言語の意味論2
―公理的意味論―

2007

3種類の意味論・意味定義法(補足)

- 操作的:

  プログラムを解釈・実行するという操作そのものに近いレベルで与えられる。

- 公理的:

  プログラムが持っている性質や機能の仕様を記述することができる。同時にその仕様をプログラムが満たすか否かを検証する方法を与える。

- 表示的:

  プログラムに対応する数学的な存在(e.g.,関数)を与える。 プログラムの数学的なモデル。

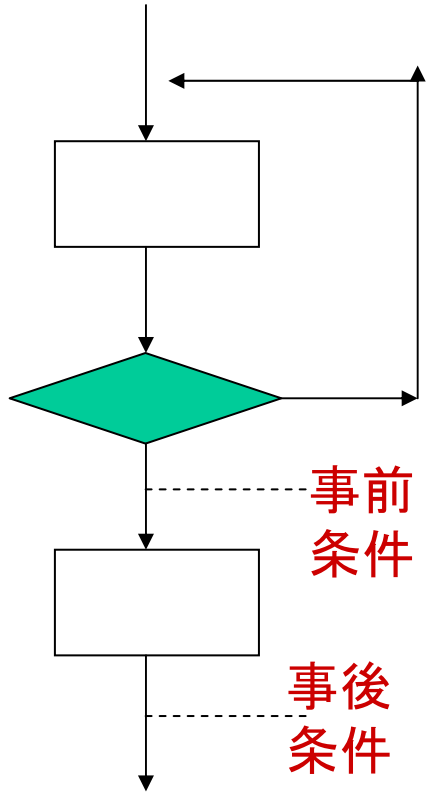
公理的方法の歴史と基本的アイデア（1）

- R.Floydがフローチャート言語の各ノードの前後で成立する論理的表明（assertion）、即ち事前条件(pre-condition)と事後条件(post-condition)をユーザに書かせ、

  （1）事前条件が成立すると仮定し、かつ

  （2）ノードの実行が終了するならば、そこで成立する事後条件を各ノードから生成する方法を示した（1967）。
- （1）（2）によって、事後条件の成立を証明することを、プログラムの正当性(correctness)

  の証明(proof)あるいは検証(verification) と呼ぶ。
- 連結するノードをまとめてひとつのノードとみなす適当な合成規則をあたえれば、複数のノード（や、ループ）を含むフローチャート言語の「意味」を与えることが出来る。



歴史と基本的アイデア（2）

- C.A.R. Hoareは、R.Floydの着想をPascal/Algol風の命令型言語に対して熟考し、言語で許される文/各命令ごとに、公理や推論規則を与えることに成功した（1968）。
- この公理系は、プログラムの文面をそのまま論理体系に組み込んで、文面を形式的な対象物(formal objects)として扱えるようにしたため、大変強力でその後の言語の形式的な研究に、大きな影響力を与えた。
- 前に話した、操作的意味論もG. Plotkinによって文面をformal objectsとして扱えるようにしたのは、Hoareの仕事に負うところが大きい。→自動化が容易

例題となるIMPのプログラム

Here is a program in IMP to compute $\Sigma_{1<m<100} m$ (The notation $\Sigma_{1\leq m\leq 100} m$ means 1+2+･･･+100).

```
S := 0;

N := 1;

(while ¬(N = 101) do S := S + N ; N := N + 1)
```

このプログラムが、終了したとき S の値が $\Sigma_{1\leq m\leq 100} m$ となっていることを、どのように証明するか？プログラムに意味が操作的に与えられると、証明困難

プログラムのより一般的な性質の証明

- このプログラムにおいて “$\mathbf{while}\ \neg(N = 101)\ \mathbf{do}\ \cdots$” の部分を、“$\mathbf{while}\ \neg(N = P + 1)\ \mathbf{do}\ \cdots$” に置き換えて、かつプログラムの実行の直前に、$P$ に任意の値を代入したとする。
- この場合でも、S の値は $\sum_{1 \leq m \leq \cancel{100}\ P} m$ と考えられるが、このような（簡単な）一般的な性質を証明するためには、プログラムの実行・解釈をベースにする操作的意味論では、なかなか難しい。

プログラムへの注釈に記法として

S := 0 ; N := 1;

{S = 0 ∧ N = 1}

(while ¬(N = 101) do S := S + N ; N := N + 1

プログラムが終了時の状態への注釈

We want a method to justify the final comment

$S := 0 ; N := 1$

$\{S = 0 \wedge N = 1\}$

$(\text{while } \neg(N = 101) \text{ do } S := S + N ; N := N + 1$

$\{S = \Sigma_{1 \leq m \leq 100} m\}$

- もし、whileループの前に$S = 0 \wedge N = 1$が成立すれば、

実行の直後、$S = \Sigma_{1 \leq m \leq 100} m$が成立することを主張すると同時に、$N = 101$も成立している。

Hoareの記法

- プログラム文面 c の直前で述語 A が成立し、かつプログラム c の実行が終了するならば述語 B が成立することを

$\{A\}\ c\ \{B\}$　あるいは　$A\ \{\ c\ \}\ B$

と表記する。 これをHoare記法と呼ぶ。

Hoare記法を用いて
プログラムが終了時に成立する主張

We want a method to justify the final comment

$S := 0 ; N := 1$

$\{S = 0 \land N = 1\}$

(while $\neg(N = 101)$ do $S := S + N ; N := N + 1$

$\{S = \Sigma_{1 \leq m \leq 100} m \quad \land \quad N = 101\}$

- もし、whileループの前に$S = 0 \land N = 1$が成立すれば、

実行の直後、$S = \Sigma_{1 \leq m \leq 100} m$が成立することを主張すると同時に、$N = 101$も成立している。

$\{A\}c\{B\}$

- この記法の解釈

  for all states σ which satisfy A if the execution of c from state σ terminates in state σ' then σ' satisfies B.

- 別の言い方をすれば:

  {A}c{B} means that any successful (i.e., terminating) execution of c from a state satisfying A ends up in a state satisfying B.

ループ不変式(loop invariant)

We can look and see what the values of S and N are the first time round the loop, S = 1, N = 2. And the second time round the loop S = 1 + 2, N = 3･･･and so on, until we see the pattern: after the i-th time round the loop S = 1 + 2 +･･･+ i and N = i + 1. From which we see, when we exit the loop, that S = 1 + 2 +･･･+100, because when we exit N = 101.

- ループの始まりと終わりに:

S = 1 + 2 + 3 + ･･･ + (N – 1)

が成立するので、ループを回るごとにこの式が成立する。これはSとNがプログラム実行中もつ基本的な関係である。一般にこのような関係を**ループ不変式(loop invariant)**と呼ぶ。

ループ不変式

- ループや再帰があるプログラムを検証・証明をする上で肝である。
- 必ず存在するが、一般には自動的に生成はできない。すなわち、一般的なこれを求めるあるアルゴリズムはない。人間が発見しなければならない。
- 近似的・発見的に求める方法は研究されてきた。

部分正当性(partial correctness)

- Assertions of the {A}c{B} are called partial correctness assertions because they say nothing about c if it fails to terminate.

極端な例として次のようなプログラム C を考える:

C ≡ while true do skip.

C の実行はどのような状態から始めても、終了しない。

上の解釈に従えば、C に対する次のような部分正当性表明は論理的validである。

{true}C{false}

simply because the execution of c does not terminate.

表明用言語(assertion language)

- プログラムの性質を表現するための厳密に定義された論理学的な言語を与えておく必要がある。それが表明用言語である。
- 対象プログラムの***Aexp***を拡張して整数の変数 *i,j,k* 等を許し、次のような算術式***Aexpv***を定義しておく:

$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$

ただし

n ranges over numbers, **N**

X ranges over locations, **Loc**

i ranges over integer variables, **Intvar**.

- これをもとに、次のような論理式を表明言語として許す。

$$A ::= true \mid false \mid a_0 = a_1 \mid a_0 \leq a_1 \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \forall i.A \mid \exists i.A$$

自由変数、代入等

- 普遍記号や存在記号を表明言語に許すので、当然、自由変数や束縛変数の問題がでてくる。(λ計算を参照のこと)
- 代入について直感的に説明すれば:

We can picture an assertion A as

A ≡ ---i---i—

say, with free occurrences of the integer variable i. Let a be an arithmetic expression, which for simplicity we assume contains no integer variables. Then

A[a/i] ≡ ---a---a- —

is the result of substituting a for i. If a contained integer variables then it might be necessary to rename some bound variables of A in order to avoid the variables in a becoming bound by quantifiers in A — this is how it's done for general substitutions

表明言語に関するその他の点

- 表明言語の表現力
  - 表明言語がプログラムの実行現象を記述しきれるか？
  - 数論を含んだ述語論理には、(プログラムとは無関係に)公理から導出できないが真である論理式がある。(ゲーデルの不完全性定理)

Hoare Logic (推論規則)

Rule for skip: $\{A\}\text{skip}\{A\}$

Rule for assignments: $\{B[a/X]\}X := a\{B\}$

Rule for sequencing:
$$\frac{\{A\}c_0\{C\} \quad \{C\}c_1\{B\}}{\{A\}c_0 ; c_1\{B\}}$$

各構文単位について、

最弱事前条件と最強事前条件を与えている。

Rule for conditionals:
$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\} \text{ if b then } c_0 \text{ else } c_1\{B\}}$$

Rule for while loops:
$$\frac{\{A \wedge b\}c\{A\}}{\{A\} \text{ while b do c}\{A \wedge \neg b\}}$$

A is loop invariant

Rule of consequence:
$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

代入規則の適用例

- {B[a/X]} X:= a {B}

  では、{ ?? } X:=X+1{X=2}

- { ?? } k:= (m+n)/2 {m> k>n}

証明の例

As an example we show in detail how to use the Hoare rules to verify that the command

$$W \equiv (\text{while } X > 0 \text{ do } Y := X \times Y ; X := X - 1)$$

does indeed compute the factorial function $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$, with 0! understood to be 1, given that X = n, a nonnegative number, and Y = 1 initially. More precisely, we wish to prove:

$$\{X = n \wedge n \geq 0 \wedge Y = 1\}\ W\ \{Y = n!\}.$$

To prove this we must clearly invoke the proof rule for while-loops which requires an invariant I. Take

$$I \equiv (Y \times X! = n! \wedge X \geq 0).$$

証明例のつづき

We show I is indeed an invariant i.e.

$\{I \wedge X > 0\}Y := X \times Y ; X := X - 1\{I\}.$

I は規則表で *A*であった。

なぜなら、From the rule for assignment we have

$\{I[(X-1)/X]\}X := X - 1\{I\}$

where $I[(X-1)/X] \equiv (Y \times (X-1)! = n! \wedge (X-1) > 0)$.

Again by the assignment rule: $I[(X-1)/X]\ [(X \times Y)/Y]$

$\{X \times Y \times (X-1)! = n! \wedge (X-1) > 0\}Y := X \times Y; X := (X-1)\{I\}.$

Clearly

$I \wedge X > 0 \Rightarrow Y \times X! = n! \wedge X \geq 0 \wedge X > 0$

$\Rightarrow Y \times X! = n! \wedge X \geq 1$

$\Rightarrow X \times Y \times (X-1)! = n! \wedge (X-1) \geq 0.$

証明の残り

Thus by the consequence rule

$$\{I \wedge X > 0\} Y := X \times Y \,;\, X := (X - 1)\{I\}$$

establishing that I is an invariant.

I は規則表で*A*であった。

Now applying the rule for while-loops we obtain

$$\{I\}\ W\ \{I \wedge X \not> 0\}.$$

Clearly $(X = n) \wedge (n \geq 0) \wedge (Y = 1) \Rightarrow I$, and

$$I \wedge X \not> 0 \Rightarrow Y \times X! = n! \wedge X \geqq 0 \wedge X \not> 0$$

$$\Rightarrow Y \times X! = n! \wedge X = 0$$

$$\Rightarrow Y \times 0! = Y = n!$$

Thus by the consequence rule we conclude

$$\{(X = n) \wedge (Y = 1)\}\ W\ \{Y = n!\}.$$

2つの宿題 (締め切り12/17講義)

Exercise 6.13 Prove, using the Hoare rules, the correctness of the partial correctness assertion:

$\{1 \leq N\}$

P := 0;

C := 1;

(while $C \leq N$ do P := P + M ; C := C + 1)

$\{P = M \times N\}$

Exercise 6.14 Find an appropriate invariant to use in the while-rule for proving the following partial correctness assertion:

$\{i = Y\}$ while $\neg(Y = 0)$ do $Y := Y - 1 ; X := 2 \times X$ $\{X = 2^i\}$