”, In *Proc. of OOPSLA '87*, ACM, pp. 147–155, 1987.

[3] Kiczales, G. and Lamping, J., ”Issues in the Design and Specification of Class Libraries,” In *Proc. of OOSPLA '92*, ACM, pp. 435–451, 1992.

[4] Lamping, J., “Typing the Specialization Interface”, In *Proc. of OOPSLA '93*, ACM, pp. 201–214, 1993.