オブジェクト指向論理型言語におけるコンパイル方式

6E−6　　鳥居 悟　　秋山 友子　　小野 越夫
(株)富士通研究所

1 はじめに

近年，プログラミングスタイル，プログラムの部品化，知識表現との親和性，等から，オブジェクト指向パラダイムが注目されている．独自のオブジェクト指向型言語としてはSmalltalk [1] が有名である．また，既存の言語にオブジェクト指向を統合した新しい言語の提案も幾つか試みられている．[2] [3] [4]

一方，Prologを代表とする論理型言語は，その記述力，人間の思考に則した実行，等から高く評価されている．

ここでは，この論理型言語とオブジェクト指向パラダイムとが統合された言語における，コード生成方式について述べる．

このコード生成方式は，以下のような特徴を持つ．

- クラス内部の最適化
  特に最適化フェーズを設けることなく，変数自身に割当て情報を持たせることで，少ないコストで最適な変数のレジスタ割当て命令コード列を生成する．
- クラス間にまたがった最適化
  クラス間でメソッドの記述を共用しつつも，実際に実行すべきメソッドを高速に探索する，論理型言語のインデキシングを利用した，メソッド選択処理の命令コードを生成する．

なお，以下ではWarrenの中間コード [5] について一般的な知識があるものと仮定している．

2 コンパイラ

このコンパイラの基本的な方針は，各クラスのコードのモジュラリティを考慮し，高速に実行するコードを，低いコストで，生成することである．

継承した親クラスのコードを子クラス内に取り込んでしまうと，全体のコード量が膨大になってしまう．さらに，各クラスのソースで示されたものに対応する命令コードはクラス単位で管理すると都合が良い．

そこでコンパイラは，これらの条件の基で最適な命令コードを生成するよう，以下のような特徴を持つ．

2.1 クラス内部の最適化

クラス内部のコードを生成する時は，処理のパスを減らした，高速な生成を行なわせることとした．

これは，ソフトウェアの開発に於けるコンパイル&デバッグのターンアラウンドの時間を短くしたいからである．

そこで，少ないコストで，最適なレジスタアロケーションの命令コードを生成する．

2.2 クラス間にまたがった最適化

オブジェクト指向プログラムの実行には，クラス間のメソッド呼出しの処理に大きなウェイトがかかる．

そこで，実際に実行すべきメソッド定義を高速に探索し，実行するような命令コードを生成する．このメソッド選択処理の命令コードは，高速に，メソッドのインデキシングを行なうものである．

3 レジスタアロケーション

まず前提として，組み込み述語が使用する引数レジスタは，ユーザ定義述語と違い，呼び出し時に自由に指定出来るものとする．また，節のボディ部において，実行制御に係わるような特殊な組み込み述語か，ユーザ定義述語までの述語呼出し列を述語呼出し群と分類し，複数の述語呼出し群に出現する変数を永久変数とする．

節のサーチは，最適化コードを生成するコストを減らすため，変数情報の収集とコード生成の2回行う．

3.1 変数情報の収集

ここでは，各変数に対して以下のような情報を収集することにした．

1. どこで出現するか
2. どこに格納してはまずいか（構造体の要素）

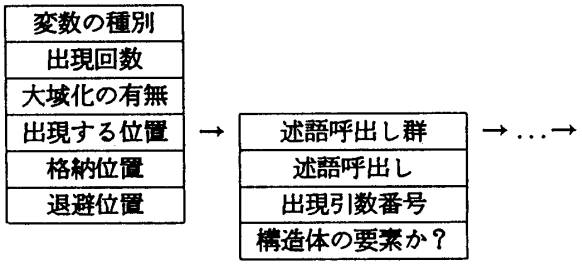

図 1: 変数が持つ情報

Code Generation on Logic Based Object Oriented Language
Satoru Torii, Yuko Akiyama, Etsuo Ono
FUJITSU Laboratories, Ltd.

具体的には，節内のどの述語に出現するか，述語呼出しに対応する引数レジスタの番号，また，その出現するところが構造体の要素であるかどうか，についての情報を収集する．

ここで収集された情報は，出現する位置の情報として，各変数に持たせることにする．変数が持つ情報の構成を図 1に示す．

3.2 コード生成

各変数のアロケーションのための命令コード列を生成する時に，その変数自身がもつ情報を利用する．

情報利用の仕方は，以下の対応表 1 の通り．この対応表を基に，変数の割り当て先レジスタを決定する．

出現場所	構造体内	割当て先レジスタ
引数 A	NO	レジスタ A
引数 A	YES	レジスタ A 以外（必ず）
引数 B	NO	レジスタ B（希望）
引数 B	YES	レジスタ B 以外（希望）
指定なし	指定なし	命令コード不要

表 1: 変数の割当て

4 メソッドインデキシング

ローカル述語に対しては，その全ての節のコード生成が終了した後，インデキシング命令を生成する．しかし，メソッド述語に対しては，このインデキシング命令を継承解決時に生成する．

以下に，このメソッド述語のインデキシング命令の生成方式について述べる．

4.1 クラスの持つ内部情報

各クラスのコード領域には，内部からのみ呼び出される述語（ローカル述語）と，外部から呼び出される述語（メソッド述語）の 2 種類の述語の命令コードが格納されている．

メソッド述語に対しては，各節に対しするコードを生成する時にその各節のインデキシングのための情報を保持しておく．そしてこの情報を，メソッドの節選択情報としてクラス内部に格納する．この節選択情報には，節のインデキシングのキーとなる値と，その節の命令コード列へのオフセット値とを持つ．

4.2 コード生成

クラスの継承解決時，このクラスの，全ての親クラスが持つメソッドの節選択情報を収集する．収集された節選択情報から，同一形式のメソッドの情報を統合し，インデキシング命令列を生成する．

このクラスのメソッドへの呼出し命令は，ここで作成されたインデキシング命令列の先頭を呼び出すことになる．このインデキシング命令列には，このクラスの内部に記述されている命令コード列，あるいは，このクラスの親クラスに記述されている命令コード列への呼出し命令が記述されている．

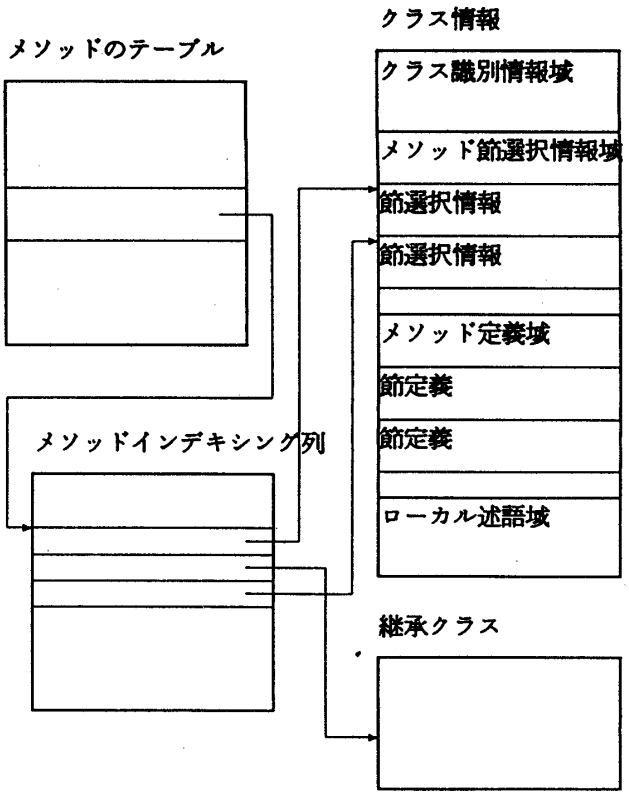

図 2: クラスの内部構造

5 今後の課題

ここでは，オブジェクト指向型と論理型とが統合された言語に対して，各クラスの命令コード列を独自に管理しているような場合に於ける，最適な命令コード列の生成方式について述べた．

今後は，この方式を使って生成された命令コード列に対して，実際のアプリケーションプログラムを対象とした効果を計測，評価していく予定である．

また，この方式を他のオブジェクト指向型言語に対して適用出来る可能性を探りたい．

参考文献

[1] Goldberg A. and Robson D. : “Smalltalk–80, The Language and Its Implementation”, Addison–Wesley, 1983

[2] 井田, 他 : “Common Lisp オブジェクトシステム ”, 共立出版, 1988

[3] Stroustrup B. : “The C++ Programming Language” Addison–Wesley, 1986

[4] Chikayama, T. : “Unique Features of ESP”, Proceedings of FGCS ’84, ICOT, 1984

[5] Warren D.H.D. : “An Abstruct Prolog Instruction Set”, SRI International, Technical Note 309, 1983