意味モデルに基づくコード生成方式

5E-10

十山　圭介　　渡邊　坦
（（株）　日立製作所システム開発研究所　）

1.　はじめに

ソース言語や目的計算機のアーキテクチャが多様化するにつれ、保守や変更を容易にし、更に新しい計算機に対して効率良くコード生成を行うことを目標としてコード生成部を定式化することが重要な課題となっている。このように、複数の計算機機種に対するコード生成部を系統的な手法で開発することには強い要望があるが、目的コードに対しては高性能性が要求されるので、マシン依存性の除去と性能との兼ね合いが問題である。

マシンに非依存な形式で高性能な目的コードを生成するためには、レジスタの割付やデータ項目へのアクセスパスの決定において機種に依存する情報をそうでないものから明確に分離しなければならず、これがリターゲッタビリティに対する課題である。　我々は言語の意味を表現するために、実行方式を基に記憶域やコードを抽象化したマシンモデル（意味モデル）を作成し、共通中間語から意味モデル、意味モデルから実マシンへの写像という段階を経ることによって目的コードを生成する方式を考案した。

2.　意味モデル

意味モデルは、対象とする言語群に共通する機能を抽象化したものを、それを実現するために必要となるマシンの資源を抽象化して表現したものである。　コード生成は本質的にマシンに依存するものであるが、目的コード設計の基礎となる実行方式は、対象とする言語群を例えばPascal風のものに限定すると、その大部分を以下の概念によって共通に扱うことができる：

a）環境の操作とデータ項目の参照，　　b）式の評価，　　c）制御の移行

これらについて、記憶域のモデル，コードのモデルを作成し、その上での評価関数を設定することによりコード生成を定式化して記述できる。　本発表ではデータ項目へのアクセスについて述べる。

2．1　記憶域の構成

データ項目は変数と定数とに分類され、変数は静的変数と動的変数とに分類されるが、環境の操作に関連して動的変数へのアクセスが重要である。変数の存在する領域を次の６つに分類して実行時の記憶域を表現する：

a）LocalFrame，　b）OuterFrame，　c）StackTopArea，
d）CallerArea，　e）CalleeArea，　f）Heap

各領域は基底によって識別される。　動的変数は、ある時点でａ）～ｆ）のいずれかの領域にあり、その開始番地(ロケーション)は各領域を識別する基底と、基底からの変位とによって特徴付けられる。すなわち、

LocalFrameにある変数 →（B，d），　OuterFrameにある変数 →（G，d）…

として各変数の位置を表現する。

これらはブロック型の言語に対する記憶域の割付を実現するものであり、入れ子になったブロックの変数をスタックによって管理することに対応している。

2．2　中間語への反映

中間語で表現された変数への参照に対して、そのノードにつくロケーション属性として上記の各領域を示す。いずれのロケーションであるかによって、基底が決定され、そこからの相対変位をシンボル表などから得ることによって(基底，相対変位)として開始アドレスを表現する。

2．3　マシンとの対応

レジスタ，特定のメモリもしくは定数に各領域の基底を保持する。基底がレジスタに保持されている場合、レジスタ相対モードによって領域内の変数にアクセスする。　ここでは、Ｂをフレームポインタとして特定のアドレスレジスタに保持し、ＧはLocalFrame内のディスプレイ領域に保持する形式として実マシンの資源との対応付けを行ない、中間語のノードに付与したロケーション属性に従って評価手続を用いてアクセス用コードを生成する。

3.　例

配列のアドレス参照について本方式によるコード生成の例を示す。

Ａ[ｘ]に対するアクセス･コードは次のようにして生成される。　Ａ[ｘ]は共通中間語において図1のように属性付きの木構造で表現され、レジスタやロケーションの属性が付与されている。　木構造の順序に従ってノードを辿り、図２に示す評価関数がＥＶＡＬ→ＬＯＣと呼びだされる経路で、ｘに対してOuterFrame，　Ａ[ｘ]に対してインデクス付LocalFrameのアドレシングコードが図３のように出力される。

A Code Generation Strategy based on Semantic directed Run-time Model
Keisuke TOYAMA, Tan WATANABE
Systems Development Laboratory, HITACHI, Ltd.

4. コード生成におけるリターゲッタビリティ

記憶域とレジスタを割付けた後の中間語に対して以下の手順でコード生成を行なう：

1）意味モデル（記憶域、コードと制御の移行）を構築する。

2）中間語の構文規則に対してモデルの各要素を属性として使用して、意味モデルの資源に対する意味記述をそれらの属性に対する方程式として記述する。

3）意味モデルから実マシンへの写像をとる

意味モデルでの各資源を実マシンのレジスタやメモリを用いて表現する。 この対応に従って2）の意味記述から得られるオブジェクト・テンプレートを展開することによって、目的とするコード系列を生成する。

機種依存部は大部分が3）に閉じ込められることになり、コード生成部において機種依存部と非依存部との分離ができることになる。 これによって、データの移動やレジスタへのロード、演算時のデータの所在等マシンに依存する部分について、評価関数を適宜決定することによりコード生成でのリターゲッティングが比較的容易になる。

5. おわりに

言語の機能に対応させてマシンの資源を抽象化した意味モデルを設定し、共通中間語に対してそのモデル上でコード・スケルトンを作成し、モデルの資源と実マシンの資源とを対応させることによって実マシンに対する目的コードを生成する方式について述べた。 この方式によれば、マシン依存部の大部分をモデルから実マシンへの資源の写像部に閉じ込めることができ、コード生成におけるリターゲッタビリティを向上させることができる。

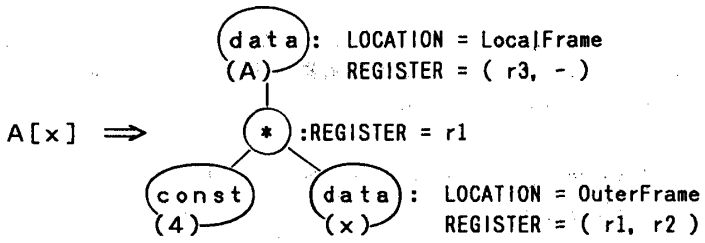

図1．中間語と属性

```
LOAD r2, disp.level(FP)
LOAD r1, offset1(r2)
SLA  r1, 2
LOAD r3, offset2(r1, FP)
```

図3．コード例

```
function LOC( t: tree,
              c: CONTEXT ): ADDRESS;
begin
  case t@.Location of
StackTopArea : ...;
LocalFrame : begin
    BASE  := FP;
    OFFSET:= シンボル テーブルより;
    INDEX :=
      if FIRSTSON( t ) = nil then
            null_index
      else  EVAL( t, c ) end;
OuterFrame : begin
    BASE  := GENLOAD( GET_LINK( level ) );
    OFFSET:= シンボル テーブルより;
    INDEX :=
      if FIRSTSON( t ) = nil then
            null_index
      else  EVAL( t, c ) end;
CallerArea : ...;
CalleeArea : ...
  end;
  LOC:= BASE || OFFSET || INDEX
end;
```

```
function EVAL( t: tree,
               c: CONTEXT ): ADDRESS+REGISTER;
var
  result1, result2: ADDRESS+REGISTER;
  op: OPERATION;
begin
  case KIND( t ) of
       :
data : begin
    if t@.Register = no_register then
      EVAL:= LOC( t, c )
    else
      EVAL:= t@.Register end;
add  : begin
    result1:= EVAL( FIRSTSON ( t ), c );
    result1:= EVAL( FIRSTSON ( t ), c );
      case t@.Optype of
    integer : op:= addint;
    real    : op:= addreal
      end;
    GENBINOP( op, result1, result2, c );
    EVAL:= result1 end;
sub  : ...;
       :
  end
end;
```

図2．評価関数

参考文献

1) Bjørner, D. & Jones, C. B. : Formal Specification & Software Development, Prentice-Hall International, 1982

2) 渡辺 他 : マルチターゲットADAコンパイラ方式, 情報処理全国大会(27)