08.09.23 ФМ-2 Лекция 1

Основные понятия. Классификация программ. Класс программ-функций. Язык P_0

Противоречие между логикой и вычислимостью
Спецификация и верификация программы
Требования
Классификация программ
Класс программ-функций
Тотальная корректность программ
Язык P_0 и его формальная операционная семантика
http://persons.iis.nsk.su/files/persons/pages/prog.pdf
http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf

Логика и вычислимость программы

Допустим, требуется доказать, что свойство P истинно при завершении исполнения программы П.

$$П \vdash P$$

Для этого нужно определить сильнейший предикат S, истинный при завершении программы П и доказать истинность формулы:

$$S \Rightarrow P$$

Предикат A *сильнее* предиката B $\cong$ $A \Rightarrow B$

Как для программы П построить сильнейший предикат S ?

С помощью *формальной семантики* R языка программирования.

$$S = R(П)$$

Программа П обладает логикой – предикатом S

Программа П автоматически вычислима

Логика первична, вычислимость вторична

Диалектическое противоречие между логикой и вычислимостью

Противоречие между надежностью и эффективностью

Понятие спецификации

Спецификация программы – точное, полное и однозначное описание преобразования данных.

Декларативность. Спецификация предшествует программе.

Требование *корректности* программы. *Программа должна соответствовать спецификации*: набор утверждений, составляющих спецификацию программы, должен быть истинным для значений результатов исполнения программы и входных данных при любом исполнении программы.

Верификация программ

Верификация программы – проверка корректности программы относительно спецификации.

Виды верификации программ: тестирование, экспертиза кода, статический анализ, формальные методы верификации. http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf

Спецификация, записанная на строгом формальном языке спецификации, является *формальной*.

Формальные методы (Formal methods) базируются на формальной спецификации программ.

Классы программ

1. Программы-функции

2. Программы-процессы (реактивные системы)

…… Языковые процессоры

Операционные среды

Цель классификации – разработка адекватной технологии программирования для каждого класса программ.

Окружение программы – программные и аппаратные компоненты, непосредственно взаимодействующие с программой при ее исполнении и реализующие входные и выходные информационные потоки программы.

Полезность методов. Метод полезен для конкретной программы, если его применение дает преимущества по сравнению с некоторой типовой технологией, в рамках который данный метод не применяется.

Требования

Инженерия требований → Системная инженерия
→ Программная инженерия

Стандарт 2011г. по инженерии требований: Systems and software engineering — Life cycle processes — Requirements engineering. ISO/IEC/ IEEE 29148.

Требование — утверждение, определяющее потребность и связанные с ней условия и ограничения.

Постановка задачи

Условия — измеримые количественные или качественные характеристики.

Ограничения — временн'ые, на возможные интерфейсы и платформы, на физические размеры, финансовые, обусловленные существующими в стране законами.

Хорошее требование должно быть:

необходимым — определять существенные свойства и особенности

формулироваться независимым от реализации образом

однозначным — просто формулируемым и однозначно понимаемым

согласованным — не иметь конфликтов по отношению к другим требованиям

полным — описание требования должно в достаточной степени его определять

единичным — определять единственное требование, а не конъюнкцию нескольких

достижимым в пределах существующих ограничений и разумного риска

трассируемым — все связи данного требования с реализацией, документацией и другими артефактами разработки системы или программы должны хорошо прослеживаться

проверяемым — должен существовать простой способ определения того, что требование выполняется

Примерный список типов требований

функциональное требование — функция или действие, определяющее поведение системы или программы

Сценарии применения (use case)

качество функционирования или использования системы или программы

интерфейс — внешний по отношению к внешнему окружению или внутренний для компонентов системы или программы

ограничения реализации (дизайна)

процессуальное требование, обусловленное контрактом или местными законами

нефункциональное требование, которое подразделяется на:

требование качества (живучесть, гибкость, переносимость, сопровождаемость, переиспользуемость, надежность, секретность, ...)

гуманитарное требование (безопасность, эффективность, защищенность, отказоустойчивость...)

Какими требования не должны быть. Следует избегать фраз:

наибольший, наилучший, минимальный, оптимальный,
прост для использования, дружественный интерфейс
почти всегда, существенно
но не ограничивается, как минимум
лучше чем, высокое качество
если возможно, как принято.

Естественный язык (80%), формализованный ЕЯ (15%),
формальный (чаще графический) язык (5%)

Требования vs Спецификации

Нефункциональные требования иногда вырождаются

Инженеры требований – иногда составляют половину персонала

Форум - ликбез по инженерии требований:

http://forum.drakon.su/viewtopic.php?f=140&t=5575&sid=b6526339cadd755498633483bdc80cf6

Класс программ-функций (невзаимодействующих программ)

Программа не взаимодействует с внешним окружением. Программу можно перестроить таким образом, чтобы все операторы ввода данных находились в начале программы, а весь вывод собран в конце программы.

Программа обязана всегда завершаться, поскольку бесконечно работающая и невзаимодействующая программа бесполезна.

Программа - функция, вычисляющая по набору входных данных (*аргументов*) некоторый набор *результатов*.

Задачи дискретной и вычислительной математики

$U(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_k)$ — программа–функция;

$n \geq 0, k > 0$

$x = x_1, x_2, \ldots, x_n$ — *аргументы* — набор входных данных,

$y = y_1, y_2, \ldots, y_k$ — *результаты* — набор выходных данных.

$U(x: y)$ — *операторная* (*предикатная*) форма записи

$y = U(x)$ — *функциональная* форма записи

Спецификация программы–функции:

$P(x)$ — *предусловие*, истинное перед исполнением программы,

$Q(x, y)$ — *постусловие*, истинное после исполнения программы.

Спецификация изображается в виде:

$$[\, P(x), Q(x, y) \,]$$

Однозначность спецификации [P(x), Q(x, y)] :

$$P(x) \,\&\, Q(x, y_1) \,\&\, Q(x, y_2) \Rightarrow y_1 = y_2$$

Тотальность спецификации [P(x), Q(x, y)] :

$$P(x) \Rightarrow \exists y.\ Q(x, y)$$

Конкретизация требования корректности программы для класса программ-функций:

Частичная корректность: если перед исполнением программы U(x: y) истинно предусловие P(x), то в случае завершения программы должно быть истинно постусловие Q(x, y) в момент ее завершения.

Условие завершения программы: если перед исполнением программы U(x: y) истинно предусловие P(x), то программа обязана завершаться.

Тотальная корректность =

частичная корректность + условие завершения

Операционная семантика программы U(x: y) – предикат R(U):

R(U)(x, y) ≅ для значения набора x исполнение программы U всегда завершается и существует исполнение программы, при котором результатом вычисления является значение набора y.

Исключается: для некоторого x существуют два разных исполнения программы, одно из которых завершается, а другое – нет.

Однозначность и тотальность программы U(x: y) для x :

$$R(U)(x, y_1) \,\&\, R(U)(x, y_2) \Rightarrow y_1 = y_2$$

$$\exists y.\ R(U)(x, y)$$

Тотальность программы для набора аргументов x есть в точности условие завершения программы для этого значения x.

Программа *однозначна* – если она однозначна для всех значений аргументов.

Истинность свойства программы $W(x, y)$ реализуется доказательством формулы: $R(U)(x, y) \Rightarrow W(x, y)$.

Формулу достаточно доказать при истинном предусловии.

Условие частичной корректности программы $U(x: y)$:

$$\forall x,y.\ P(x)\ \&\ R(U)(x, y) \Rightarrow Q(x, y)$$

Условие завершения программы $U(x: y)$ при истинном предусловии:

$$\forall x.\ P(x) \Rightarrow \exists y.\ R(U)(x, y)$$

Формула **тотальной корректности** программы:

$$\forall x.\ P(x) \Rightarrow [\forall y.\ R(U)(x, y) \Rightarrow Q(x, y)\ ]\ \&\ \exists y.\ R(U)(x, y) \quad (1)$$

Термин «*корректность*» далее используется в смысле тотальной корректности.

Лемма 1. Если программа корректна относительно спецификации, то спецификация тотальна.

2. Предикатное программирование. 2.1.3. Язык P_0 и его формальная операционная семантика

Цель – определить минимальный полный *базис предикатных программ* в виде языка P_0, построить для него операционную семантику. Расширить P_0 до удобного языка программирования **P**. Разработать методы дедуктивной верификации предикатных программ.

Программа – это функция: $y = U(x)$

Программа – это математический предикат: $U(x: y)$

Корректное вычисленное значение y реализуется при истинности $U(x, y)$. Поэтому для формальной семантики R языка вычислимых предикатов: $R(U) = U$.

Предикат $U(x, y)$ является *вычислимым* относительно x, если существует интерпретатор, вычисляющий значение y, для которого $U(x, y)$ – истинно, по записи предиката на языке исчисления предикатов.

Введем другой, более удобный, язык, так чтобы выполнение соответствующей программы U(x: y) на этом языке было эквивалентно исполнению предиката U(x, y) на языке исчисления предикатов.

Единое обозначение U(x: y) для программы и предиката.

Предикатная программа на языке $\mathbf{P_0}$:

<имя предиката>(<аргументы>: <результаты>)
{ <оператор> }

Аргументы и результаты – непересекающиеся наборы имен переменных.

Вызов предиката B(x: y), где B – имя предиката.

Вызов предиката вида A(x: y), где A – имя переменной *предикатного типа*.

Значение A – имя предиката.

Пусть x, y и z – разные непересекающиеся наборы переменных; x может быть пустым, наборы y и z не пусты. Логическая переменная e со значениями **true** и **false**. Пусть B и C – имена предикатов, A и D – имена переменных предикатного типа.

Операторами являются:

оператор суперпозиции B(x: z); C(z: y),
параллельный оператор B(x: y) || C(x: z),
условный оператор **if** (e) B(x: y) **else** C(x: y),
вызов предиката B(x: y)
и *оператор каррирования* D(y: z) {B(x, y: z)}.

F(x, y) = F(x)(y) - каррирование в функциональном стиле

B(x, y : z) = B(x)(y : z)

Таблица 1. Вычислимые предикаты и их программная форма

Вычислимый предикат	Программа на языке P_0
$H(x: y) \cong \exists z.\ B(x: z)\ \&\ C(z: y)$	H(x: y) { B(x: z); C(z: y) }
$H(x: y, z) \cong B(x: y)\ \&\ C(x: z)$	H(x: y, z) { B(x: y) || C(x: z) }
$H(x: y) \cong (e \Rightarrow B(x: y))\ \&\ (\neg e \Rightarrow C(x: y))$	H(x: y) { **if** (e) B(x: y) **else** C(x: y) }
$H(x: y) \cong B(\tilde{x}: y)$	H(x: y) { B($\tilde{x}$: y) }
$H(A, x: y) \cong A(x: y)$	H(A, x: y) { A(x: y) }
$H(x: D) \cong \forall y,z.\ D(y: z) \equiv B(x, y: z)$	H(x: D) { D(y: z) {B(x, y: z)} }
$H(A, x: D) \cong \forall y,z.\ D(y: z) \equiv A(x, y: z)$	H(A, x: D) { D(y: z) {A(x, y: z)} }

Набор $\tilde{x}$ составлен из x с добавлением имен предикатных программ.

Операторы каррирования: D(y: z) {B(x, y: z)} и D(y: z) {A(x, y: z)}.

Результат оператора – новый предикат D

Полная программа – конечный набор предикатных программ.

Исполнение полной программы начинается с некоторой предикатной программы, называемой *главной*.

Любое имя предиката B, встречающееся в операторной части любой предикатной программы, определено одним из способов:

- в составе полной программы имеется предикатная программа с именем B;
- предикат B является *базисным*;
- предикат B является *параметром* полной программы и определяется в другой полной программе.

Базисный предикат соответствует элементарной операции над числами или логическими значениями и считается предопределенным в языке $\mathbf{P_0}$. Примеры базисных предикатов: +(a, b: c), <(a, b: e), =(a, b: e), где e – логическая переменная. Они соответствуют следующим отношениям: c = a + b, e = (a< b), e = (a = b). Базисный предикат =(a: b) соответствует отношению b = a . Это оператор присваивания.

Эквивалентная *функциональная форма* операторов.

Оператор вызова предиката B(x: y) эквивалентен оператору присваивания y = B(x), где B(x) предстает как соответствующий вызов функции.

Оператор суперпозиции B(x: z); C(z: y) может быть записан в виде y = C(B(x)).

Параллельному оператору B(x: y) || C(x: z) соответствует пара соседних аргументов в составе некоторого другого вызова функции E(…B(x), C(x)…).

Оператор каррирования D(y: z) {B(x, y: z)} может быть записан в виде оператора присваивания D = B(x), где B(x) – каррированая форма вызова функции.

Операционная семантика операторов

Пусть A – переменная предикатного типа. В операционной семантике переменную A будем представлять структурой (A.name, A.pref) из двух полей: name – имя предиката и pref – *префикс каррирования*, определяющий набор значений, фиксированных применением оператора каррирования.

При подстановке имени программы C в качестве аргумента некоторого вызова предиката B(x˜: y) этот аргумент представляется структурным значением (C, ( )), где ( ) – обозначает пустой набор значений в качестве префикса каррирования.

Для произвольной предикатной программы H(x: y) { <оператор> } реализуется тождество

$$R(H)(x, y) \equiv \|\text{<оператор>}\|,$$

где ||<оператор>|| обозначает *операционную семантику* <оператора>.

Операционная семантика операторов Таблицы 1

$\|B(x: z); C(z: y)\| \cong \exists z.\ \|B(x: z)\| \ \&\ \|C(z: y)\|$

$\|B(x: y) \ ||\ C(x: z)\| \cong \|B(x: y)\| \ \&\ \|C(x: z)\|$

$\|\textbf{if}\ (e)\ B(x: y)\ \textbf{else}\ C(x: y)\| \cong (e \Rightarrow \|B(x: y)\|) \ \&\ (\neg e \Rightarrow \|C(x: y)\|)$

$\|B(\tilde{x}: y)\| \cong R(B)(\|\tilde{x}\|, y)$

$\|A(x: y)\| \cong \|A.name(A.pref, x, y)\|$

$\|D(y: z)\ \{B(x, y: z)\}\| \cong D = (B, (x))$

$\|D(y: z)\ \{A(x, y: z)\}\| \cong D = (A.name, (A.pref, x))$

Здесь (A.pref, x) обозначает новый префикс каррирования, полученный добавлением фиксированных значений набора x к префиксу A.pref; $\|\tilde{x}\|$ обозначает замену любого вхождения в $\tilde{x}$ имени предиката C на (C, ( )).

Поскольку ||B(x: y)|| ≅ R(B)(x, y), операционные семантики части операторов преобразуются к следующему виду:

||B(x: z); C(z: y)|| ≡ ∃z. R(B)(x, z) & R(C)(z, y)

||B(x: y) || C(x: z)|| ≡ R(B)(x, y) & R(C)(x, z)

||**if** (e) B(x: y) **else** C(x: y)|| ≡ (e ⇒ R(B)(x, y)) & (¬e ⇒ R(C)(x, y))

||A(x: y)|| ≡ R(A.name)(A.pref, x, y)

Допустим, полная программа состоит из набора предикатных программ для предикатов H_1, H_2, … , H_k. Рассмотрим систему логических тождеств:

$R(H_j)(x_j, y_j) \equiv$ ||<оператор>||, j = 1,…,k. (3)

Операционная семантика операторов в правой части тождеств выражается через $R(H_1)$, $R(H_2)$, … , $R(H_k)$, а также через операционные семантики базисных предикатов.

Семантика рекурсивных предикатов

dependent(B, C) — *непосредственная зависимость* B от C ≅
в программе предиката B имеется вызов предиката C.

def(B, C) — B *определяется через* предикат C ≅ существуют предикаты X_1, …, X_m ($m > 1$) такие, что $B = X_1$, $C = X_m$ и $dependent(X_i, X_{i+1})$ для $i = 1, …, m – 1$.

Предикат B является *рекурсивным* ≅ def(B, B)

rec(B) = {C | def(B, C) & def(C, B)} — *рекурсивное кольцо*

Лемма $C \in rec(B) \Rightarrow rec(B) = rec(C)$.

B *косвенно зависит* от C ≅ C подставляется аргументом вызова B(x˜: y) в случае, когда C зависит от B. Более экзотическая форма рекурсии может быть реализована через оператор каррирования.

! Сложные формы рекурсии запрещены

Если предикат C является аргументом вызова B(x˜: y), то B и C не могут находиться в одном рекурсивном кольце.
В рекурсивном кольце не может быть предикат, определяемый через оператор каррирования.

Пусть имеется рекурсивное кольцо предикатов H_1, H_2,…, H_n. Систему логических тождеств (3) представим в виде:

$$R(H_j)(x_j, y_j) \equiv K_j(R(H_1), R(H_2), \dots, R(H_n))(x_j, y_j),\ j = 1,\dots,n.$$

Здесь функция K_j обозначает ||<оператор>|| для программы предиката H_j, в зависимости от операционных семантик предикатов рекурсивного кольца. Будем считать, что наборы x_1, y_1, x_2, y_2, …, x_n, y_n составлены из различных переменных. Перепишем систему в векторном виде:

$$G = K(G), \qquad (4)$$

где $G = (R(H_1), R(H_2), \dots, R(H_n))$ – вектор семантик предикатов рекурсивного кольца; $K = (K_1, K_2,\dots, K_n)$ – вектор функций, применяемых к семантикам предикатов.

Допустим, система (4) определений кольца предикатов не использует сложные формы рекурсии и удовлетворяет отмеченному ограничению.

Из Лемм 9 и 10 следует, что вектор-функция K в системе уравнений $G = K(G)$ для кольца предикатов является непрерывной.

Формальная семантика условного оператора **if** (e) B(x: y) **else** C(x: y) не является непрерывной относительно вхождения переменной e . Поэтому вхождение e не может быть источником рекурсии.

В соответствии с теоремой Клини о неподвижной точке решение системы $G = K(G)$ есть предел последовательности $\{G^m\}_{m\geq 0}$, где $G^0 = \varnothing$, $G^{m+1} = K(G^m)$, $m \geq 0$.

Данная система имеет непустое решение лишь при наличии в рекурсивном кольце предиката, определяемого через условный оператор **if** (e) B(x: y) **else** C(x: y), причем лишь один из предикатов, B или C, может принадлежать рекурсивному кольцу.

Теорема 3. Для произвольной предикатной программы H(x: y) на языке P_0 истинно тождество R(H) = H. Предполагается, что данное тождество истинно для всех базисных предикатов.

Доказательство. Допустим, программа H(x: y) принадлежит полной программе П. Обозначим через SUBS(П) набор предикатов, являющихся значениями переменных предикатного типа, встречающихся в программе П. Для набора предикатов M из П обозначим через П[M] минимальную полную программу, являющуюся частью П и содержащую программы предикатов набора M. Пусть $П_0 = П$, $П_{j+1} = П_j[SUBS(П_j)]$, j = 0, 1, 2, …. Докажем, что при некотором k программа $П_k \neq \varnothing$, а $П_{k+1} = \varnothing$. Допустим обратное, т. е. существует такое k, что $П_k \neq \varnothing$ и $П_{k+1} = П_k$…….

Теорема 4. Произвольная предикатная программа H(x: y) на языке P_0 является однозначной. Напоминаем, что требование однозначности введено для всех базисных предикатов.

Случай неоднозначных предикатов

Для оператора суперпозиции B(x: z); C(z: y) в случае неоднозначного предиката B(x: z) не гарантируется завершение оператора суперпозиции.

Для некоторого z_1 реализуется B(x: z_1) & C(z_1: y).

Исполнение B(x: z) может также завершиться другим значением z_2, т.е. истинно B(x: z_2).

Предикат C(z_2: u) не определен для z_2, т.е. будет ложным для любых результатов u. В соответствии с определением операционной семантики предикат R должен быть ложным для оператора суперпозиции на значениях (x, y), поскольку исполнение не всегда завершается.

Дополнительное условие корректности для неоднозначного предиката B: $\forall z.\ (B(x: z) \Rightarrow \exists u.\ C(z: u))$.