Теория вычислительных процессов и структур

Лекция №1. Введение в предмет

Содержание лекции

Введение в предмет I

Курс лекций посвящён изучению математических моделей и методов, позволяющих описать программу, а также процесс её выполнения, математическим языком с целью последующего всестороннего анализа, разработки средств трансляции и преобразования программ.

Зачем это нужно? Почему это важно?

Алгоритм, заложенный в программу, сама программа в виде двоичного кода, транслированного с текста на некотором языке программирования, должны удовлетворять исходным требованиям. К главным требованиям следует отнести:

1. *Корректность работы программы*. Программа должна выдавать тот и только тот конечный результат, который мы ожидаем при заданных входных данных.
2. *Производительность программы*. Хотелось бы, чтобы программа работала максимально быстро и эффективно настолько, насколько это позволяет вычислительная система, исполняющая программу в данный момент.

Введение в предмет II

Поставив такие требования перед написанием программы, возникает вопрос: **А как написать такую корректную и высокопроизводительную программу?**
Написав программу, возникает вопрос: **А как проверить, что наша программа действительно является корректной и работает максимально быстро?**
Есть несколько способов ответить на эти вопросы:

1. *Проектирование, прототипирование и моделирование*. Использовать различные CASE-средства, средства моделирования, средства прототипирования и другие инструментальные средства, основанные на языках моделирования и формальных языках в процессе разработки программы, чтобы иметь возможность проверять (ну или хотя бы следовать) исходным требованиям.

Введение в предмет III

2. *Тестирование программного обеспечения (ПО)*. Сейчас это наиболее популярный подход как в промышленном, так и в научном программировании. По разным оценкам на этап тестирования и отладки закладывается не менее 50-70% времени от всего времени на разработку и выпуск программы в эксплуатацию. Очень затратно, очень трудоёмко, но поддаётся автоматизации. Правда никогда не может дать 100% результата, несмотря на то, что существуют десятки подходов к тестированию ПО. Зачастую выявляет только факт ошибки, но никак не приближает к причине ее возникновения.О причине разработчик должен догадаться самостоятельно на основе анализ программы и своих знаний о её внутренней структуре.
3. *Верификация ПО*. Используется в единичных случаях, зачастую только в научных проектах, а также некоторых проектах, связанных с безопасностью жизнедеятельности человека. За отсутствием методов и средств не всегда можно применять в том или ином процессе разработки программ. При заданном уровне знаний и автоматизации может быть не затратно и не трудоёмко. Связано с серьёзными математическими

Введение в предмет IV

исследованиями в области теоретического программирования и теории вычислений. Дает 100% результат за счёт применения *формальных математических методов*, правильность и обоснованность применения которых опять же может (должна) быть доказано математически. Может определить не только факт наличия ошибки, но и указать на причину её возникновения.

В каждом случае мы интуитивно ожидаем наличия "волшебной кнопки", нажав на которую нам скажут: ваша программа содержит пять ошибок в двух файлах.
А как реализовать эту "волшебную кнопку"? Как подойти к программе, как к объекту исследования, чтобы понять, что и как она делает, на основании этого сделать выводы о ее корректности и производительности?
Задав этот вопрос, мы обоснованно пришли к исходной теме, которой посвящён предстоящий курс лекций: **методы и подходы построения модели программы с целью последующего анализа модели программы для обнаружения различных ей свойств (корректность, производительность и другие)**.

Введение в предмет V

Отметим, что в данном курсе лекций упор делается на изучение моделей программ, ориентированных на представление параллелизма. Также отметим, что данное направление исследование развивается с одной стороны давно (уже более 40 лет), с другой стороны математическая сложность проблемы моделирования и верификации ПО не позволяет в этом направлении двигаться быстрыми шагами. Поэтому многие результаты пока являются исключительно теоретическими и в настоящее время не могут в полной мере применяться на практике.

Пример I

Рассмотрим программу разделения множеств, разработанную Э.Дейкстрой. Она много обсуждалась в публикациях, ее частичная корректность была доказана разными авторами, однако лишь недавно (в 1996 г.) Ю.Г. Карпов формально доказал отсутствие свойства тотальной корректности.

Задача разбиения множеств: Пусть заданы два множества натуральных чисел S и T. Сохраняя мощность множеств S и T, необходимо собрать в S наименьшие элементы множества $S \cup T$, а в T - наибольшие.

Программа называется *корректной*, если при остановке она вырабатывает правильный результат. Программа называется *тотально корректной*, если она всегда останавливается и вырабатывает правильный результат.

Последовательный алгоритм и программа очевидны: множества S и T сливаются, затем слитое множество упорядочивается и вновь разделяется на множества S' и T', удовлетворяющие условиям задачи.

Для параллельного решения задачи используется следующий алгоритм:

Пример II

1. Определяются два параллельно протекающих процесса Small и Large.
2. Процесс Small выбирает максимальный элемент в множестве S, а процесс *Large параллельно (в то же самое время)* находит минимальный элемент во множестве T.
3. Процессы *Small* и *Large* синхронизируются и обмениваются данными: наибольшее значение множества S пересылается процессом Small процессу Large для включения в множество T, а наименьшее значение множества T пересылается процессом Large процессу Small для включения в множество S.
4. Далее циклически повторяются шаги 3 и 4.
5. Программа останавливается, когда наибольший элемент в множестве S окажется меньше наименьшего элемента в множестве T.

Пример III

По завершении программы все элементы множества S должны оказаться не больше любого элемента множества T, а мощности этих множеств не изменяются.
Программа состоит из двух параллельных процессов, P = [Small || Large].

Small::
$mx := max(S);$
$\alpha!mx;$
$S := S - \{mx\};$
$\beta?x;$
$S := S \cup \{x\};\ mx := max(S);$
$*[mx > x \ \longrightarrow$
$\alpha!mx;$
$S := S - mx;$
$\beta?x;$
$S := S \cup \{x\};\ mx := max(S);$
$\longrightarrow]\ stop$

Large::
$\alpha?y;$
$T := T \cup \{y\};\ mn := min(T);$
$\beta!mn;$
$T := T - \{mn\};\ mn := min(T);$
$*[mn < y \ \longrightarrow$
$\alpha?y;$
$T = T \cup \{y\};\ mn := min(T);$
$\beta!mn;$
$T := T - \{mn\}; mn := min(T);$
$\longrightarrow]\ stop$

Обозначения:

Пример IV

- $\alpha!mx$ - передача по каналу α сообщения (значение) mx;
- $\alpha?y$ - получения из канала α сообщения (значения) y;
- $*[expr \longrightarrow body \ldots]stop$ - цикл выполнения *body*, пока выполняется условие *expr*.

Пример реализации программы (выдержки) на C++.
Можно указать тестовый пример, на котором эта программа работает некорректно: $S = \{5, 10, 15, 20\}$, $T = \{17, 18, 30, 40, 60\}$.
Первое же вхождение процесса Large в цикл приводит к тупиковой ситуации, поскольку Small завершится, не входя в цикл. В частности, ручной прокруткой можно проверить, что на множествах $S = \{5, 10, 15, 20\}$, $T = \{14, 17, 18, 30, 40, 60\}$ программа работает правильно: сортирует множества и завершается после однократного прохождения циклов в обоих процессах.

Начала теории вычислений I

Понятие *Вычисление* определяется интуитивно и исходит из основ математики, связанных с такими понятиями как *Теорема* и *Доказательство*.
Появлению теории вычислений (theory of computation) способствовали несколько факторов:

1. Математики стали задумываться о том, что такое строгость рассуждений, как предмет исследования математиков.
2. В 19 веке математика занялась не числовыми объектами — теория множеств.
3. Математики задумались над вопросом! что значит вычислить функцию? что есть описание аргументов? что такое результат? и т.д.

Эти три группы направлений исследований привели к формированию теории алгоритмов (или теории вычислений), как самостоятельного раздела математики — информатика (или computer science, или ещё можно назвать теоретическим программированием).
Для описания алгоритмов:

Начала теории вычислений II

1. Теория вычислимых функций (Клинни, Черч, Гедель) 1930-1936 г.
2. Абстрактные машины и модели (машина Тьюринга, машина Поста, графовые модели)
3. Комбинаторные модели (на основе операторов работы над текстовым представлением данных, алгебраические модели).

Мы будем заниматься в основном вторым и третьим разделами. В предмете будем изучать следующие модели:

- некоторые вопросы теории схем программ;
- теория сетей Петри
- теория последовательных взаимодействующих процессов (Хоар);
- некоторые вопросы верификации: модель Крипке, темпоральная логика и др. (Кларк).

Теория множеств I

Мы будем использовать стандартную нотацию теории множеств. Пусть A и B – некоторые множества.

- Равенство, объединение, пересечение, вычитание и включение этих множеств будет обозначаться как $A = B$, $A \cup B$, $A \cap B$, $A \setminus B$, $A \subset B$, соответственно.
- Принадлежность (не принадлежность) элемента a множеству A будет обозначаться как $a \in A$ ($a \notin A$).
- Символом $\emptyset$ обозначается пустое множество, т.е. множество, не содержащее ни одного элемента.
- Также выделяется множество всех натуральных чисел, которое будет обозначаться как $\mathcal{N} = \{1, 2, 3, \ldots\}$, и множество всех целых неотрицательных чисел – $\mathcal{N}_0 = \mathcal{N} \cup \{0\}$.
- Запись вида

  $$\{a \in A \mid a \text{ удовлетворяет определённому условию}\}$$

  задает подмножество множества A, состоящее из тех элементов A, которые удовлетворяют указанному условию.

Теория множеств II

- Множество A есть подмножество множества B ($A \subseteq B$), если каждый элемент множества A есть элемент B
- Множество всех подмножеств множества A будет обозначаться как $\mathcal{P}(A)$. Будем также писать $\mathcal{P}^2(A) = \mathcal{P}(\mathcal{P}(A))$ и $\mathcal{P}^+ = \mathcal{P}(A) \setminus \emptyset$.
- Пусть $\rho \subseteq \mathcal{P}(A)$. Объединением ρ называется множество $\|\rho\| = \bigcup_{x \in \rho} x$.

Отношения и функции I

Декартовым произведением двух множеств A и B называется множество $A \times B = \{(a, b) \mid a \in A, b \in B\}$.

- Подмножество декартового произведения $\rho \subseteq A \times B$ называется (бинарным) отношением на A и B.
- Для бинарного отношения ρ определяются $Dom(\rho) = \{a \mid (a, b) \in \rho\}$ (*domain*) и $Cod(\rho) = \{b \mid (a, b) \in \rho\}$ (*codomain*).
- Функцией f из A в B, обозначается как $f : A \to B$, называется отношение $f \subseteq A \times B$ такое, что для любого $a \in A$ существует единственный элемент $(a, b) \in f$. Далее будем писать $b = f(a)$.
- Пусть $A' \subseteq A$. Тогда определим два типа *проекций функции f* на A' как следующие функции: $f\lceil A' = \{(a, b) \in f \mid a \in A'\}$ и $f\lfloor A' = \{(a, b) \in f \mid a \notin A'\}$.

Наборы I

Набором называется конечное упорядоченное множество элементов, записывающееся с помощью угловых скобок, например $\langle x_1, ..., x_n \rangle$, где элементами x_i, в свою очередь, могут быть множества, наборы, мультимножества и др.
Проекцией набора называется функция $pr_i : X_1 \times ... \times X_n \to X_i$ такая, что $pr_i(\langle x_1, ..., x_i, ..., x_n \rangle) = x_i$.
Запись вида $\alpha = \langle x_1, ..., x_n \rangle$ будет обозначать "именованный" набор, т.е. набор, у которого есть имя α. Именованный набор $\alpha = \langle x_1, ..., x_n \rangle$ можно строго представить как набор $\langle \alpha, x_1, ..., x_n \rangle$, где первая позиция отведена под имена наборов. Так наборы $\alpha = \langle 1, 2 \rangle$ и $\beta = \langle 1, 2 \rangle$ различны, т.к. обозначают различные наборы $\langle \alpha, 1, 2 \rangle$ и $\langle \beta, 1, 2 \rangle$. Поэтому далее записи вида $\alpha = \langle x \rangle$ не должна вызывать изумления, т.к. просто означает набор из двух элементов $\langle \alpha, x \rangle$.
Элементы набора $\alpha = \langle A, B, C, ... \rangle$ часто будут обозначаться как A_α, B_α, $C_\alpha, \ldots$

Мультимножества I

Пусть $A = \{a_1, a_2, ..., a_n\}$ некоторое множество[1]. Мультимножеством на множестве A называют функцию $\mu : A \to \mathcal{N}_0$, которая каждому элементу из A сопоставляет неотрицательное целое число.
Мультимножество как обычную функцию можно задавать в виде множества пар

$$\{(a_1, n_1), (a_2, n_2), ..., (a_k, n_k)\}.$$

Однако, часто их удобнее записывать в виде формальной суммы: $\mu = n_1 a_1 + n_2 a_2 + ... + n_k a_k$ или $\mu = \sum_i n_i a_i$, где $n_i = \mu(a_i)$ – число экземпляров элемента a_i в мультимножестве (кратность). Далее, в зависимости от контекста, будут использоваться обе эти формы записи мультимножеств. Если $n_i = 0$, то этот член в формальной сумме будет опускаться.
Пусть $\mu_1 = n_1 a_1 + ... + n_k a_k$ и $\mu_2 = m_1 a_1 + ... + m_k a_k$ два мультимножества, определённые на множестве A.

- Будем писать $\mu_1 = \mu_2$, если $n_i = m_i$ для всех $0 \leq i \leq k$; $\mu_1 \leq \mu_2$, если $n_i \leq m_i$ для всех $0 \leq i \leq k$; $\mu_1 < \mu_2$, если $\mu_1 \leq \mu_2$, $\mu_1 \neq \mu_2$.

Мультимножества II

- Сумма и разность мультимножеств μ_1 и μ_2 определяются соответственно как

$$(\mu_1 + \mu_2)(a) = \mu_1(a) + \mu_2(a),$$

$$(\mu_1 - \mu_2)(a) = \begin{cases} \mu_1(a) - \mu_2(a), & \text{если } \mu_2(a) \le \mu_1(a); \\ 0, & \text{в противном случае.} \end{cases}$$

- Декартовым произведением двух мультимножеств μ_1 и μ_2 называется мультимножество

$$\mu_1 \times_\mu \mu_2 = \left\{ \left((a_i, a_j), n_i \cdot m_j\right) \;\middle|\; (a_i, n_i) \in \mu_1, (a_j, m_j) \in \mu_2 \right\}$$

- Из определения суммы и произведения мультимножеств нетрудно вывести следующие свойства:

$$\mu_1 + \mu_2 = \mu_2 + \mu_1$$

$$(\mu_1 + (\mu_2 + \mu_3)) = ((\mu_1 + \mu_2) + \mu_3)$$

$$\mu_1 \times_\mu (\mu_2 + \mu_3) = \mu_1 \times_\mu \mu_2 + \mu_1 \times_\mu \mu_3$$

Мультимножества III

- Если $n_i = 0$ для всех $0 \leq i \leq k$, то такое мультимножество называется *пустым мультимножеством* и обозначается как $\mathbf{0}$.
- Множество всех мультимножеств, определённых на множестве A будет обозначаться как $\mu(A)$ или, когда это не приведёт к недоразумениям, как μA. Определим также множество всех непустых мультимножеств как $\mu^+(A) = \mu(A) \setminus \{\mathbf{0}\}$.
- Будем обозначать множество, элементы которого используются в мультимножестве μ как $\|\mu\| = \{a \in A \mid \mu(a) > 0\}$.
- Будем писать $a \in \mu$ если $\exists n \geq 1 : (a, n) \in \mu$.

Так как мультимножество есть функция, то записи $\mu\lceil A$ и $\mu\lfloor A$ означают проекции μ на множество A.

---

[1]Иногда мультимножества называются *комплектами* (*bags*).

Строки I

Пусть A – некоторое множество. Тогда множество всех возможных конечных строк (слов), составленных из элементов A обозначается как A^*.
Пустая строка обозначается как ε.
Множество всех непустых строк обозначается как $A^+ = A^* \backslash \{\varepsilon\}$.
Конкатенация двух строк $v, w \in A^*$ записывается как vw.
Пусть $a \in A$ и $v = a_1 a_2 ... a_n$. Тогда будем писать $a \in v$ если $\exists i : a_i = a$.

Основы логики I

- Высказывание — это утверждение или повествовательное предложение, о котором можно сказать, что оно истинно или ложно.
- Конъюнкция, дизъюнкция, отрицание, исключающее или, будут обозначаться $\wedge$, $\vee$, $\neg$, $\veebar$ соответственно.
- Условное высказывание (импликация) обознается соответственно $p \to q$.
- Эквивалентность высказываний будет обозначаться $p \wedge p \equiv p$.

Графы I

- Граф есть конечное множество V, называемое *множеством вершин*, и подмножество $E \subseteq V \times V$, называемое *множеством рёбер*.Граф обозначается $G(V, E)$. Элементы a и b множества V называются *связанными* ребром, если $(a, b) \in E$. Ребро (a, b) называют также *инцидентным* к вершинам a и b. Также две вершины называют *смежными*, если они соединены ребром.
- *Степенью* вершины $deg(v)$ называется количество ребер, инцидентных этой вершине. Если $deg(v) = 0$, то v — изолированная вершина.
- Граф $G' = \langle V', E' \rangle$ называется *подграфом* графа $G = \langle V, E \rangle$ обозначается $G' = \langle V', E' \rangle \subseteq G = \langle V, E \rangle$, если $V' \subseteq V$ и $E' \subseteq E$.
- Пусть $G = \langle V, E \rangle$ — граф с вершинами $v_0, v_1, v_2, \ldots, v_k \in V$ и рёбрами $e_1, e_2, e_3, \ldots, e_k \in E$. *Путём длины* k из v_0 в v_k называется последовательность $v_0 e_1 v_1 e_2 v_2 e_3 v_3 \ldots e_k v_k$ такая, что $e_i = (v_{i-1}, v_i)$. *Простым путём* из v_0 в v_k называется путь, в котором нет повторяющихся вершин. Граф G называется *связным*, если имеется путь между любыми двумя его различными вершинами.

Графы II

- Пусть $G = \langle V, E \rangle$ — граф. *Циклом* называется путь ненулевой длины, соединяющий вершину саму с собой и не содержащий повторяющихся рёбер.
- Граф называется *полным*, если любые две его вершины соединены ребром.
- Граф $G = \langle V, E \rangle$ называется двудольным, если множество его вершин V можно разбить на две части $V = A \cup B$, и каждое ребро соединяет вершины только из разных подмножеств $e = (a, b) \in E$ или $e = (b, a) \in E$, где $a \in A$ и $b \in B$.