プログラミング言語論

プログラムの意味論

水野嘉明

目次

2

1. 意味論とは

- プログラムの意味論とは
  プログラムに意味を与える方法

3

1. 意味論とは

- プログラムの意味の定義
  - 自然言語による記述
  - ハードウェアの動作としての理解
  - 既知のプログラミング言語による類似の文・式の意味の記述

厳密に定義することは困難

4

1. 意味論とは

- 例

CやJavaの 後置++演算子は、「式を評価した後、その変数の値を1増やす」と説明されることが多い

「x=1; x = x++」で何が起きるかは、分かりにくい

5

1. 意味論とは

- 厳密で矛盾なく意味を定義するには
  - 計算機とは独立して定義できる
  - 定義が厳密で、曖昧さがない
  - プログラムの推論・解析に役立つ

等が必要

形式的意味論（formal semantics）

6

1. 意味論とは

- 形式的意味論の目的
  プログラムの意味を厳密に定義

  ➢プログラムの正しさを形式的に検証
  ➢検証を支援するツールの作成
  ➢自動証明法の研究
  ➢プログラミング言語の設計の検証

7

1. 意味論とは

- 形式的意味論の種類
  ➢操作的意味論 (Operational Semantics)
  ➢表示的意味論 (Denotational Semantics)
  ➢公理的意味論 (Axiomatics Semantics)

色々な意味論が提唱されているが、おおむねこの3つのどれかに分類されるか、あるいはこれらの組み合わせである

8

1. 意味論とは

- 形式的意味論の概要
  ➢操作的意味論
    - プログラムの意味を、抽象的な計算機の動作(状態遷移)として記述する

9

1. 意味論とは

- 表示的意味論
  - プログラムの意味を、ある数学的な関数として定式化する
- 公理的意味論
  - プログラムに関する公理と推論規則を与え、定理としての意味を導く

10

2. 操作的意味論

2.1 定義
2.2 意味関数
2.3 文の意味

11

2. 操作的意味論

- C言語の非常に小さいサブセットの形式的意味記述を考える
  - main関数のみ
  - 型は整数型 int のみ
  - 代入文、および基本的な制御文
  - 簡単な入出力文

12

2.1 定義

プログラム例

```
void   main() {
   int    x, y, s, d;
   scanf("%d", &x);
   scanf("%d", &y);
   s = x + y;  d = x - y;
   printf(%d   %d¥n", s, d);
}
```

13

2.1 定義

- 状態の定義
  - 変数名の集合を Var とする
  - 値の集合を Value とする

前ページのプログラム例ならば、

Var = {x, y, s, d, input, output}
Value = (Z∪{¥n})* ∪ {true, false}

14

2.1 定義

- 定義域をVar、値域をValueとする関数

  σ : Var → Value

  を*状態*と呼ぶ
- 状態の集合を Sとする

  S = {σ | σ : Var → Value}

15

2.1 定義

注: 状態σとは

- 各変数にどのような値が対応しているか、が状態である
- σ(x) とは、状態σにおける変数xの値である
- 変数と値の対応が変わる(つまり変数の値が変わる)と、別の状態となる

16

2.1 定義

- σ[a/x]
  - 状態σから変数 xの値が aに変化した、新しい状態を

    $\underline{\underline{\sigma[a/x]}}$

    と書く

17

2.1 定義

- σ ∈S、a∈Value、x∈Varのとき
  σ[a/x]∈S であり、

  (σ[a/x])(x) = a

  (σ[a/x])(y) = σ(y)
  (ただし、y≠x)

  となる

18

2.1 定義

- 例
  状態σ、σ'∈S について、ある変数 x(∈Var)以外のすべての変数 yに対して σ(y) = σ'(y) の時、
  $\sigma' = \sigma[\sigma'(x)/x]$
  である

19

2.2 意味関数

- プログラムの文の意味を
  状態σ∈Sを別の状態σ'∈Sにかえる関数
  とする
- 文に対し、その文の意味を返す関数を *意味関数* と呼び、**M**で表わす
  M[[文]]: S→S

20

2.2 意味関数

注: **M**は関数であるので、
M(文)
と書くべきであるが、プログラムの文を引数とするときは
M[[文]]
と書くことが多い

21

2.2 意味関数

- 代入文 x = a; の意味

  M[[x=a;]] = $\lambda\ \sigma.\sigma[a/x]$

  ※文「 x=a; 」の意味は、

  状態σの変数xの値を 定数の値aで置き換えて新しい状態とすること

  ※λ記法を使わずに書くと

  M[[x=a;]](σ) = $\sigma[a/x]$

22

2.2 意味関数

- 文の並び 文$_1$文$_2$…文$_n$ の意味 （n>1）

  M[[文$_1$文$_2$…文$_n$]] =

  λσ. **M**[[文n]] (**M**[[文$_1$…文$_{n-1}$]](σ))

  状態σが文$_1$によりσ$_1$になり、文$_2$によりσ$_2$になり、…、文$_n$により状態σ$_n$となる

23

2.2 意味関数

- 式の意味

  f : S → Value

  状態に応じて値を対応させる関数

  式には値が対応するが、状態によってその値は異なる

- 式の意味を与える関数

  E[[式]]: S → Value

24

2.2 意味関数

- $x \in Var$、$c \in Z \cup \{true, false\}$、$e_1$と$e_2$を式とするとき、
  - $\mathbf{E}[[x]](\sigma) = \sigma(x)$
  - $\mathbf{E}[[c]](\sigma) = c$
  - $\mathbf{E}[[(e_1)]] = \mathbf{E}[[e_1]]$
  - $\mathbf{E}[[e_1 \text{ op } e_2]](\sigma) = \mathbf{E}[[e_1]](\sigma) \ \overline{\text{op}} \ \mathbf{E}[[e_2]](\sigma)$

25

2.2 意味関数

- 前記の定義は、
  - 変数のみの式の意味は、その状態での変数の値
  - 定数のみの式の意味は、その定数の値
  - 括弧でくくった式の意味は、括弧の中の式の意味

26

2.2 意味関数

- ($式_1$ op $式_2$) の意味は、
  ($式_1$の意味) $\overline{\text{op}}$ ($式_2$の意味)

ただし、opは(その言語での)2項演算子、$\overline{\text{op}}$ は、対応する(数学的に定義された)演算子
(+ ⇔ +、&& ⇔ ∧、<= ⇔ ≦ 等)

例: $\mathbf{E}[[e_1 * e2]] = \mathbf{E}[[e_1]] \times \mathbf{E}[[e_2]]$

27

2.3 文の意味

- 代入文 x = *e* ; の意味

M[[x=*e* ;]] = λ σ.σ[**E**[[*e*]](σ)/x]

状態σにおける式*e*の値

この値でxを置き換えて新しい状態とする

28

2.3 文の意味

例: 文「 s=x+y; 」の意味

M[[s=x+y;]] = λ σ.σ[σ(x)+σ(y)/s]

状態σの変数sの値を その状態での変数xの値σ(x)と変数yの値σ(y)の和で置き換えて新しい状態とすること

29

2.3 文の意味

- if 文の意味

M[[if (*e*) s_1 else s_2]] =
λ σ. if-then-else(**E**[[*e*]](σ),
M[[s_1]](σ), **M**[[s_2]](σ))

もし **E**[[*e*]](σ)が真ならば、**M**[[s_1]]、さもなければ **M**[[s_2]]が、この if文の意味

30

2.3 文の意味

注: if-then-elseの評価は、
- 第1引数をまず評価し、
- それが真であれば第2引数を
- 偽であれば第3引数を評価する

31

2.3 文の意味

- if 文の意味 (2)

$\mathbf{M}$[[if (*e*) s_1]] =
$\lambda\ \sigma$. if-then-else($\mathbf{E}$[[*e*]](σ),
$\mathbf{M}$[[s_1]](σ), σ)

else節がないので、$\mathbf{E}$[[*e*]](σ)が偽ならば
状態σは変わらない

32

2.3 文の意味

- while文「while(*e*) s_1」の意味

$\mathbf{M}$[[while(*e*) s_1]] =
$\lambda\ \sigma$. if-then-else($\mathbf{E}$[[*e*]](σ),
$\mathbf{M}$[[while(*e*) s_1]]($\mathbf{M}$[[s_1]](σ)),
σ)

33

2.3 文の意味

- f = **M**[[while(*e*) s_1]] とおいて
  前ページの式を書き換えると

f (σ) = if-then-else(**E**[[*e*]](σ),

f (**M**[[s_1]](σ)), σ)

式eが環境σで真ならば、s_1を実行し
新たな環境で再度 f を評価する。
さもなければ、σは変化しない。

34

3. 表示的意味論

3.1 表示的意味論とは
3.2 表示的意味論における意味

35

3.1 表示的意味論とは

- 表示的意味論とは
  プログラムの集合をPとする
  数学的な値の集合（意味領域）Dと
  意味関数 **M**：P→D の対 （D,**M**）で
  意味を与える方法

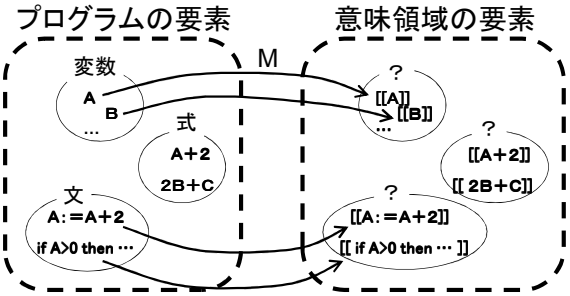
36

3.1 表示的意味論とは

- $p \in$ Pの意味を $\mathbf{M}[[p]] \in$ Dで与えることから、p は$\mathbf{M}[[p]]$を指し示す(denote)といい、$\mathbf{M}[[p]]$ を p の*表示*(denotation)、あるいは*表示的意味*(denotational meaning)という
  ⇒ ここから、表示的意味論という

37

3.1 表示的意味論とは



38

3.2 表示的意味論における意味

- プログラムの実行に伴う状態の変化を記述する点は操作的意味論と同じだが、それを*数学的手法* でやるところが違う

39

3.2 表示的意味論における意味

- 操作的意味論では、次の関数を定義した
  - **文**の意味を与える関数
    M[[文]]: S→S
  - 式の意味を与える関数
    E[[式]]: S → Value

40

3.2 表示的意味論における意味

- 次のような式や文の意味は、操作的意味論と同様
  - **E**[[x]](σ) = σ(x)
  - **E**[[c]](σ) = c
  - **E**[[e_1 op e_2]](σ) = **E**[[e_1]](σ) op **E**[[e_2]](σ)
  - **M**[[x=*e* ;]] = $\lambda\ \sigma.\sigma$[**E**[[*e*]](σ)/x]

41

3.2 表示的意味論における意味

- while 文の意味
  - 操作的意味論では、if-then-else を用いて、操作的に定義
  - 表示的意味論では、数学的に厳密な意味として、次の関数 F の最小不動点を while文の意味とする

42

3.2 表示的意味論における意味

- while文「while(*e*) s_1」の意味
  ≡ 次の関数Fの最小不動点

$$F = \lambda f.\lambda\sigma.\ \text{if-then-else}(\mathbf{E}[[e]](\sigma),\ f(\mathbf{M}[[s_1]](\sigma)),\ \sigma)$$

（不動点とは F(f)=f を満たす関数f のこと）

43

3.2 表示的意味論における意味

- プログラムの意味をきわめて厳密に取り扱うことができる
- 難解な数式になってしまうところが欠点

44

4. 公理的意味論

4.1 準備
4.2 表明
4.3 公理と推論規則
4.4 プログラムの正当性

45

4. 公理的意味論

- ホーア(Hoare)の公理系と呼ばれる
  - 公理と推論規則(Hoareの論理)を与える
  - 得られる結果(定理)をプログラムの意味とする

46

4.1 準備

- 言葉の意味
  - *公理* (axiom)
    - その他の命題を導きだすための前提として導入される最も基本的な仮定のこと
  - *定理* (theorem)
    - 公理を前提として推論規則によって導きだされた命題

47

4.1 準備

  - *表明* (assertion)
    - プログラムの中の特定の場所で変数の値の間に成立する条件式

48

4.1 準備

- 推論規則の書式

$$\frac{S1 \quad S2 \quad S3 \quad \cdots}{S}$$

とは、横線の上の式がすべて成立するならば、横線の下側の式が成立することを示している

49

4.1 準備

- 例1

$$\frac{a=b \quad b=c}{a=c}$$

- 例2

$$\frac{A \quad A \Rightarrow B}{B}$$

50

4.1 準備

- Aを論理式、xを変数、eを式とする
  Aの中に出現するすべてのxを、eで置き換えた式を

$$A[e/x]$$

で表わす

51

4.1 準備

- 例1

  (y=x)[a/x] = (y=a)

- 例2

  ((x+y) < (x−z))[x−y/x]

  = ((x−y)+y) < ((x−y)−z)

52

4.2 表明

- Hoareの論理で扱う 表明 (assertion)

  {A}P{B}

  AとBは論理式、Pはプログラム
  またはプログラムの文

- プログラムPの実行前にAが成立

- Pを実行前し停止した後はBが成立

53

4.2 表明

- A を*事前条件* (precondition)、
  B を*事後条件* (postcondition)
  という
- {A}P{B}は、次の論理式と等価

$$\forall \sigma, \sigma' \in S, (\mathbf{E}[[A]](\sigma) \wedge \mathbf{M}[[P]](\sigma) = \sigma') \Rightarrow \mathbf{E}[[B]](\sigma')$$

54

4.3 公理と推論規則

- 公理（代入の規則）

$$\{A[e/x]\}\, x=e;\, \{A\}$$

ここで、xは変数、eは式、Aは論理式である

55

4.3 公理と推論規則

- 推論規則(1) -- 文の並びの規則

$$\frac{\{A\}\,文_1\,\{B\}\quad \{B\}\,文_2\cdots 文_n\,\{C\}}{\{A\}\,文_1 文_2\cdots 文_n\,\{C\}}$$

$$\frac{\{A\}\,文_1 文_2\cdots 文_n\,\{C\}}{\{A\}\{文_1 文_2\cdots 文_n\}\{C\}}$$

56

4.3 公理と推論規則

- 推論規則(2) -- 条件文の規則

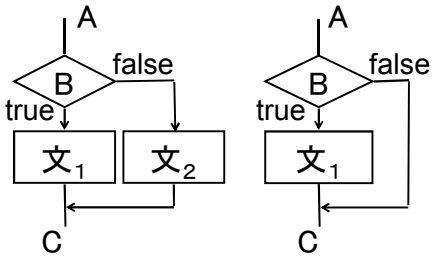
$$\frac{\{A\wedge B\}\,文_1\,\{C\}\quad \{A\wedge\neg B\}\,文_2\,\{C\}}{\{A\}\ \text{if}\ (B)文_1\text{else}文_2\,\{C\}}$$

$$\frac{\{A\wedge B\}\,文_1\,\{C\}\quad A\wedge\neg B\Rightarrow C}{\{A\}\ \text{if}\ (B)文_1\,\{C\}}$$

57

4.3 公理と推論規則

➢条件文に対する流れ図



58

4.3 公理と推論規則

● 推論規則(3) -- 帰結の規則

$$\frac{A \Rightarrow A' \quad \{A'\}文\{B\}}{\{A\}文\{B\}}$$

$$\frac{\{A\}文\{B'\} \quad B' \Rightarrow B}{\{A\}文\{B\}}$$

59

4.3 公理と推論規則

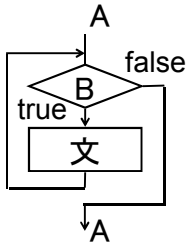
● 推論規則(4) -- while文の規則

$$\frac{\{A \wedge B\}文\{A\}}{\{A\}\ \text{while}(B)文\ \{A \wedge \neg B\}}$$

60

4.3 公理と推論規則

➢while文に対する流れ図



Aを
ループ不変表明
という

61

4.4 プログラムの正当性

- プログラムPに要求される条件を事後条件Bとする

表明 {A}P{B}が定理として得られる
（Pの実行前にAが成立するとき、Pを実行し停止した後はBが成り立つ）

Pの正当性を保障する条件＝A

62

4.4 プログラムの正当性

- 例題：次の文 P を考える

```
P = { i = 1; x = 1;
      while(i ≦ n) { x = x * i; i = i + 1; } }
```

プログラム Pの意図は、x＝n!
{A}P{x=n!} が定理となるAを求めてみる

63

4.4 プログラムの正当性

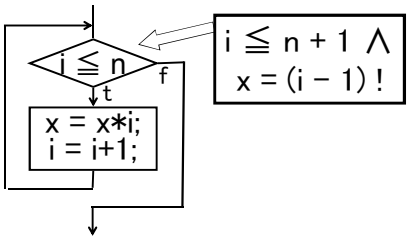
- while文の推論規則を適用するには、
  $\{A_1 \wedge (i \leqq n)\}$ {x=x*i; i=i+1;} $\{A_1\}$
  となる ループ不変表明 (loop invariant assertion) A_1が必要

$$i \leqq n + 1 \wedge x = (i - 1)!$$

64

4.4 プログラムの正当性

- ループ不変表明A_1は、whileの条件 ($i \leqq n$) 判定時に、常に成立



65

4.4 プログラムの正当性

- このループ不変表明A_1を用いると、事前条件を {n≧0}とした

$$\{n \geqq 0\}\ P\ \{x = n!\}$$

が定理として導き出せる

- これにより、n≧0ならばPが正しいことが証明できた

66

4.4 プログラムの正当性

➢この定理の導出（証明）は、別紙「例題の証明」に記載しておく

（例題終了）

67

4.4 プログラムの正当性

- Hoareの論理を用いて プログラムの正当性を示すには、
  - ➢適切なループ不変表明を見つけることが必要
  - ➢見つけることは 難しい
  - ➢必ず存在する

68

5. 各意味論の関係

- 意味記述の抽象度
  - ➢操作的意味論
  - ➢表示的意味論
  - ➢公理的意味論

低

高

69

5. 各意味論の関係

- 各意味論の間には、密接な関係

↓

- どれも、プログラミング言語で書かれたプログラムの意味を正確に理解するための道具

70

お疲れ様でした