マルチリンガル・マルチターゲット コンパイラの中間言語に関する一考察

2G－7

山良政一*，　長島千成*，　梅川竜一**
*　富士通株式会社
**　㈱富士通静岡エンジニアリング

1. はじめに

言語処理系は，言語ごと・システムごとにコンパイラを提供してきた．ところが，昨今では，「言語間・システム間のＭＭＩの統一」，「言語を意識しない共通の開発環境」，「高性能」，「多種システム間での可搬性の向上」，などが要求されている．この要求を満たすコンパイラとして，①同一システムに対しては，言語間で最適化部とコード生成部を共有し，②同一言語に対しては，システム間で構文解析部と最適化部を共有するようなマルチリンガル・マルチターゲットコンパイラを試作した．このときに，まず問題となるのは，中間言語である．ここでは，実際に試作したものを通して，この中間言語について考察を行う．

2. 言語・システムに依存しない中間言語

従来の中間言語は，言語に依存した情報やシステムに依存した情報を含んでいた．しかし，マルチリンガル・マルチターゲットコンパイラを実現するためには，言語・システムに依存しない中間言語が必要になる．（図１　マルチリンガル・マルチターゲットコンパイラの模型図）　そのため，中間言語は，新規に，高級言語から独立して，設計しなければならない．この観点にたち，FORTRAN,C,Pascalを処理できるマルチリンガルコンパイラをプロトタイプとして試作した．このコンパイラの中間言語を設計する上で以下の二点に注意した．

－　言語として完成された高級言語（この場合は，Ada ）をモデルとする．

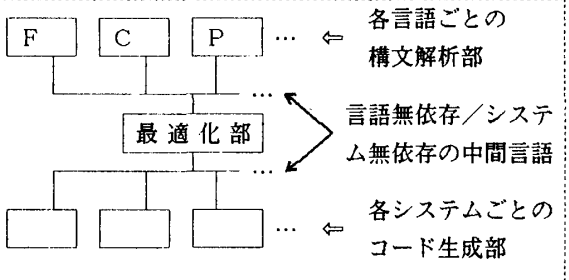

図１　マルチリンガル・マルチターゲットコンパイラの模型図

－　システムは，命令セットの少ない仮想マシンを想定する．

また，中間言語が言語無依存・システム無依存になるように以下のことにも留意した．

－　言語に依存した情報は，すべて各言語ごとの構文解析部が吸収する．

－　システムに依存した情報は，すべて各システムごとのコード生成部が吸収する．

これらのことを考慮して，かつ矛盾のないように検討して，共通中間言語を設計した．（図２　言語依存部とシステム依存部）

3. 中間言語の問題点

こうしてできあがった中間言語を使って，コンパイラを作り動作させた．すると，設計時には気がつかなかった問題点がいくつかあがってきた．

①デバッグ情報の保持形態

従来から，コンパイラはデバッグ機能をオプションとしてサポートしてきた．このデバッグ情報（特に，オブジェクトとソースの対応をとる情報）は，ソース情報がオブジェクトで必要となる．しかし，ソース情報は，構文解析部でしか保持されないためにデバッグ情報の保持形態が困難になった．このため，ソース言語の変数の型などを出力するインタフェースが持てなくなった．

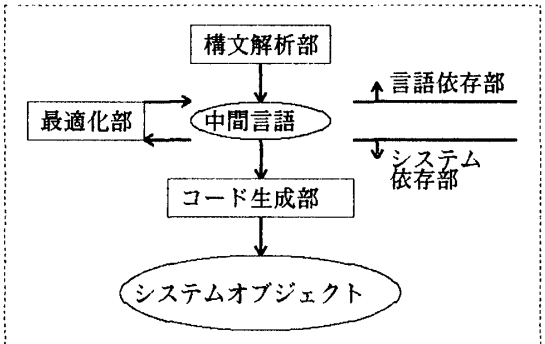

図２　言語依存部とシステム依存部

The Consideration on the Intermediate Notation of the Multilingual, Multitarget Compiler.
Masakazu YAMARYO, Chinari NAGASHIMA, Ryuichi UMEKAWA
FUJITSU,Ltd.
FUJITSU SHIZUOKA Engineering,Ltd.

②言語固有の最適化の困難性

共通コンパイラとして，最適化部を共有させたが，こうすることで，言語個々の最適化が難しくなった．例えばループ構造を取り上げてみる．中間言語の仕様として，ループを一意に定めると，FORTRAN のDOループは，うまく最適化ができたが，C のfor 文に対しては，あまりうまく最適化ができなかった．同じループとはいっても，仕様によっていろいろ処理が考えられるためである．また，Pascalの制約チェックなど最適化に向かない仕様をもつ言語もあるため，一意な最適化自体も困難になった．

③ワーキングセットの拡大

中間言語が，各言語のスーパーセットとなるために，ワーキングセットが，従来の中間言語のサイズのおよそおよそ２．５倍ほどになった．（メモリ上）　我々が設計した中間言語では，テキストの数は減らした（仮想マシンを想定したため）が，その分，辞書の数がかなり増えてしまった．また，辞書の保持する情報も言語ごとの微妙な影響を受けて，辞書自体がかなりの大きさとなってしまったためと考えられる．

④構文解析部の肥大

実際にコンパイラを作ったところ，構文解析部のコンパイラに対する依存度が，従来のものと比較して，およそ３倍弱大きくなった．（図３　コンパイラ内でのフェーズ別依存度（概要））この原因として，(1)中間言語が従来と異なり，かなり独立性の高いものとなった，(2)そのため，構文解析部は，中間言語オブジェクトを出力する働きをしなくてはならなくなった，などのことが考えられる．

4.　中間言語の改善

これらの問題は，すべて『共通』にしたために生じた問題である．従って，これらを解決するためには，一部分『共通』性を犠牲にしなくてはならない．

①デバッグ情報の保持形態の解決法

ソース情報を局所化して，中間言語の辞書部に持つという方法をとった．コンパイラのデバッグ機能をオプションで指定するようにして，この場合のみ参照できるようにした．こうすることで，通常処理から参照できないことから，参照も局所化され，共通中間言語として十分扱えるようになった．

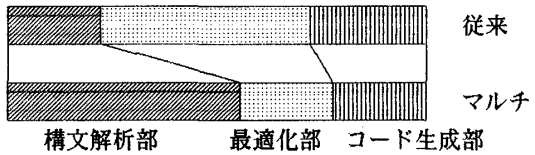

図３　コンパイラ内でのフェーズ別依存度（概要）

②言語固有の最適化の困難性の解決法

デバッグの場合と同様，言語固有の辞書を局所的にもつことで解決した．こうすることで，言語に依存した最適化もできるようにした．

③ワーキングセットの拡大の解決法

これもデバッグ同様，言語固有の情報を局所化して中間言語が保持し，言語間共通情報と固有情報をソースでサブセッティング可能とする方法（その言語で使われない冗長な情報を削除する方法）をとった．こうすることで，最適なワーキングセットが獲得できる．

5.　まとめ

4.で述べたいずれの方法をとっても，言語共通性は多少なりとも失われる．現実に，4.③を実現してしまったために，言語間結合が困難になってしまった．また，3.④は，全体の性能面からみて止むを得ない話であり，解決案は見つからなかった．しかし，今回，プロトタイプコンパイラを作って得た感触はかなり有用なものであった．今後も，解決案を模索しつつ実現にむけて検討していく．

【参考文献】

CHOW F.C. , GANAPATHI M. : 'Intermediate languages in compiler construction'
(ACM SIGPLAN Not. 18,11 (Nov. 1983))

TANNENBAUM A.S. , VANSTAVEREN H , STEVENSON J.W. : 'Using peephole optimization of intermediate code ' (ACM Trans. Program. Lang. Syst. 4,1 (Jan. 1982))

WULF W. , GOOS G.EDS. : 'DIANA Reference Manual' (Computer Science Rept. (Mar. 1981))

GANAPATHI M. , FISCHER C.N. : 'Attributed linear intermediate representations for retargetable code generators'
(Softw. Prac. Exper. 14,4 (Apr. 1984))

BAKER T.P. : 'A single-pass syntax-directed frontend for Ada'
(ACM SIGPLAN Not. 17,6 (Jun. 1982))

情報処理振興事業協会編：最新Ａｄａ基準文法書（共立出版）

細谷僚一監修：ＤＩＡＮＡ入門／言語仕様／応用（啓学出版　1986）