1M−4

代数的仕様の論理型プログラムへの変換法

濱口　毅　酒井　正彦　阿草　清滋
名古屋大学工学部

1　はじめに

代数的仕様記述の既存の直接実現系としてDimple, Cdimple[1]があり，仕様をそれぞれLisp，Cのプログラムに変換する．しかしこれらは仕様の階層化がなされておらず，読解性を損ねていた．本論文では仕様をクラスと呼ばれるモジュールに分割し，クラス間に継承，パラメタライズなどの階層性を持たせて記述することを目的として，設計した仕様記述言語について述べ，論理型プログラミング言語に変換する方法を考察する．

2　階層構造を持つ代数的仕様記述

仕様はクラスと呼ばれるモジュールを単位として記述する．他のクラスとして記述された仕様を継承する新たなクラスを仕様記述したり，型変数を用いて記述された仕様に対して型変数を具体的なデータ型に置換えて仕様を記述するパラメタライズの機能を持つ．

クラスはおおよそ次のような形式である．(太字のイタリックは予約語であり，中括弧でくくられたものは省略可能である．)

{generic}class クラスの識別名
　{inherit 継承するクラス名*}*
　{use 使用するgenericクラス名
　　replace 型変数
　　with 置換えるデータ型*}*
　sort: 新たに定義されるデータ型の宣言
　{parameter 型変数の宣言*}*
　function
　　関数記号の宣言
　end;
　variable
　　変数記号の宣言
　end;
　等式の記述
end.

図1　仕様の形式の概略

仕様の最初にはクラスの識別名の宣言する*class*，あるいは型変数をともなうgenericクラスを宣言するための*generic class*が記述される．またクラスを継承する場合には*inherit*を用いる．パラメタライズを行う場合は*use*を用いて使用するgenericクラスを宣言した後，*replace*…*with*… を用いて型変数の置換えを記述する．クラスで新たに定義するデータ型の名前を*sort*で宣言する．genericクラスの定義の場合には*parameter*で型変数を宣言する．その後に関数記号の宣言，変数記号の宣言と続き，最後に仕様の中心となる等式を記述する．

3　代数的仕様記述の論理型言語への変換

本仕様記述言語では，継承やパラメタライズの機能を持つため，LispやCなどの通常の言語への変換は容易ではない．Prologにオブジェクト指向パラダイムを取り入れた言語としてCESP(Common Extended Self-contained Prolog)があり，階層化された仕様を実現するのに適していると考えられる．CESPは手続き型言語ではなく論理型言語であるため等式によって記述された仕様を論理式に変換しなければならない．ここではその変換方式について述べる．簡明を期するためCESPの代わりにPrologへの変換について述べる．

仕様記述においてn引数の関数記号funcの宣言は次のような形式になる．

$$func : sort_1, sort_2, \ldots sort_n \rightarrow sort_{func};$$

これは引数の型が$sort_1$,…,$sort_n$であり返値の型が$sort_{func}$である関数$func$を宣言している．このような関数に対応して，ある値がその関数の返す値であるかどうかを判定する$n+1$引数述語を作ることができる．例えば関数$func$に対応する述語

$$Func(x_1, x_2, \ldots, x_n, y)$$

は，次の論理式を満たすように作られる．

$$Func(x_1, x_2, \ldots, x_n, func(x_1, x_2, \ldots, x_n))$$

How to Transform Algebraic Specification into Logic Program
Takeshi HAMAGUCHI, Masahiko SAKAI, Kiyoshi AGUSA
Nagoya University

また，等式の中で関数記号 $func$ が左辺のトップレベルに現れるものを $func$ に関する変換規則としてとらえる．各々の変換規則を，等式の両辺を対応する述語に変換し，右辺から左辺への含意とみなす．一般に仕様に現れる等式は次のような形をしている．

$$func(term_1, \ldots, term_n) == term_{n+1}$$

左辺は対応する述語に変換し，右辺については現れる関数をすべて対応する述語に置換え，論理積で結ぶ．このようにして得られた論理式より，Prolog のプログラムが得られる．

次の整数の加算の仕様は以下のように変換される．

```
class addtion_of_integer
  sort: integer;
  function
    zero: -> integer;
    s: integer -> integer;
    add: integer, integer -> integer;
  end;
  variable
    x, y: integer;
  end;
  add(s(x), y) == s(add(x,y));
  add(zero(), y) == y;
end.
```

図2　整数の加算の仕様

ここでは関数の引数および返値の型は整数型のみなので型については考慮しない．関数記号の宣言部よりこの仕様では $zero(), s(x), add(x,y)$ の3個の関数が現れることがわかる．またこれらの関数に対応した述語を作ることができ，まず，次の常に真となる論理式が得られる．

$$\mathrm{Zero(zero())}$$
$$\mathrm{S}(x, \mathrm{s}(x))$$
$$\mathrm{Add}(x, y, \mathrm{add}(x, y))$$

この例では等式の左辺のトップレベルに現れる関数記号は add のみである．従って，等式から次のような論理式が得られる．

$$\begin{aligned} \mathrm{Add(zero}, y, r) &\leftarrow r = y \\ \mathrm{Add(s}(x), y, r) &\leftarrow \mathrm{Add}(x, y, t) \wedge \mathrm{s}(t, r) \end{aligned}$$

これらの論理式を Prolog のプログラムに直すと次のようになる．

$$zero(zero). \quad (1)$$
$$s(X, s(X)). \quad (2)$$
$$add(zero, Y, R) : -R = Y. \quad (3)$$
$$add(s(X), Y, R) : -add(X, Y, T), s(T, R). \quad (4)$$
$$add(X, Y, R) : -R = add(X, Y). \quad (5)$$

加算 $add(s(s(s(zero))), s(s(zero)))$ の計算は例えば次のようなゴール節を与えることにより行う．

$$? - add(s(s(s(zero))), s(s(zero)), Z).$$

このゴール節に対する導出は次のように行われる．下向き矢印 (↓) の右に書かれている数字は適用したプログラムの節の番号，さらにその右は行われた代入である．

$? - add(s(s(s(zero))), s(s(zero)), Z).$
$\downarrow (4)\{X = s(s(zero)), Y = s(s(zero)), R = Z\}$
$? - add(s(s(zero)), s(s(zero)), T_1), s(T_1, Z).$
$\downarrow (4)\{X = s(zero), Y = s(s(zero)), R = T_1\}$
$? - add(s(zero), s(s(zero)), T_2), s(T_2, Z), s(T_1, Z).$
$\downarrow (4)\{X = zero, Y = s(s(zero)), R = T_2\}$
$? - add(zero, s(s(zero)), T_3), s(T_3, T_2), s(T_2, T_1), s(T_1, Z).$
$\downarrow (3)\{Y = s(s(zero)), R = T_3, T_3 = s(s(zero))\}$
$? - s(s(s(zero)), T_2), s(T_2, T_1), s(T_1, Z).$
$\downarrow (2)\{X = s(s(zero)), T_2 = s(s(s(zero)))\}$
$? - s(s(s(s(zero))), T_1), s(T_1, Z).$
$\downarrow (2)\{X = s(s(s(zero))), T_1 = s(s(s(s(zero))))\}$
$? - s(s(s(s(zero)))), Z).$
$\downarrow (2)\{X = s(s(s(zero))), Z = s(s(s(s(s(zero)))))\}$
□

最後に空節が導かれて，Z には $s(s(s(s(s(zero)))))$ が代入されている．これは2と3の和を計算しており，要求された仕様に合致している．

4　おわりに

等式で記述された仕様を Prolog で扱うことのできる Horn 節に変換する方法について述べた．現在，階層化された仕様の実現法については検討中である．

謝辞

日頃御指導下さる名古屋大学稲垣康善教授，坂部俊樹助教授，また御討論下さった本研究室の皆様に感謝致します．

参考文献

[1] 酒井正彦, 坂部俊樹, 稲垣康善: 抽象データ型の代数的仕様の直接実現系 Cdimple, コンピュータソフトウェア Vol.4,No.4(1987),pp.16-27.

[2] 山本隆広: データ型の代数的記述機能を持つ手続き型プログラミング言語に関する研究, 名古屋大学大学院工学研究科情報工学専攻修士論文 (1989).