表現形式の変換による仕様のデバッグ

2R－8

山本　剛[*1]　所　洋一[*2]　小野　康一[*1]　深澤　良彰[*1]　門倉　敏夫[*1]
[*1]早稲田大学　理工学部　[*2]ＮＨＫ　富山放送局

１．はじめに

ソフトウェア・システムが大規模になるにつれて，ソフトウェアの信頼性，生産性の向上のために，ソフトウェアの仕様記述が議論されるようになった．

前提終了条件表記の仕様記述法[1]では，抽象データ型（ＡＤＴ）に対する操作を行なう前に成立すべき条件（状態）と操作後に成立すべき条件（状態）を記述する．この仕様記述法は操作の前後における状態のみの記述であるため，比較的記述性がよい．

代数的仕様記述法[2]では，抽象データ型を代数とみなし，それを公理と呼ばれる等式の集合で仕様記述することが基本となっている．公理は直感的ではあるが，複雑になると完全な記述を行なうのはかなりの熟練を要する．

本研究では，与えられた前提終了条件表記の仕様から代数的仕様への変換を行なう．これにより，ユーザはエラーの波及した代数的仕様を参照して，前提終了条件表記の仕様をデバッグする．

２．仕様変換手法

前提終了条件表記の仕様を代数的仕様に変換する場合，各操作モジュールのパラメータに，そのパラメータの型と同じ型を出力する操作モジュールを組み合わせ，同じ状態のものを選べば，公理を生成することが可能である．しかし，この組み合わせは，操作数の増加に伴って指数関数的に増加する．

そこで本研究では，以下で述べるような，組み合わせ爆発の起こらない発見的な変換手法を考案した．しかし，この手法では，前提終了条件表記の仕様で記述された内容全体を代数的仕様記述に反映していないため，エラーが見つからないこともある．

【Ⅰ】ある抽象データ型の操作の入力パラメータとして，それ以外の操作の結果を採用し，出力時における状態（モジュール内部変数等）を計算する．ここで，入力パラメータに採用する操作にパラメータがある場合，更に他の操作結果は採用しない．また，入力パラメータが複数個ある場合，入力パラメータに採用する操作は最大で１個とする．

【Ⅱ】Ⅰの結果のうち同じ状態のもの（同じ値をとるもの）を組み合わせ，等式を作成する．ここで，操作モジュールに条件分岐が含まれる場合は，各条件毎に組み合わせを行なう．

【Ⅲ】Ⅱの各等式について，次のような選択規準でふるいにかける．

① 等式の右辺に現われる変数は，必ず左辺にもなければならない．

② 等式の右辺の操作モジュール内の条件分岐が，左辺の操作に依存してはいけない．

③ 等式の左辺の操作数≧右辺の操作数，でなければならない．

④ 等式の左辺の操作が同じで右辺が違うような組み合わせが複数個存在する場合には，

（等式の左辺の操作数－右辺の操作数）

が大きい方，等しい場合には，

（等式の左辺の変数の数－右辺の変数の数）

が大きい方，即ち，よりリダクションの進む方を選択する．

⑤ 操作モジュールが条件分岐している場合は，全ての条件が揃っていなければならない．

３．仕様変換例

操作モジュールtop の終了条件部にエラーを含んだ，スタックについての図１のような前提終了条件表記の仕様を本変換手法で変換する．この仕様から，変換Ⅰにより図２のような，操作を組み合わせた結果の状態が得られる．更に，変換Ⅱにより図３が得られ，これら各々の等式に対して，変換Ⅲの①～⑤の選択規準を順に適用すると，図４のような等式が残る．

例えば，図３の第１式は，図２の【1a】と【3b】が組み合わされた式で，これには選択規準③が適用され，ふるいおとされる．また，図３の第３式には適用する選択規準がないので最終的に公理として残る．この式を見ると，スタックs1に要素e1を積んでから，スタックのトップの要素を取りだすとNULLになっているのがわかるが，実際は，e1が取りだされなければならないので，topかpushのどちらかの操作が意図するものと違っているのがわかる．

Specification Debugging by Transforming Representation

Tsuyoshi YAMAMOTO[*1], Youichi TOKORO[*2], Kouichi ONO[*1], Yoshiaki FUKAZAWA[*1] and Toshio KADOKURA[*1]
[*1] School of Science and Engineering, Waseda University.　[*2] NHK Toyama Broadcasting Station.

```
MODULE creates() => stack
    INTERFACE
        s       : PARAMETER(WRITE)stack.
        stack   : IMPORT(TYPE).
    ENDINTERFACE
    SPECIFICATION
        PRE-creates  : TRUE
        POST-creates : s.p'=0
    ENDSPECIFICATION
ENDMODULE creates

MODULE push(stack,ElemType) => stack
    INTERFACE
        s : PARAMETER(READ/WRITE)stack.
        e : PARAMETER(READ)ElemType.
        stack : IMPORT(TYPE).
    ENDINTERFACE
    SPECIFICATION
        PRE-push  : TRUE
        POST-push : s.s'[s.p]=e AND s.p'=s.p+1
    ENDSPECIFICATION
ENDMODULE push

MODULE pop(stack) => stack
    INTERFACE
        s : PARAMETER(READ/WRITE)stack.
        stack : IMPORT(TYPE).
    ENDINTERFACE
    SPECIFICATION
        PRE-pop  : emptys(s)==FALSE
        POST-pop : s.p'=s.p-1
    ENDSPECIFICATION
ENDMODULE pop

MODULE top(stack) => ElemType
    INTERFACE
        s : PARAMETER(READ)stack.
        e : PARAMETER(WRITE)ElemType.
        stack : IMPORT(TYPE).
    ENDINTERFACE
    SPECIFICATION
        PRE-top  : emptys(s)==FALSE
        POST-top : e'=s.s[s.p]
    ENDSPECIFICATION
ENDMODULE top
```

図１．エラーを含んだモジュール仕様（一部）

```
【1a】creates()        :={ s.s=[],   s.p=0 }
【3b】pop(creates())   :={ s.s=[],   s.p=0 }
【3c】pop(push(s1,e1)) :={ s.s=[S1₀,…,S1s1.p-1],
                           s.p=s1.p}
【4c】top(push(s1,e1)) := NULL
【6a】s1 := { s.s=[S1₀,…,S1s1.p-1], s.p=s1.p }
【6f】NULL
```

図２．変換Ｉの結果（一部）

```
【1a】creates() = 【3b】pop(creates())
【3c】pop(push(s1,e1)) = 【6a】s1
【4c】top(push(s1,e1)) = 【6f】NULL
```

図３．変換IIの結果（一部）

```
【3c】pop(push(s1,e1)) = 【6a】s1
【4c】top(push(s1,e1)) = 【6f】NULL
```

図４．変換IIIにより残った等式（一部）

４．評価

表１に，いくつかの抽象データ型について，抽象データ型の操作数，生成された等式数，及び，選択規準を適用した回数を示す．このうち，STACK,QUEUE,LISTについてはエラーが等式に反映したが，EDITORについては選択規準⑤を適用すると等式が残らないため，適用していない．

表１．各ＡＤＴのサイズと選択規準適用回数

ＡＤＴ名		STACK	QUEUE	LIST	EDITOR
入力操作数		5	5	6	9
選択規準	①	6	8	4	１５６
	②	7	4	4	２２
	③	3	2	0	７５
	④	1	1	0	0
	⑤	6	5	0	—
	計	２３	２０	8	２５３
出力等式数		5	5	2	１４

本研究における変換手法では，エラーの種類によって，変換後の等式にそのエラーが反映しなかったり，複数のエラーによって等式にエラーが反映しないこともある．前者の場合は，変換IIIの選択規準を⑤から順にゆるめることによって対応可能であるが，後者の場合は，本手法では発見不可能である．

５．おわりに

本研究では，前提終了条件表記により記述された仕様から代数的仕様に変換する手法を与えた．この手法は，記述性のよい前提終了条件表記の仕様から，動作が直感的でわかりやすい代数的仕様に，仕様の表現形式を変換することにより，前提終了条件表記の仕様に含まれるエラーの検出を支援する．

本変換手法では，発見的な方法で前提終了条件表記の仕様から代数的仕様へ変換している．そのため，前提終了条件表記の仕様の完全なテストを行なうには，他のテスト手法と組み合わせて使用すべきである．今後は，前提終了条件表記の仕様から高級言語への変換と，前提終了条件表記の仕様からのテストデータ抽出を合わせて，テストの自動化を行ない，本手法と併用する必要があろう．

参考文献

[1] Beichter, F., Herzog, O. and Petzsch, H., “SLAN-4 － A Software Specification and Design Language,” IEEE Trans. Software Eng., vol. SE-10, pp. 155-162, 1984.

[2] 稲垣康善，“抽象データ型の概念と仕様記述法，” 情報処理，Vol. 27, No. 2, pp. 120-128, 1986.