3L-2

代数的仕様の階層的記述が可能なＵｌｔｒａＣ

山本隆広　　坂部俊樹　　稲垣康善
（名古屋大学　工学部）

１．はじめに

抽象データ型の概念およびその代数的仕様記述法は，高い抽象レベルでの階層的なソフトウェアの開発を可能にするばかりでなく，仕様の直接実現によるプロトタイピング，プログラムの正当性検証などを可能にする．しかし，データの入出力などのように，抽象データ型で記述するより既存のプログラミング言語を使って記述する方が容易な場合がある．この観点から，筆者等のグループでは，Ｃ言語の文法に，データ型を代数的に定義する機能を加えて拡張したUltraCを設計した[3]．

UltraCの目的の一つはプロトタイピングのツールとして用いることであるが，そのためにはコンパイル時間を短くするために分割コンパイル可能なことが望ましい．そこで，抽象データ型定義部分の分割コンパイル可能なUltraCの処理系を実現をしたので報告する．

２．Cdimple

UltraCの本処理系は，旧UltraCと同様に，本研究室ですでに開発された代数的仕様記述直接実現系Cdimple を基礎に作成されている．Cdimpleは等式によって記述された仕様が入力されると，その仕様を項書き換えシステムと見なして計算をするＣ言語のプログラムを出力する．生成されたＣ言語のプログラムをコンパイルし，そのオブジェクトを実行して項を入力すると，その項の正規形が出力される．

３．package とinstance

データ型の代数的定義をするために，旧UltraCではpackageブロックとinstanceブロックの構文が，Ｃ言語の構文に付け加えられている．packageの中では代数的仕様によってパラメータ付き抽象データ型を定義する．packageはデータ型を仮引数としてとり，仮引数に実引数として他のデータ型を与えることによって新たなデータ型を定義する．仮引数はpackageのブロックの中でparameter文によって定義する．仮引数データ型への実引数データ型の割当はinstanceブロックで記述する．instanceブロックで宣言された新しいデータ型は通常のＣ言語のデータ型と同様に扱うことができる．packageと instanceで定義したデータ型の変数を宣言するときは<instance名>.<ソート名><変数名>とし，演算を使うときは<instance名>.<関数名>（<引数>，<引数>，…）と書く．

４．データ型の階層化

UltraCでは，抽象データ型の定義を階層的にするために，packageとinstanceの構文を用いる．packageは直感的には代数的な仕様であるが，packageの仕様の中で別のpackageの仕様を引用することが許される．この場合，引用する仕様と引用される仕様とが矛盾したり，引用される仕様の意味が，引用する仕様の影響を受けて変わってはならない．そこで，あるpackageの中での，別のpackageの引用に関して次の制限を設ける．

①呼ばれたpackageの中の関数は，呼び出し側のpackageの仕様の等式の左辺に現れてはならない．
②package間の関数の引用関係が再帰的であってはならない．

さらに，旧UltraCの仕様に対する次の２つの条件も満たさなければならない．

③packageの中では，どの２つの等式についても，一方の左辺と他方の左辺の部分項とは（同じ等式の左辺同士は除いて）単一化可能でない．（仕様の等式を項書き換えシステムと見なしたとき，無曖昧であること）
④各々の等式では，同じ変数が左辺に２回以上現れない．（仕様の等式を項書き換えシステムと見なしたとき，線形であること）

５．関数の自動検索機能

packageで抽象データ型の定義の階層が深くなると，どの関数がどのpackageに含まれているのかを検索することが煩雑になる．そこで，関数がどのpackageで定義されたのかを自動的に検索する機能を付け加えた．あるpackage Ａの中で，他のpackage Ｂの関数を使って仕様記述するときはpackage Ａの中でextern<関数名>（<引数>,<引数>,…）,…;と宣言する．そして，instanceブロックで use<instance Ｂ'>と宣言すると，package Ａの中で宣言した関数は instance Ｂ' を定義したpackage Ｂの中にあるものとして，その関数がpackage Ｂの中で実際に定義されているかどうかを調べる．あればその関数の仕様を加え，なければその関数はＣ言語で定義されていると見なして，Ｃ言語のライブラリにある関数をリンクする．

６．UltraCの処理系の実現

Cdimpleは，packageとinstanceとで定義された代数的仕様をＣ言語の関数に変換する．各関数プログラムは，仕様中に宣言された関数記号ｆに対応し，引数として項$t_1,\cdots,t_n$を受け取ると，項ｆ（$t_1,\cdots,t_n$）を仕様の等式によって書き換えて得られる項を返す．Cdimpleの生成するプログラムにおける項を表現するリストのセルは図１のような構造体となっている．

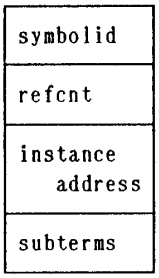

図１　項を表すリストのセル

UltraC with Hierarchical Algebraic Specification
Takahiro YAMAMOTO, Toshiki SAKABE, Yasuyoshi INAGAKI
School of Engineering, Nagoya University

symbolidは関数記号を表す番号であり，instanceをコンパイルするときに，それぞれのinstanceごとに，異なった値になるように正の整数が割り当てられる．項書き換えを行うときはこの番号をもとにパターンマッチングが行なわれる．refcntはガベッジコレクションのための領域である．instance addressはsymbolidによって表される関数記号を含むpackageをインスタンス化しているinstanceを表す番号である．subtermsは，部分項を表すセルへのポインタの配列である．

このように，Cdimpleによって生成された関数プログラムでは関数記号を正の整数で管理するため，項のマッチングを効率よく行えるが，その反面，分割コンパイルをすると関数番号の衝突がある場合に，誤った動作をすることになる．この点を改良するため，本UltraCでは，Cdimpleを次のように改良している．

等式の右辺に別のpackageの関数記号があらわれたときは，その関数記号以下の項を表すセルへのポインタと，そのセルの間に特別なセル（symbolidが -9000であるようなセル）が挿入されるようにする．こうすることによって，Cdimpleの変更を最小にし，しかも，書き換えの効率の悪化を招くことなく，分割コンパイルが可能になる．

packageとinstance以外の部分で，代数的に定義された関数を呼び出す部分はCdimpleの生成したプログラムとのインターフェイスをとる関数をとおして呼び出す．

7．拡張Cによる記述

図２にUltraCで書かれたプログラムの例を示す．この例では，配列を代数的に定義し，その配列を使って"package test_ar"でスタックを定義している．"base_ar"というpackageで，"input_data"（配列に値をいれる関数），"array_data"（２番めの引数を添え字として配列から値を取り出す関数）を定義している．"parameter data"という文は"data"というソートが"base_ar"の仮引数であることを宣言している．そして，"instance test_ar tb_ar"は，"instance test_ar tb_ar"のブロックの"use int_ar"の宣言によって，"package base_ar"で定義した配列の仕様に新たな仕様を付け加えてスタックの仕様を定義していることを宣言している．また，"typedef int data"の宣言で"pacakge test_ar"の仮引数"data"にＣ言語のint型を実引数として与えてint型のスタック"tb_ar"を作っていることを宣言している．package中で定義されinstanceブロックで定義域と値域を与えられた関数は通常のＣ言語の関数と同様に扱うことができる．

8．まとめ

階層的な代数的仕様記述による抽象データ型の定義機能を持つＣ言語である UltraC の文法を改善し，分割コンパイルができる処理系を実現（ガベッジコレクションは除く）した．実際的なプログラムの記述を通して，UltraCの評価をすることは今後の課題である．

```
#include <stdio.h>
package base_ar {
   parameter data;
   extern bool EQ_INT(int, int);
   array input_data(array, int, data)
     , newarray();
   data array_data(array, int); {
     array_data(input_data(a, i, d), j)
       == IF(EQ_INT(i, j), d, array_data(a, j));
   } };
instance base_ar int_ar {
   typedef int data;
};
package test_ar {
   parameter data;
   extern array
     input_data(array, int, data)
       , newarray();
   extern data array_data(array, int);
   extern int SUM_INT(int, int)
     , DIFF_INT(int, int);
   stack push(stack, data), pop(stack)
     , new(), stackdata(array, int);
   data top(stack);
   {
      push(stackdata(a, j), x) ==
        stackdata(input_data(a
                             , SUM_INT(j, 1)
                             , x)
                  , SUM_INT(j, 1));
      pop(stackdata(a, j)) ==
        stackdata(a, DIFF_INT(j, 1));
      top(stackdata(a, j)) ==
        array_data(a, j);
      new() == stackdata(newarray(), 0);
  } };
instance test_ar tb_ar {
   typedef int data;
   use int_ar;
};
main()
{
   tb_ar.stack b;
   tb_ar.data x, a;
   b = tb_ar.new();
   for(x = 10; x < 14; x++) {
      b = tb_ar.push(b, x);
   }
   for(x = 10; x < 14; x++) {
      a = tb_ar.top(b);
      printf("test data is %d¥n", a);
      b = tb_ar.pop(b);
   } }
```

図２ 配列で実現したスタックの拡張Ｃによる記述

謝辞

日頃御指導賜る豊技大本多波雄学長，中京大福村晃夫教授，並びに御討論下さる平田富夫講師，酒井正彦氏をはじめとする研究室の諸氏に感謝する．

文献

[1]酒井，他：抽象データ型の代数的仕様の直接実現系 Cdimple，ｺﾝﾋﾟｭｰﾀｿﾌﾄｳｪｱ 1987 Oct.

[2]W. G. Olthoff: Augmentation of Object-Oriented Programming by Concepts of Abstract Data Type Theory: The ModPascal Experience, OOPSLA 1986 Proceedings.

[3]山本，坂部，稲垣：UltraCにおける抽象データ型の階層的定義機能，東海連大 1988.