B-034

隠蔽代数に基づく命令型プログラム言語の意味論の記述と検証
Description and Verification of Semantics of Imperative Programming Language Based on Hidden Algebra

渡辺 真啓 † 中村 正樹 † 二木 厚吉 †
Masahiro Watanabe Masaki Nakamura Kokichi Futatsugi

1. はじめに

隠蔽代数の性質や有効性を命令型言語の意味論を記述することで明らかにする。具体的には隠蔽代数を記述できる代数仕様言語 CafeOBJ ‡で仕様を与え、各種の検証を行う。命令型言語の意味論は状態遷移機械としてモデル化することができ、また隠蔽代数が状態遷移機械を扱いやすいことから適当な例であると考えられる。

命令型言語の意味論を隠蔽代数に基づいてモデル化することの利点として、等式変形による単純な論理体系を基本にしていること、状態空間に付随する演算を記述者の視点に基づいて定義することにより適度な抽象度で振舞を記述でき、その上でプログラムの振舞等価性や正しさを定義できること、CafeOBJ で記述することで仕様の模擬実験や検証を行えることが挙げられる。

2. 命令型言語の意味論のモデル化

2.1 構文定義

言語の構文は整数式、条件式、プログラムにより構成される。まず、整数式と条件式は次のとおりである。

```
mod! EXP{
  pr(INT + BOOL)
  [Int Var < Exp] [Bool < Tst]
  ops (_+_) (_-_) (_*_) : Exp Exp -> Exp
  op -_ : Exp -> Exp
  ops (_<_) (_=_) (_<=_) : Exp Exp -> Tst
  ops (_and_) (_or_) : Tst Tst -> Tst
  op (not_) : Tst -> Tst
}
```

pr(...) で CafeOBJ 内蔵の整数モジュール INT および論理値モジュール BOOL を輸入する。変数を表す Var ソートを宣言する。それらを拡張する形で整数式 Exp, 条件式 Tst を定義する。例えば、変数 x, y に対し、(- x) * (2 + y) は整数式、(x < 2) and (y < 2) は条件式である。

次にプログラムの構文を以下に示す。

```
mod! PGM{
  pr(EXP)
  [BPgm < Pgm]
  op _:=_ : Var Exp -> BPgm
  op _;_ : Pgm Pgm -> Pgm
  op if_then_else_fi : Tst Pgm Pgm -> Pgm
  op for_do_od : Int Pgm -> Pgm
}
```

Pgm はプログラムを表すソートである。代入、連結文、条件文、繰り返し文を記述できる。代入文は基本要素という意味で BPgm で定義している。以下はプログラムの例である。

```
y := 1 ; x := y + 1
if x < y then t := x else t := y fi
```

2.2 意味定義

隠蔽代数では、対象システムは隠蔽ソートとしてカプセル化され、システムの状態を変化させる操作演算と、情報を得る観測演算によりその振舞が定義される。ここでは、プログラムの実行環境（意味システム）を隠蔽ソートとし、操作演算としてプログラムの適用、観測演算として式による意味システムの情報獲得（式の評価値）を定義するモデルを提案する。

意味システムを次のように定義する。

```
mod* SYSTEM{
  pr(PGM)
  *[System]*
  op _[_] : System Exp -> Int
  op _;;_ : System Pgm -> System
```

System は隠蔽ソート、_[_] は観測演算、_;;_は操作演算に対応する。意味システムの状態 S に対して、S における式 E の評価値は S [[E]]、プログラム P を適用した状態は S ;; P で表される。

式の評価値は以下のように再帰的に定義される。変数の評価値が与えられれば、式の評価値が一意に定まる。

```
eq S [[I : Int]] = I .
eq S [[E + F]] = S [[E]] + S [[F]] .
...
```

プログラムの意味は意味システムの状態に対する観測値の変化として定義される。

```
vars X Y : Var  var T : Tst  vars P Q : Pgm
eq S ;; (X := E) [[X]] = S [[E]] .
eq S ;; (X := E) [[Y]] = S [[Y]] .
beq S ;; (P ; Q) = (S ;; P) ;; Q .
bcq S ;; if T then P else Q fi = S ;; P
    if S[[T]] = true .
bcq S ;; if T then P else Q fi = S ;; Q
    if  S[[T]] = false .
bcq S ;; for I do P od = S
    if I <= 0 .
bcq S ;; for I do P od =
S ;; P ;; for I - 1 do P od
    if (I > 0) .
}
```

†北陸先端科学技術大学院大学 情報科学研究科
‡homepage http://www.ldl.jaist.ac.jp/cafeobj

3. CafeOBJ によるシミュレーション

CafeOBJ は仕様に現れる等式を書き換え規則と見なして、与えられた項を簡約する機能がある。この機能を用いて記述した仕様のシミュレーションや検証が可能となる。意味システム s における変数 x, y の値がそれぞれ 1 と 2 であるとき (eq s [[x]] = 1, eq s [[y]] = 2) の整数式 x + y + 3、条件式 (x + y) < 4 の値は CafeOBJ のモジュール SYSTEM のもとでの 'red' コマンドの実行により以下のように簡約される。

```
%SYSTEM> red s[[x + y + 3]] .
6 : Int
%SYSTEM> red s[[(x + y) < 4]] .
true : Bool
```

意味システムにプログラムを適用した結果は式による評価値を通して確かめられる。

```
%SYSTEM> red (s ;; (y := 1 ; x := y)) [[x]] .
1 : Int
%SYSTEM> red (s ;; (x := 4 ; y := 3 ;
if x < y then t := x else t := y fi)) [[t]] .
3 : Int
```

変数 x, y の値を交換するプログラム swap が、実際に変数の値を交換しているかを次のように確認することができる (eq swap = (t := x ; x := y ; y := t))。

```
%SYSTEM> red (s ;; swap) [[x]] .
s [[y]] : Int
%SYSTEM> red (s ;; swap) [[y]] .
s [[x]] : Int
```

4. プログラムの等価性

隠蔽代数においては、同じ隠蔽ソートの項同士の振舞等価性が定義される。振舞等価性の直観的な意味は、二つの状態にどのように操作演算を作用させて観測しても観測値に違いが認められない、というものである [2]。

振舞等価性 二つの隠蔽ソートの項 s ,s' が振舞等価であるとは、任意の観測演算 o と任意の操作演算 a_1,..., a_n に対して、

$$\mathbf{o}(\mathbf{a}_n(...(\mathbf{a}_1(s)))) = \mathbf{o}(\mathbf{a}_n(...(\mathbf{a}_1(s'))))$$

であることであり、このとき $s \sim s'$ と書く[§]。 ⊣

振舞等価性を利用してプログラムの等価性を次のように定義することで、振舞に基づいたプログラムの等価性を議論できるようになる。

定義 1 二つのプログラム $P1$, $P2$ が等価であるとは、任意の意味システム S にたいして、

$$S;;P1 \sim S;;P2$$

となることである。⊣

定義 1 よりプログラムの等価性を証明するためには、任意のプログラムの列の適用と任意の式について調べる必要があり手間がかかる。しかし実際には、次の定理が成立する。

定理 1 二つのプログラム $P1$, $P2$ が等価であるための必要十分条件は、任意の意味システム S と任意の変数 X に対して

$$(S;;P1)[[X]] = (S;;P2)[[X]]$$

となることである。 ⊣

つまり、プログラムの等価性を示すには、各変数の値がプログラム適用後に等しいことを示せば良い。

次の二つの変数交換プログラムが等価であることを示す。

```
eq swap = t := x ; x := y ; y := t ; t := 0

eq swap' = x := (x + y) ; y := (x - y) ;
           x := (x - y) ; t := 0
```

変数 V を、x, y, t とそれ以外の変数について場合分けし、S ;; swap [[V]] == S ;; swap' [[V]] を 'red' コマンドで簡約するとそれぞれ true を返す。よって定理 1 より swap と swap' は等価である。

5. 並行プログラム

提案した仕様を拡張して、並行プログラムの意味を定義する。並行計算を表す構文として以下の演算子を宣言する。

```
op _||_ : Pgm Pgm -> Pgm
```

並行プログラム実行後の意味システムの状態の観測値は、すべてのプロセスの可能な実行順序に対してそれぞれの観測値を参照し、その集合として定義する。例えば、次のプログラムを考える。

```
(x := 1 ; y := 1 || x := 2 ; y := 2)
```

このプログラムの実行後の式 x + y の値は 2 | 3 | 4 となる。並行プログラムの等価性は観測値の整数の集合の等価性によって与えられる。

また、並行プログラムでは欠かせない不可分実行の機構を定義している。

6. まとめと今後の課題

命令型プログラムの意味論を隠蔽代数に基づいて定義した。隠蔽代数で定義することにより、状態遷移機械としてのプログラムの意味論を意味システムの振舞という観点から定義した。プログラムの等価性を振舞等価性に基づいて定義した。等価性を確かめるためのより簡単な十分条件を導いた。また CafeOBJ の実行によりシミュレーションや検証を行った。

今回の仕様では停止しないプログラムは構文的に記述できないようになっている。なぜならプログラムの意味を実行停止後の観測値によって与えているためである。実際、停止しないプログラムを許すと、定理 1 が成立しなくなる。しかし、一般に並行プログラムは停止しないことが前提であることが多く、停止しないプログラムの等価性について意味のある議論の可能性を検討する必要がある。また、より実用的なプログラムの例を記述して各種検証を行うことも今後の課題とする。

参考文献

[1] J.Goguen, G.Malcolm: Algebraic Semantics of Imperative Programs, MIT-Press, 1996

[2] Razvan Diaconescu, Kokichi Futatsugi: Behavioral Coherence in Object-Oriented Algebraic Specification In the Journal of Universal Computer Science, Springer, 2000

[§] ただし $\mathbf{a}_i(s)$ はデータ項 t_i を引数にとった $a_i(s, t_1, ..., t_n)$ の略記である。$\mathbf{o}(s)$ についても同様。