

osn 1. Предел и непрерывность функций одной и нескольких переменных. Свойства функций непрерывных на отрезке.

osn 2. Производная и дифференциал функций одной и нескольких переменных. Достаточные условия дифференцируемости.

osn 3. Определенный интеграл, его свойства. Основная формула интегрального исчисления.

osn 4. Числовые ряды. Абсолютная и условная сходимость. Признаки сходимости: Даламбера, интегральный, Лейбница.

osn 5. Функциональные ряды. Равномерная сходимость. Признак Вейерштрасса. Непрерывность суммы равномерно сходящегося ряда непрерывных функций.

osn 6. Криволинейный интеграл, формула Грина.

osn 7. Производная функции комплексного переменного. Условия Коши-Римана. Аналитическая функция.

osn 8. Степенные ряды в действительной и комплексной области. Радиус сходимости.

osn 9. Ряд Фурье по ортогональной системе функций. Неравенство Бесселя, равнство Парсеваля, сходимость ряда Фурье.

osn 10. Прямая и плоскость, их уравнения. Взаимное расположение прямой и плоскости, основные задачи на прямую и плоскость.

osn 11. Алгебраические линии и поверхности второго порядка, канонические уравнения, классификация.

osn 12. Системы линейных алгебраических уравнений. Теорема Кронекера-Капелли. Общее решение системы линейных алгебраических уравнений.

osn 13. Линейный оператор в конечномерном пространстве, его матрица. Норма линейного оператора.

osn 14. Ортогональные преобразования евклидова пространства. Ортогональные матрицы и их свойства.

osn 15. Характеристический многочлен линейного оператора. Собственные числа и собственные векторы.

osn 16. Формализация понятия алгоритма. Машины Тьюринга, нормальные алгоритмы Маркова. Алгоритмическая неразрешимость. Задача останова. Задача самоприменимости.

osn 17. Понятие архитектуры ЭВМ. Принципы фон Неймана. Компоненты компьютера: процессор, оперативная память, внешние устройства. Аппарат прерываний.

osn 18. Операционные системы. Процессы, взаимодействие процессов, разделяемые ресурсы, синхронизация взаимодействующих процессов, взаимное исключение. Программирование взаимодействующих процессов с использованием средств ОС UNIX (сигналы, неименованные каналы, IPC).

osn 19. Системы программирования. Основные компоненты систем программирования, схема их функционирования. Общая схема работы компилятора. Основные методы, используемые при построении компиляторов.

osn 20. Основные принципы объектно-ориентированного программирования. Реализация этих принципов в языке C++. Примеры.

osn 21. Базы данных. Основные понятия реляционной модели данных. Реляционная алгебра. Средства языка запросов SQL.

osn 22. Виды параллельной обработки данных, их особенности. Компьютеры с общей распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.

osn 23. Основные методы обработки изображений: тональная коррекция, свертка изображений, выделение краев.

osn 24. Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Вронского.

osn 25. Теоремы существования и единственности решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной.

osn 26. Функции алгебры логики. Реализация их формулами. Совершенная дизъюнктивная нормальная форма.

osn 27. Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шенюна.

osn 28. Вероятностное пространство. Случайные величины. Закон больших чисел в форме Чебышева.

osn 29. Квадратурные формулы прямоугольников, трапеций и парабол.

osn 30. Методы Ньютона и секущих для решения нелинейных уравнений.

osn 31. Численное решение задачи Коши для обыкновенных дифференциальных уравнений. Примеры методов Рунге-Кutta.

osn 32. Задача Коши для уравнения колебания струны. Формула Даламбера.

osn 33. Постановка краевых задач для уравнения теплопроводности. Метод разделения переменных для решения первой краевой задачи.



Я равномерно схожусь к нежеланию ботать.

ВНИМАНИЕ!
спасибо за внимание

GOOSi
Материалы для ГОСов. Кря.

LaTeX-исходники этого материала вы можете найти здесь: <https://github.com/TheFieryLynx/GOOSi>

Мальчик, водочки нам принеси. Мы МГУ закончили.

Финальный период Второй мировой войны Гитлер провел в бункере, всячески оттягивая свой конец...

«Путин говорил, что попадет к нам», — Рай подал заявку на вступление в НАТО.

На первом свидании

Она: Мне нравятся Битлз
Он, пытаясь ее впечатлить: Офицант, принесите нам два килограмма говна, пожалуйста

— Синус, косинус, тангенс, котанганс, кунилингус — найдите лишнее слово.

— Ну... не знаю, тут четыре лишних...

Стокгольм признал безуспешными все попытки отозвать из России свой синдром.

В дверь постучали 8 раз.

— Осьминог — догадался Штирлиц

— Догадался — догадался осьминог.

Китайские астрономы обнаружили, что в России уже 22 года продолжается год Крысы.

Маленький одиночка встал не с той ноги и упал

Едут батя с сыном на шестерке, перевернулись — едут на девятке

В дверь постучали 1024 раза.

Гигабайт — подумал Штирлиц.

Долбаб — ловко парировали 128 осьминогов.

Монеточка парни перед сексом:

— Выбирай, орел или решка?

Однажды в студеную зимнюю пору лошадка пипицкой примерзла к забору.

В дверь постучали 64 раза

— Восемь осьминогов, с улыбкой сказал уже подготовленный Штирлиц

— Не догадался — За дверью весело улыбались сороконожка и три осьминога

Заходит в древнем риме мужик в бар, поднимает два пальца и говорит:

— Мне пять кружек пива, пожалуйста.

Штирлиц и Мюллер ездили по очереди на танке. Очередь редела, но не расходилась...

ОН переменуется в Организацию Обеспокоенных Наций.

— А вот когда умирает черепашка, проносится ли у нее жизнь перед глазами или тиха так супремедленно проплывает?

— Я имею в виду, к свидетелю.

— К свидетелю вопросов нет, ваша честь.

Что общего между клитором и КГБ?

Одно неловкое движение языком и ты в жопе

В дверь постучали, в дверь постучали...

— Ддос-атака — хотел было подумать Штирлиц, но залагал.

Иностранный журналист спрашивает у Путина: «Господин президент, за что посадили Алексея Навального?»

— За решетку.

От работы портовой шлюхой ее останавливали только то, что в городе не было порта.

А спонсор этого дня — батюшка на батуте, вывалившись перед этим два литра пива.

Батюшка на батуте, вывалившись перед этим два литра пива — поп-рыгун

В дверь постучали 256 раз

— 32 осьминога — подумал Штирлиц

— Заебал впусти — кричал Мюллер

Песков опроверг информацию о раке у Путина, заявив, что у него краб.

Okko откроет российский аналог Pornhub «Чпокко».

В дверь вежливо постучали ногой.

— Безруков! — догадался Штирлиц.

Минкульт: в честь 9 мая будет издан ремейк знаменитой военной поэмы: «Насилий Мародеркин».

Спрашивают у бывшей проститутки: «Как вам удалось стать миллионеркой?»

— Я всегда с собой беру ви-де-о-ка-ме-ру!!!

Встречаются два мужика в пустыне. Один говорит: «Что, гололед, да?». Второй отвечает: «Нет, с чего ты взял?». Первый ему и говорит: «А нахрена столько песка насыпали?»

Накануне голосования прокуратура ещё раз напоминает, что вменительство граждан в выборный процесс в России недопустимо.

Аnekdot от Никитина:
Грин работал у отца на ферме, а когда отец умер — занялся математикой и спился. Можете рассказать это в 6 билете.

Олег перед сексом тщательно помылся, причем так тщательно, что вроде секс как уже и не нужен.

Россия объявила о победе в конкурсе «Европенавидение».

Чтобы не перепутать, бабушка назвала одного новорожденного котенка Барсик, а второго утонила.

Деду время, а потехе я посвятил жизнь

В Кремле открылся «Бункер Кинг».

А чего вы удивляетесь, что нефть стоит дешевле воды? Вы вообще нефть пробовали? Её же пить невозможно!

Мюллер выглянул в окно. По улице шел Штирлиц, ведя на поводке крохотную, зеленую с оранжевыми полосками, шестиногую собачонку.

«Странно, — подумал Мюллер, — этого анекдота я еще не знало»

По разные стороны Москвы — XXI век

дор 1. Теорема Поста о полноте систем функций в алгебре логики.
дор 2. Графы, деревья, планарные графы; их свойства. Оценка числа деревьев.
дор 3. Логика 1-го порядка. Выполнимость и общезначимость. Общая схема метода резолюций.
дор 4. Логическое программирование. Декларативная семантика и операционная семантика: соотношение между ними. Стандартная стратегия выполнения логических программ.
дор 5. Сортировка. Простейшие алгоритмы – сортировка выбором, вставками, обменом. Оценка сложности алгоритмов сортировки. Быстрая сортировка и ее сложность в среднем и в наихудшем случаях.
дор 6. Язык ассемблера как машиннозависимый язык низкого уровня. Организация ассемблерной программы, секции кода и данных (на примере ассемблера пасм или пасм). Основные этапы подготовки к скомпилированию ассемблерной программы: трансляция, редактирование внешних связей (компоновка), загрузка.
дор 7. Операционные системы. Управление оперативной памятью в вычислительной системе. Алгоритмы и методы организации и управления страницей оперативной памятью.
дор 8. Зависимости в реляционных отношениях: функциональные, многофункциональные, проекции/соединения. Проектирование реляционных БД на основе принципов нормализации отношений. Нормальные формы.
дор 9. Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задач на параллельных вычислительных системах.
дор 10. Глобальные и локальные модели освещения в компьютерной графике. Модель Фонга.
дор 11. Классификация языков, определяемых конечными автоматами, регулярными выражениями и правoliniевыми грамматиками. Эквивалентность и минимизация конечных автоматов.
дор 12. Функции FIRST и FOLLOW. LL(1)-грамматики. Конструирование таблиц предсказывающего анализа.
дор 13. Жизненный цикл программного обеспечения (ПО). Основные виды деятельности при разработке ПО. Каскадная и итерационная модели жизненного цикла.
дор 14. Качество программного обеспечения и методы его контроля. Тестирование и другие методы верификации.
дор 15. Основные понятия криптографии. Односторонняя функция с секретом. Протокол Диффи-Хеллмана выработки общего секретного ключа по открытому каналу связи.
дор 16. Основные принципы построения и архитектура сети Интернет. Алгоритмы и протоколы внешней и внутренней маршрутизации. Явление перегрузки и методы борьбы с ней.
дор 17. Теоретические основы передачи данных, физический уровень стека протоколов. Системы передачи данных Ethernet и Wi-Fi: алгоритмы работы, управление множественным доступом к каналу.
дор 18. Базисные типы данных в языках программирования. Основные проблемы, связанные с базисными типами и способы их решения в различных языках. Понятие абстрактного типа данных и способы его реализации в современных языках программирования.
дор 19. Понятие о парадигме программирования. Основные парадигмы программирования. Языки и парадигмы программирования.
дор 20. Основные характеристики функциональных языков программирования. Использование понятий функционального программирования (замыкания, анонимные функции) в современных объектно-ориентированных языках.
дор 21. Синхронизация в распределенных системах. Синхронизация времени. Логические часы. Выборы координатора. Взаимное исключение. Координация процессов.
дор 22. Отказоустойчивость в распределенных системах. Типы отказов. Фиксация контрольных точек и восстановление после отказа. Репликация и протоколы голосования. Надежная групповая рассылка.
дор 23. Распределенные файловые системы. Доступ к директориям и файлам. Семантика одновременного доступа к одному файлу нескольких процессов. Кэширование и размножение файлов.
дор 24. Промежуточные представления программы: абстрактное синтаксическое дерево; последовательность трехадресных инструкций. Базовые блоки и граф потока управления.
дор 25. Локальная оптимизация при компиляции программы. Ориентированный ациклический граф и метод нумерации значений.
дор 26. Глобальная оптимизация при компиляции программы. Построение передаточных функций базовых блоков. Монотонные и дистрибутивные передаточные функции. Метод неподвижной точки и его применение для нахождения достигающих определений.
дор 27. Постановка задачи дискретной оптимизации. Метод ветвей и границ. Задача целочисленного линейного программирования.
дор 28. Комбинаторные методы нахождения оптимального пути в графе.
дор 29. Потоки в сетях. Алгоритм построения максимального потока. Оценка сложности алгоритма.

0.0.1 OSN 1 Предел и непрерывность функций одной и нескольких переменных. Свойства функций непрерывных на отрезке.

Множество всех упорядоченных совокупностей (x_1, \dots, x_m) из чисел x_1, \dots, x_m называется **m -мерным координатным пространством** A_m .

Имеется некоторое множество M и некоторая функция $\rho: M \times M \rightarrow \mathbb{R}^+$. Функция ρ называется **метрикой** (расстоянием), а пара (M, ρ) – **метрическим пространством**, если $\forall x, y, z \in M$ выполнено:

$$1. \rho(x, y) > 0 \text{ и } \rho(x, y) = 0 \Leftrightarrow x = y$$

$$2. \rho(x, y) = \rho(y, x) \text{ (симметричность)}$$

$$3. \rho(x, y) \leq \rho(x, z) + \rho(z, y) \text{ (неравенство треугольника)}$$

Если каждой точке M из $\{M\}$ точек E_m ставится в соответствие по известному закону некоторое число u , то говорят, что на множестве $\{M\}$ задана функция $u = f(M)$. **М-область определения функции** $u = f(M)$. Число u , соответствующее данной M из $\{M\}$ – значение функции в M . Совокупность $\{u\}$ всех частных значений $u = f(M)$ – множество значений этой функции.

Предел по Гейне. Число $b \in R$ называется **пределным значением функции** $u = f(M)$ в точке $A \in R^m$ (при $M \rightarrow A$), если для \forall сходящейся к A последовательности M_1, \dots, M_n, \dots точек множества M , где $M_n \neq A$, соответствующая последовательность $f(M_1), \dots, f(M_n), \dots$ значений функций сходится к b .

Предел по Коши. Число $b \in R$ называется **пределным значением функции** $u = f(M)$ в точке $A = (a_1, \dots, a_m)$, если $\forall \varepsilon > 0 \exists \delta: \forall M \in \{M\}, \text{ удовлетворяющих } 0 < \rho(M, A) < \delta, \text{ выполняется } |f(M) - b| < \varepsilon$.

Теорема об эквивалентности определений предела. Определение предела функции по Коши и по Гейне эквивалентны.

▲ (\Rightarrow) $\exists b$ – предел $u = f(M)$ в т. A по Гейне, но опр. по Коши не выполнено $\Rightarrow \exists \varepsilon > 0: \forall \delta > 0 \exists M \in \{M\}: 0 < \rho(M, A) < \delta, |f(M) - b| \geq \varepsilon \Rightarrow \text{для } \delta_n = \frac{1}{n} \exists M_n : 0 < \rho(M_n, A) < \delta_n, |f(M_n) - b| \geq \varepsilon \Rightarrow \{M_n\} \rightarrow A \Rightarrow \text{по Гейне } \{f(M_n)\} \rightarrow b \Rightarrow \text{противоречие с } |f(M_n) - b| \geq \varepsilon$.

(\Leftarrow) \Rightarrow $\exists b$ – предел $u = f(M)$ в т. A по Коши и $\{M_n\} \rightarrow A$. Фиксируем $\varepsilon > 0$, то Коши $\exists \delta > 0: \forall M \in \{M\}: 0 < \rho(M, A) < \delta, |f(M) - b| < \varepsilon$. Т.к. $\{M_n\} \rightarrow A$, то для этого $\exists N \in \mathbb{N}: \forall n \geq N, 0 < \rho(M_n, A) < \delta \Rightarrow |f(M_n) - b| < \varepsilon \Rightarrow \{f(M_n)\} \rightarrow b$ ■

Последовательность M_1, \dots, M_n называется **фундаментальной**, если $\forall \varepsilon > 0 \exists N = N(\varepsilon) \in \mathbb{N}: \forall m \geq N, p \in \mathbb{N}$ выполнено $\rho(M_{m+p}, M_p) < \varepsilon$.

Критерий Коши сходимости последовательности: последовательность M_1, \dots, M_n сходится \Leftrightarrow последовательность фундаментальная.

Функция $f(M)$ удовлетворяет в точке M условию Коши, если $\forall \varepsilon > 0 \exists \delta: \forall M', M'' \in U(\delta)$, удовлетворяющих $0 < \rho(M', M) < \delta, 0 < \rho(M'', M) < \delta$, следует $|f(M') - f(M'')| < \varepsilon$

Критерий Коши 3 предела ф-ции. Чтобы функция $f(x)$ имела коначное предельное значение в точке a , необходимо и достаточно, чтобы функция $f(a)$ удовлетворяла в этой точке условию Коши.

▲ (\Rightarrow) $\lim_{M \rightarrow A} f(M) = b$. Выберем $\varepsilon > 0 \Rightarrow$ по опр. предела по Коши для $\exists \delta > 0, \forall M \in \{M\}: 0 < \rho(M', A) < \delta, 0 < \rho(M'', A) < \delta \Rightarrow |f(M') - b| < \frac{\varepsilon}{2}, |f(M'') - b| < \frac{\varepsilon}{2}$. Тогда $|f(M') - f(M'')| = |(f(M') - b) - (f(M'') - b)| \leq |f(M') - b| + |f(M') - b| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} < \varepsilon$.

(\Leftarrow) $\square f(M)$ удовл. в т. A усл. Коши, $\{M_n\} : \{M_n\} \rightarrow A, M_n \neq A$. Выберем $\varepsilon > 0$ соотв. $\delta > 0$ такое, что выполнены усл. Коши, для этого $\exists N \in \mathbb{N}: \forall n \geq N \Rightarrow 0 < \rho(M_n, A) < \delta$ (т.к. $\{M_n\} \rightarrow A$). Таким образом для $p = 1, 2, \dots \Rightarrow 0 < \rho(M_{n+p}, A) < \delta$ при $n \geq N \Rightarrow$ в силу усл. Коши $|f(M_{n+p}) - f(M_n)| < \varepsilon \Rightarrow \{f(M_n)\}$ – фундаментальная $\Rightarrow \{f(M_n)\}$ сходит к некоторому b .

А для $A, \{M_n\} \rightarrow A, \{f(M_n)\} \rightarrow b, \{f(M'_n)\} \rightarrow b'$. Тогда $f(M_1), f(M'_1), \dots, f(M_n), f(M'_n), \dots$ сходятся \Rightarrow все сёд подпосл-ти сходятся к одному пределу $\Rightarrow b = b'$ ■

Функция $f(x)$ называется **непрерывной в т. a** , если $\lim_{x \rightarrow a} f(x) = f(a)$ (функция должна быть задана в т. a). Для функции нескольких переменных можно определить непрерывность по каждой из переменных.

Теорема об арифметических операциях над непрерывными функциями. $\square f(M)$ и $g(M)$ непрерывны в т. A . Тогда $f(M) + g(M), f(M) - g(M), f(M)g(M), f(M)/g(M)$ (последнее при условии $g(M) \neq 0$) непрерывны в т. A .

\square функции $x_1 = \phi_1(t_1, \dots, t_k), \dots, x_m = \phi_m(t_1, \dots, t_k)$ заданы на множестве $\{N\}$ евклидова пространства E_m , t_1, \dots, t_k – координаты точек в E_k $\Rightarrow \forall N(t_1, \dots, t_k) \in \{N\}$ ставится в соответствие точка $M(t_1, \dots, t_m)$ евклидова пространства E_m . $\square \{M\}$ – множество всех этих точек $u = f(x_1, \dots, x_m)$ – функция m переменных, заданная на $\{M\} \Rightarrow$ на множестве $\{N\}$ пространства E_k определена **сложная функция** $u = f(\phi_1(t_1, \dots, t_k), \dots, \phi_m(t_1, \dots, t_k)) = f(x(t))$

Теорема о непрерывности сложной функции. \square функции $x_1 = \phi_1(t_1, \dots, t_k), \dots, x_m = \phi_m(t_1, \dots, t_k)$ непрерывны в т. $a = (a_1, \dots, a_m)$, а функция $u = f(x_1, \dots, x_m)$ непрерывна в т. $b = (b_1, \dots, b_m)$. Тогда сложная функция $f(x(t))$ непрерывна в т. a .

Свойства функций, непрерывных на отрезке (тут именно отрезок, поэтому доказываем для функции одной переменной):

Теорема о сохранении знака. $\square f(x)$ определена на мн-ве X , непрерывна в т. a и $f(a) > 0$ ($f(a) < 0$). Тогда $\exists \delta > 0: \forall x \in X, x \in (a - \delta, a + \delta) \Rightarrow f(x) > 0$ ($f(x) < 0$).

▲ $\forall \varepsilon > 0 \exists \delta(\varepsilon) > 0: \forall x \in X, 0 < |x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon$.

$\square \varepsilon = \frac{|f(a)|}{2} \Rightarrow -\varepsilon < f(x) - f(a) < \varepsilon \Rightarrow f(a) - \frac{|f(a)|}{2} < f(x) < f(a) + \frac{|f(a)|}{2}$ (тот же знак) ■

Теорема о прохождении через 0. $\square f(x)$ непрерывна на $[a, b]$, $f(a) > 0; f(b) < 0$. Тогда $\exists c \in (a, b): f(c) = 0$.

▲ $\square f(a) < 0, f(b) > 0, A = \{x \in [a, b]: f(x) < 0\}. A \neq \emptyset$ (т.к. $a \in A$) и ограничено сверху (например, числом b) $\Rightarrow \sup A = c$. Покажем, что $f(c) = 0$.

$\square f(c) > 0$. Тогда $c \neq a$ и по т. о со хр. знака $\exists \delta > 0: f(x) > 0 \forall x \in (c - \delta, c) \Rightarrow c \neq \sup A \Rightarrow$ противоречие $\Rightarrow f(c) \leq 0$.

$\square f(c) < 0$. Тогда $c \neq b$ и по т. о со хр. знака $\exists \delta > 0: f(x) < 0 \forall x \in (c, c + \delta) \Rightarrow c \neq \sup A \Rightarrow$ противоречие $\Rightarrow f(c) = 0$. ■

Теорема о достижении значения. $\square f(x)$ непрерывна на $[a, b]$, тогда $\forall \gamma \in [a, b]$, где $\alpha = \min\{f(a), f(b)\}, \beta = \max\{f(a), f(b)\}, \exists c \in [a, b]: f(c) = \gamma$.

▲ Если $\gamma = \alpha$ или $\gamma = \beta$ – очевидно. $\square \alpha < \gamma < \beta$. $\square g(x) = f(x) - \gamma$. Она удовл. усл. пред. теоремы $\Rightarrow \exists c \in [a, b]: g(c) = 0$, т.е. $f(c) = \gamma$ ■

Теорема Больцано-Вейерштрасса (пункта ниже). Из любой ограниченной последовательности $\{x_n\}$ можно выделить сходящуюся подпоследовательность.

▲ $\square \{X\}$ – мн-во значений последовательности $\{x_n\}$. Если $\{X\}$ – конечно, то найдется подмн-сть такая, что $x_1 = x_2 = x_3 = \dots$. Если $\{X\}$ – бесконечно, то по принципу Больцано-Вейерштрасса (у любого отр. беск. мн-ва есть хотя бы 1 предельная точка) у $\{X\}$ есть предельная точка \Rightarrow сходящаяся к этой точке подпосл-ть. ■

1-я теорема Вейерштрасса. Если $f(x)$ непрерывна на сегменте $[a, b]$, то она ограничена на нём.

▲ Выберем $\{x_n\}: x_n \in [a, b], |f(x_n)| > n$. По теореме Б-В можно выделить сход. подпосл-ть $\{x_{k_n}\}$, предел с которой в $[a, b]$. Очевидно, что посл-ть $\{f(x_{k_n})\}$ беск. большая, но в силу непр-ти функции в т. с эта посл-ть должна сходить к $f(c) \Rightarrow$ противоречие. ■

2-я теорема Вейерштрасса. Если $f(x)$ непрерывна на сегменте $[a, b]$, то она достигает на нем своих ТВГ и ТНГ.

▲ $f(x)$ непр. на $[a, b] \Rightarrow$ она огр. на $[a, b] \Rightarrow \exists M, m -$ ТВГ и ТНГ $f(x)$ на $[a, b]$. $\square f(x) < M \forall x \in [a, b]$. Введем $g(x) = \frac{1}{M-f(x)}$.

$g(x)$ – непр. на $[a, b]$, причем знаменатель не обр. в 0 \Rightarrow огр. на $[a, b] \Rightarrow \exists A > 0: \frac{1}{M-f(x)} \leq A \forall x \in [a, b] \Rightarrow M-f(x) \geq \frac{1}{A} \Rightarrow$

$f(x) \leq M - \frac{1}{A} \forall x \in [a, b] \Rightarrow M \neq \sup f(x) \Rightarrow$ противоречие (для ТНГ аналогично) ■

Функция $f(x)$ называется **равномерно непрерывной** на множестве X , если для $\forall \varepsilon > 0 \exists \delta = \delta(\varepsilon) > 0: \forall x', x'' \in X: |x' - x''| < \delta$, выполняется $|f(x') - f(x'')| < \varepsilon$.

Теорема о равномерной непрерывности (Кантора). Непрерывная на сегменте $[a, b]$ функция равномерно непрерывна на нем.

▲ $\square f(x)$ непр. на $[a, b]$, но не р/н на нем. Тогда $\exists x'_n, x''_n \in [a, b]: |x'_n - x''_n| < \frac{1}{n} \forall n \in \mathbb{N}$, но $|f(x'_n) - f(x''_n)| \geq \varepsilon$.

$\{x'_n\}$ – огр. $\Rightarrow \exists \{x'_n\} \subset [a, b]: \exists \lim_{n \rightarrow \infty} x'_n = c. \square \{x''_n\} \subset [a, b]:$

$|x''_n - x'_n| + |x'_n - c| \Rightarrow \{x''_n\} \rightarrow c$. По определению по Гейне непрерывности в точке $\{f(x'_n)\} \rightarrow f(c), \{f(x''_n)\} \rightarrow f(c)$ – противоречие с $|f(x'_n) - f(x''_n)| \geq \varepsilon$. ■

0.0.2 OSN 3 Определенный интеграл, его свойства. Основная формула интегрального исчисления.

Определение: $\square f(x)$ задана на $[a, b]$, $a < b$, T – разбиение $[a, b]: a = x_0 < x_1 < \dots < x_n = b$ на n частичных сегментов $[x_0, x_1], \dots, [x_{n-1}, x_n]$. $\square \xi_i$ – любая точка $[x_{i-1}, x_i], \Delta x_i = x_i - x_{i-1}$ – длина сегмента. $\Delta = \max(\Delta x_i)$.

Число $I(x_i, \xi_i)$, где $I(x_i, \xi_i) = f(\xi_i)\Delta x_1 + f(\xi_2)\Delta x_2 + \dots + f(\xi_n)\Delta x_n = \sum_{i=1}^n f(\xi_i)\Delta x_i$ называется **интегральной суммой** $f(x)$, соответствующей данному разбиению T сегмента $[a, b]$ и данному выбору промежуточных точек ξ_i на частичных сегментах $[x_{i-1}, x_i]$.

Число I называется **пределом интегральных сумм** $I(x_i, \xi_i)$ при $\Delta \rightarrow 0$, если для $\forall \varepsilon > 0 \exists \delta = \delta(\varepsilon)$: для \forall разбиения T сегмента $[a, b]$, для которого $\Delta = \max \Delta x_i < \delta$, независимо от выбора точек ξ_i на $[x_{i-1}, x_i]$ выполняется неравенство $|I(x_i, \xi_i) - I| < \varepsilon$.

$$I = \lim_{\Delta \rightarrow 0} I(x_i, \xi_i)$$

Функция называется **интегрируемой (по Риману)** на $[a, b]$, если \exists конечный предел I интегральных сумм $f(x)$ при $\Delta \rightarrow 0$. Предел I – определённый интеграл от $f(x)$ по $[a, b]$: $I = \int_a^b f(x) dx$

$\square f(x)$

0.0.5 OSN 8 Степенные ряды в действительной и комплексной областях. Радиус сходимости.

Степенной ряд и область его сходимости.

Степенным рядом называется функциональный ряд вида

$$a_0 + \sum_{n=1}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + \dots,$$

где $a_0, a_1, a_2, \dots, a_n, \dots$ — постоянные вещественные числа, называемые коэффициентами ряда.

Составим с помощью коэффициентов a_n ряд числовую последовательность:

$$\{ \sqrt[n]{|a_n|} \}, \quad (n = 1, 2, \dots) \quad (1)$$

Теорема Коши-Адамара.

1. Если последовательность 1 не ограничена, то степенной ряд сходится лишь при $x = 0$.

2. Если последовательность 1 ограничена и имеет верхний предел $L > 0$, то ряд абсолютно сходится для значений x , удовлетворяющих $|x| < \frac{1}{L}$, и расходится для значений x , удовлетворяющих неравенству $|x| > \frac{1}{L}$.

3. Если последовательность 1 ограничена и ее верхний предел $L = 0$, то ряд абсолютно сходится для всех значений x .

1. Пусть последовательность 1 не ограничена. Тогда при $x \neq 0$ последовательность $|x| \sqrt[n]{|a_n|}$ — $\sqrt[n]{|a_n x^n|}$ также не ограничена, т. е. у этой последовательности имеются члены со сколь угодно большими номерами n , удовлетворяющие неравенству $\sqrt[n]{|a_n x^n|} > 1$, или $|a_n x^n| > 1$. Но это означает, что для ряда (при $x \neq 0$) нарушено необходимое условие сходимости, т. е. ряд расходится при $x \neq 0$.

2. Пусть последовательность 1 ограничена и ее верхний предел $L > 0$. Докажем, что ряд абсолютно сходится при $|x| < \frac{1}{L}$, и расходится при $|x| > \frac{1}{L}$.

• Фиксируем сначала любое x , удовлетворяющее неравенству $|x| < \frac{1}{L}$. Тогда найдется $\varepsilon > 0$, такое, что $|x| < \frac{1}{L + \varepsilon}$.

В силу свойств верхнего предела все элементы $\sqrt[n]{|a_n|}$, начиная с некоторого номера n , удовлетворяют неравенству $\sqrt[n]{|a_n|} < L + \frac{\varepsilon}{2}$. Таким образом, начиная с этого номера n , справедливо неравенство $\sqrt[n]{|a_n x^n|} = |x| \sqrt[n]{|a_n|} < L + \frac{\varepsilon}{2} < 1$, т. е. ряд абсолютно сходится по признаку Коши.

• Фиксируем теперь любое x , удовлетворяющее неравенству $|x| > \frac{1}{L}$. Тогда найдется $\varepsilon > 0$ такое, что $|x| > \frac{1}{L - \varepsilon}$.

По определению верхнего предела из последовательности 1 можно выделить подпоследовательность $\{ \sqrt[n]{|a_{n_k}|} \}$, $(k = 1, 2, \dots)$, сходящуюся к L . Но это означает, что, начиная с некоторого номера k , справедливо неравенство $L - \varepsilon < \sqrt[n]{|a_{n_k}|} < L + \varepsilon$.

Таким образом, начиная с этого номера k , справедливо неравенство $\sqrt[n]{|a_n x^n|} = |x| \sqrt[n]{|a_n|} > L - \varepsilon > 1$, или $|a_n x^n| > 1$, откуда видно, что нарушено необходимое условие сходимости ряда и он расходится.

3. Пусть последовательность 1 ограничена и ее верхний предел $L = 0$. Докажем, что ряд абсолютно сходится при любом x . Фиксируем произвольное $x \neq 0$ (при $x = 0$ ряд заведомо абсолютно сходится). Поскольку верхний предел $L = 0$ и последовательность 1 не может иметь отрицательных предельных точек, число $L = 0$ является единственной предельной точкой, а следовательно, является пределом этой последовательности, т. е. последовательность является бесконечно малой. Но тогда для положительного числа $\frac{1}{2|x|}$ найдется номер, начиная с которого $\frac{1}{2|x|} < \frac{1}{2|x|}$ — $\sqrt[n]{|a_n|} < \frac{1}{2|x|} < 1$, т. е. ряд абсолютно сходится к признаку Коши.

Радиус сходимости.

Теорема. Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

$$R = \frac{1}{\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|}}$$

(в случае, когда $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$, $R = \infty$)

▲ Очевидно из предыдущей теоремы ■

Для случаев комплексного пространства:

Ряд вида $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ называется степенным рядом с центром разложения в точке z_0 , где $\{a_n\}$ — фиксированная последовательность комплексных чисел.

Теорема Коши-Адамара.

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)</math$

0.0.9 OSN 9 Ряд Фурье по ортогональной системе функций. Неравенство Бесселя, равенство Парсеваля, сходимость ряда Фурье.

Два элемента f и g евклидова пространства называются **ортогональными**, если скалярное произведение $\langle f, g \rangle = 0$. Рассмотрим в произвольном бесконечномерном евклидовом пространстве E некоторую последовательность элементов.

$$\psi_1, \psi_2, \dots, \psi_n, \dots \quad (2)$$

Последовательность (2) называется **ортонормированной системой**, если входящие в эту последовательность элементы попарно ортогональны и имеют норму, равную единице.

Пусть в произвольном бесконечномерном евклидовом пространстве E задана произвольная ортогонализованная система элементов $\{\psi_k\}$. Рассмотрим какой угодно элемент f пространства E .

Назовём **рядом Фурье** элемента f по ортогонализованной системе $\{\psi_k\}$ ряд вид:

$$\sum_{k=1}^{\infty} f_k \psi_k,$$

в котором через f_k обозначены постоянные числа, называемые **коэффициентами Фурье** элемента f и определяемые равенствами $f_k = \langle f, \psi_k \rangle$, $k = 1, 2, \dots, n$.

$S_n = \sum_{k=1}^n f_k \psi_k$ называется **n-й частичной суммой ряда Фурье**.

Рассмотрим наряду с n -й частичной суммой произвольную линейную комбинацию первых n элементов ортогонализованной системы $\{\psi_k\}$:

$$\sum_{k=1}^n C_k \psi_k$$

какими угодно постоянными числами C_1, C_2, \dots, C_n .

Величина $\|f - g\|$ называется **отклонением** f по норме данного евклидова пространства.

Задача о начальном приближении: $\min_{\forall \{C_i\} \in \mathbb{R}} \|f - \sum_{k=1}^n C_k \psi_k\|$

Будем минимизировать квадрат нормы:

$$\|f - \sum_{k=1}^n C_k \psi_k\|^2 = \left\langle f - \sum_{k=1}^n C_k \psi_k, f - \sum_{k=1}^n C_k \psi_k \right\rangle = \langle f, f \rangle -$$

$$2 \sum_{k=1}^n C_k \langle f, \psi_k \rangle + \sum_{k=1}^n C_k^2 = \|f\|^2 + \sum_{k=1}^n (C_k^2 - 2C_k f_k) = \left\{ \pm \sum_{k=1}^n f_k^2 \right\} =$$

$$\|f\|^2 - \sum_{k=1}^n f_k^2 + \sum_{k=1}^n (C_k - f_k)^2,$$

минимум достигается при $C_k = f_k$, $k = 1, 2, \dots, n$. Таким образом, доказана следующая теорема:

Теорема. Среди всевозможных линейных комбинаций элементов ортогонализованной системы $\{\psi_k\}$ евклидова пространства наименьшее отклонение от произвольного элемента f из пространства имеет n -ю частичную сумму ряда Фурье элемента f по системе $\{\psi_k\}$.

Следствие 1. \forall элемента f данного евклидова пространства, \forall ортогонализованной системы $\{\psi_k\}$ при произвольном выборе постоянных C_k и $\forall n$ справедливо неравенство

$$\|f\|^2 - \sum_{k=1}^n f_k^2 \leqslant \left\| \sum_{k=1}^n C_k \psi_k - f \right\|^2$$

▲ $\|f - \sum_{k=1}^n C_k \psi_k\|^2 = \|f\|^2 - \sum_{k=1}^n f_k^2 + \sum_{k=1}^n (C_k - f_k)^2 \geq \|f\|^2 - \sum_{k=1}^n f_k^2$ ■

Следствие 2 (тождество Бесселя). \forall элемента f данного евклидова пространства, \forall ортогонализованной системы $\{\psi_k\}$ и $\forall n$ справедливо равенство

$$\left\| \sum_{k=1}^n f_k \psi_k - f \right\|^2 = \|f\|^2 - \sum_{k=1}^n f_k^2$$

▲ Подставить $C_k = f_k$ в $\|f - \sum_{k=1}^n C_k \psi_k\|^2 = \|f\|^2 - \sum_{k=1}^n f_k^2 + \sum_{k=1}^n (C_k - f_k)^2$ то

■ Неравенство Бесселя. \forall элемента f данного евклидова пространства, \forall ортогонализованной системы $\{\psi_k\}$ выполняется неравенство Бесселя:

$$\sum_{k=1}^n f_k^2 \leqslant \|f\|^2$$

▲ Из тождества Бесселя: $\|f - \sum_{k=1}^n f_k \psi_k\|^2 \geq 0 \Rightarrow \|f\|^2 - \sum_{k=1}^n f_k^2 \geq 0$

$0 \Rightarrow \|f\|^2 \geq \sum_{k=1}^n f_k^2$ ■

Ортогонализованная система $\{\psi_k\}$ называется **замкнутой**, если \forall элемента f данного евклидова пространства E и \forall числа $\varepsilon > 0$ найдётся такая линейная комбинация конечного числа элементов $\{\psi_k\}$, отклонение которой от f (по норме пространства E) меньше ε .

Другими словами, любой элемент пространства можно с любой степенью точности приблизить по норме этого пространства линейной комбинацией конечного числа первых элементов этой системы.

Теорема. Если ортогонализованная система $\{\psi_k\}$ является замкнутой, то \forall элемента f рассматриваемого евклидова пространства неравенство Бесселя переходит в точное равенство

$$\sum_{k=1}^n f_k^2 = \|f\|^2,$$

называемое **равенством Парсеваля**.

▲ Фиксируем произвольный элемент f рассматриваемого евклидова пространства и произвольный $\varepsilon > 0$. Т.к. система f_k является замкнутой, то найдётся такой номер n и такие числа C_1, C_2, \dots, C_n , что квадрат нормы, стоящий в правой части неравенства из следствия 1, будет меньше ε . В силу следствия 1 это означает, что для произвольного $\varepsilon > 0$ найдётся номер n , для которого $\|f\|^2 - \sum_{k=1}^n f_k^2 < \varepsilon$.

$\forall m > n$ это неравенство будет тем более справедливо, так как при возрастании номера n сумма, стоящая в левой части может только возрастать. В соединении с неравенством Бесселя это означает, что ряд сходится к сумме $\|f\|^2$. ■

[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

0.0.10 OSN 11 Алгебраические линии и поверхности второго порядка, канонические уравнения, классификация.

Если в пространстве V_3 зафиксированы точка O и базис $\{e_1, e_2, e_3\}$, то говорят что в пространстве задана **аффинная система координат** (или **общая декартова система координат**) $\{O, e_1, e_2, e_3\}$. Точка O называется **началом координат**. Оси, проходящие через начало координат и определенные векторами $\{e_1, e_2, e_3\}$, называются **осами координат**. (Обозначается как O_{xyz}). Если вектора e_i взаимно перпендикулярины, то задана **прямоугольная система координат**.

Пусть Oxy – аффинная система координат на плоскости. **Алгебраическая линия второго порядка** определяется уравнением $F(x, y) = 0$, где $F(x, y)$ – алгебраический многочлен второй степени от переменных x и y с вещественными коэффициентами:

$$F(x, y) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y + a_{33} = 0, \\ a_{11}^2 + a_{22}^2 + a_{33}^2 \neq 0.$$

Это ур-е называется **общим уравнением алгебраической линии второго порядка** на плоскости. Группы слагаемых $a_{11}x^2 + 2a_{12}xy + a_{22}y^2$ называются **квадратичной частью уравнения**, группа слагаемых $2a_{13}x + 2a_{23}y$ – **линейной частью**, а a_{33} – свободным членом. Введем обозначения:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}, \quad b = \begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}, \quad X = \begin{pmatrix} x \\ y \end{pmatrix}, \\ B = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} = \begin{pmatrix} A & b \\ b^T & a_{33} \end{pmatrix}$$

Тогда уравнение примет вид:

$$F(x, y) = X^T AX + 2b^T X + a_{33} = 0, \quad A \neq A^T, \quad A \neq O.$$

Теорема. Общее уравнение линии второго порядка, заданное в прямоугольной декартовой системе координат, переходом к другой прямоугольной системе координат приводится к одному из следующих типов уравнений:

1. $\lambda_1 x^2 + \lambda_2 y^2 + a_0 = 0$, где $\lambda_1 \lambda_2 \neq 0$
2. $\lambda_2 y^2 + 2b_0 x = 0$, где $\lambda_2 b_0 \neq 0$
3. $\lambda_2 y^2 + c_0 = 0$, где $\lambda_2 \neq 0$

Эти уравнения называются **приведенными уравнениями** линии второго порядка.

▲ Шаг 1: (преобразование базиса). **Метод вращений**. Если $a_{12} \neq 0$, то поворотом осей можно привести квадратичную часть $F(x, y)$ к сумме квадратов: $F(x, y) = a'_{11}x'^2 + a'_{22}y'^2 + a'_{13}'x' + a'_{23}'y' + a_{33} = 0$.

Шаг 2: (перенос начала). Если в полученным ур-е содержится несущий квадрат какой-либо переменной, то переносом начала можно освободиться от этой переменной в первой степени. Если $a'_{11} \neq 0$ и $a'_{22} \neq 0$, то

$$x'' = x' + \frac{a'_{13}}{a'_{11}}, \quad y'' = y' + \frac{a'_{23}}{a'_{22}}, \quad a'_{33} = a_{33} - \frac{a'_{13}^2}{a'_{11}} - \frac{a'_{23}^2}{a'_{22}}$$

$$a'_{11}x''^2 + a'_{22}y''^2 + a'_{33} = 0$$

Все промежуточные и окончательные системы координат оставались прямоугольными, т.к. преобразования базиса с помощью ортогональной матрицы перехода сохраняют свойства ортогонализации. ■

Классификация линий второго порядка

Теорема. Общее уравнение линии второго порядка, заданное в прямоугольной декартовой системе координат, определяет одну и только одну из девяти линий. Для каждой из них существует прямоугольная система координат, в которой уравнение этой линии имеет **канонический вид**:

I тип:

$$1. \frac{x^2}{a^2} + \frac{y^2}{b^2} = \pm 1 \text{ – эллипс (минимый эллипс);}$$

2. $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 0$ – пара мнимых пересекающихся прямых (пара пограничных прямых); Только начало координат удовлетворяет ур-ю мним. пер. прям.

$$3. \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \text{ – гипербола;}$$

II тип: $y^2 = 2px$, $p > 0$ – парабола;

III тип:

1. $y^2 = \pm a^2$, $a \neq 0$ – пара параллельных прямых (пара мнимых параллельных прямых); Ни одна точка не удовлетворяет ур-ю мним. прям.

$$2. y^2 = 0$$
 – пара совпадающих прямых.

IV тип: $y^2 = 2px$, $p > 0$ – параболический цилиндр;

V тип:

1. $y^2 = \pm a^2$ – пара параллельных плоскостей (пара мнимых параллельных плоскостей);

$$2. y^2 = 0$$
 – пара совпадающих плоскостей.

VI тип: $y^2 = 2px$, $p > 0$ – параболический цилиндр;

VII тип:

1. $y^2 = \pm a^2$ – пара параллельных плоскостей (пара мнимых параллельных плоскостей);

$$2. y^2 = 0$$
 – пара совпадающих плоскостей.

Изложено в *Линейной алгебре и аналитической геометрии*, Г. Д. Ким, page 192-200, 329-341]

0.0.11 OSN 13 Линейный оператор в конечномерном пространстве, его матрица. Норма линейного оператора.

Полем называется множество F введенными на нем алгебраическими операциями сложения и умножения, а также если выполнены следующие аксиомы:

- Коммутативность сложения: $\forall a, b \in F : a + b = b + a$
- Ассоциативность сложения: $\forall a, b, c \in F : (a + b) + c = a + (b + c)$
- Существование нулевого элемента: $\exists 0 \in F : \forall a \in F : a + 0 = 0$
- Существование противоположного элемента: $\forall a \in F : \exists -a \in F : a + (-a) = 0$
- Коммутативность умножения: $\forall a, b \in F : a \cdot b = b \cdot a$
- Ассоциативность умножения: $\forall a, b, c \in F : (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Существование единичного элемента: $\exists E \in F : \forall a \in F : a \cdot E = a$
- Существование обратного элемента для ненулевых элементов: $(\forall a \in F : a \neq 0) \exists a^{-1} \in F : a \cdot a^{-1} = E$

• Дистрибутивность умножения относительно сложения: $\forall a, b, c \in F : a \cdot (b + c) = a \cdot b + a \cdot c</math$

0.0.13 OSN 16 Формализация понятия алгоритма. Матчины Тьюринга, нормальные алгоритмы Маркова. Алгоритмическая неразрешимость. Задача останова. Задача самоприменимости.

Интуитивное понятие алгоритма — четкая система действий, позволяющая определенным образом обработать входные данные выдать результат решения задачи. Важен исполнитель алгоритма. Одна и та же система действий для одного исполнителя будет алгоритмом, а для другого — нет.

Алгоритм применим к входным данным, если исполнитель за конечное число шагов остановится и выдаст (какой-то) ответ. В противном случае алгоритм не применим к конкретным входным данным, т.е. он не останавливается, либо завершает свой выполнение аварийно (ломается).

Основные свойства алгоритма:

1. **Определенность** (понятность). Исполнитель алгоритма абсолютно точно знает, как выполнять все шаги алгоритма.

2. **Детерминированность**. Если алгоритм применим к конкретным входным данным, то он всегда и везде выдаст одинаковый ответ, а если неприменим, то всегда и везде застывают или сломается.

3. **Дискретность или структурность**. Каждый достаточно сложный шаг алгоритма тоже является алгоритмом и может быть разложен на более простые шаги. Это же касается и обрабатываемых алгоритмом данных.

Существуют разные способы формализации понятия алгоритма, Θ два из них: машины Тьюринга и нормальные алгоритмы Маркова.

Машинна Тьюринга — гипотетическая машина (из-за использования бесконечной ленты). Автомат может двигаться вдоль ленты и поочереди обозревать содержимое ячеек. Он может находиться в одном из нескольких состояний q_1, \dots, q_k . В зависимости от того, какую букву s_i автомат видит в состоянии q_j , то есть от пары (s_i, q_j) (i — номер ячейки, j — номер состояния) автомат может выполнить следующие действия:

- запись новой буквы в обозреваемую ячейку;
- сдвиг влево или вправо на одну ячейку;
- переход в новое состояние.

Пример: перенести первый символ непустого слова Р в его конец.

| | a | b | c | A | комментарий |
|-------|-------------------|-------------------|-------------------|--------|--------------------------|
| q_1 | λ, R, q_2 | λ, R, q_3 | λ, R, q_4 | $R,$ | анализ 1 симв., удаление |
| q_2 | $, R,$ | $, R,$ | $, R,$ | $a, !$ | запись a справа |
| q_3 | $, R,$ | $, R,$ | $, R,$ | $b, !$ | запись b справа |
| q_4 | $, R,$ | $, R,$ | $, R,$ | $c, !$ | запись c справа |

Тезис Тьюринга: если кто-то предложит какой-либо алгоритм обработки слов в заданном алфавите, то можно построить эквивалентную машину Тьюринга, которая будет применима и неприменима к одинарным множествам слов. В случае машины Тьюринга **алгоритм** — это то, что может быть реализовано МТ.

Нормальный алгоритм Маркова:

Нет понятия ленты и подразумевается непосредственный доступ к любым частям преобразуемого слова. Пусть A, B — слова в некотором алфавите. Нормальный алгоритм можно записать в следующем виде:

$A \left\{ \begin{array}{l} \rightarrow \\ \mapsto \\ \mapsto \end{array} \right\} B_i$. Каждая пара — формула подстановки для замены подслов в преобразуемом слове. Идетесь вхождение слова A_1 в исходное слово. Если нашли, то заменим его на B_1 , если нет, то ищем A_2 и так далее. Затем возвращаемся в начало и снова ищем вхождение A_1 . Процесс заканчивается, если ни одна из подстановок не применима, либо применялась завершающая формула, в которой \mapsto .

Пример: $A = \{a, b\}$. Преобразовать слово P так, чтобы в его начале оказались все символы a , а в конце — все символы b .

$$\{ba \rightarrow ab\}$$

Тезис Маркова: если кто-то предложит какой-либо алгоритм обработки слов в заданном алфавите, то его можно нормализовать, т.е. построить эквивалентный нормальный алгоритм Маркова, который будет применим и неприменим к одинарным множествам слов. Машина Тьюринга и нормальные алгоритмы Маркова эквивалентны.

Самоприменимость

Входное слово, которое подаётся на вход алгоритму, может быть записью какого-то другого алгоритма. Когда алгоритм применим к своей записи, он называется **самоприменимым**.

Теорема. Если есть два алгоритма таких, что выходные данные одногожно использовать как входные данные для другого, то обязательно существует третий алгоритм, который работает как суперпозиция (композиция, последовательное выполнение) двух первых алгоритмов.

[Давалась без доказательства]

Задача останова

Пусть требуется построить алгоритм X , который, получая на входе запись любого алгоритма A и его конкретные входные данные D , определяет, применим ли A к этим данным D (остановится ли A , получив на входе D).

Теорема. Такого алгоритма X не существует. [Давалась без доказательства]

Алгоритмическая неразрешимость

Существуют задачи, для которых в принципе невозможно построить алгоритм их решения, они и называются **алгоритмически неразрешимыми**. Пусть требуется построить алгоритм X , который, получая на вход запись любого алгоритма A , определяет, самоприменим ли этот A , и нет.

Теорема. Алгоритм X не существует.

▲ Доказательство от противного. Пусть алгоритм X существует, и, получив на вход запись алгоритма A , он вырабатывает ответ DA (Да), если A самоприменим, и ответ NET (Нет), если несамоприменим. Построим вспомогательный алгоритм Y , вот его запись в форме НАМ:

$$\left\{ \begin{array}{l} DA \rightarrow DA \\ NET \mapsto NET \end{array} \right.$$

Как видно, мы специально сделали так, чтобы выходные данные алгоритма X можно подать на вход алгоритма Y . Тогда обязательно существует алгоритм Z , который работает как суперпозиция $X * Y$, то есть $Z = X * Y$. Θ самоприменим ли Z .

— Пусть Z самоприменим, тогда $\langle \text{запись } Z \rangle \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle DA \rangle Y \rightarrow \langle \text{запись } Z \rangle$. Зациклились, предположение неверно.

— Пусть Z несамоприменим, тогда $\langle \text{запись } Z \rangle \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle NET \rangle Y \rightarrow \text{Стоп}$, алгоритм самоприменим, предположение неверно.

Как видно, оба предположения неверны, поэтому делаем вывод, что алгоритм Z не существует. Однако алгоритм Y существует (мы его построили), поэтому не существует алгоритм X . ■

[В. Н. Пильщиков, *Машинна Тьюринга и алгоритмы Маркова. Решение задач. Учебно-методическое пособие.*]

0.0.14 OSN 14 Ортогональные преобразования евклидова пространства. Ортогональные матрицы и их свойства.

Базисом линейного пространства называется упорядоченная линейно независимая система векторов пространства, через которую линейно выражается любой вектор пространства.

Ортонормированный базис называется базисом, векторы которого имеют единичную длину и в случае $n > 1$ попарно перпендикуляры.

Ортогональные матрицы и их свойства.

- Матрица $Q \in \mathbb{C}^{n \times n}$ называется **унитарной**, если $Q Q^H = Q^H Q = I$
- Матрица $Q \in \mathbb{R}^{n \times n}$ называется **ортогональной**, если $Q Q^T = Q^T Q = I$
- Матрица $Q \in \mathbb{C}^{n \times n}$ называется **ортогональной**, если $Q^H = \text{эрмитова-сопряженная матрица: } q_{ij} = \bar{q}_{ij}$
- Матрица $Q \in \mathbb{R}^{n \times n}$ называется **ортогональной**, если $Q^T = \text{транспонированная матрица: } q_{ij} = q_{ji}$

Свойства ортогональной матрицы:

1. Q — обратима, причём $Q^{-1} = Q^T$; $\Delta |Q^T| = |Q| \Rightarrow |Q|^2 = 1 \Rightarrow |Q| \neq 0 \Rightarrow \exists Q^{-1}, Q^T Q = I \Rightarrow \{\text{домн. справа на } Q^{-1}\} \Rightarrow Q^T = Q^{-1}$ ■
2. $\det(Q) = \pm 1$; $\Delta |Q^T| = |Q| \Rightarrow |Q|^2 = 1 \Rightarrow |Q| = \pm 1$ ■
3. $\forall \lambda$ — собственное значение $Q \Rightarrow \lambda = \pm 1$. Δ Пусть $Qx = \lambda x$ тогда $(x, x) = (Qx, Qx) = (Qx, x) = (\lambda x, \lambda x) = \lambda^2 (x, x) \Rightarrow (x, x) = \lambda^2 (x, x) \Rightarrow \lambda^2 = 1$ ■

Линейный оператор U , действующий в унитарном (евклидовом) пространстве, называется **унитарным** (ортогональным) оператором, если $U^* U = U U^* = I$

• Оператор U унитарен (ортогонален) \Leftrightarrow в любом ортогональном базисе он имеет унитарную (ортогональную) матрицу.

• Для унитарного (ортогонального) оператора U справедливы равенства: $U^* = U^{-1}$, $|\det U| = 1$.

• Унитарный (ортогональный) оператор нормален.

Критерий унитарности. В конечномерном унитарном (евклидовом) пространстве, называется **унитарным** (ортогональным) оператором, если $U^* U = U U^* = I$

• Оператор U унитарен (ортогонален) \Leftrightarrow в любом ортогональном базисе он имеет унитарную (ортогональную) матрицу.

• Для унитарного (ортогонального) оператора U справедливы равенства: $U^* = U^{-1}$, $|\det U| = 1$.

• Унитарный (ортогональный) оператор нормален.

Критерий изометрии. В конечномерном унитарном (евклидовом) пространстве V следующие утверждения равносильны:

• Оператор U унитарен (ортогонален)

• $U^* U = I$

• $U U^* = I$

• оператор U изометричен

• оператор U сохраняет длину, т.е. $|Ux| = |x|, \forall x \in V$

• оператор U переводит любой ортогональный базис V в ортогональный базис

• оператор U переводит хотя бы один ортогональный базис V в ортогональный базис

• $U U^* = I \Rightarrow U U^* = I \Rightarrow \exists U^{-1}, U U^* = I \Rightarrow U^{-1} U U^* = I \Rightarrow U^* U = I$

• $(U \Leftrightarrow U^*)$ очевидно

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $|x| = \sqrt{(x, x)}$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $|x| = \sqrt{(x, x)} = \sqrt{|x|^2 - |x|^2 - |y|^2 + |y|^2}/2$. V — унитарное пространство

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x + y, x + y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к. $(x, y) = (x, y) - (x, x) - (y, y)/2$

• $(U \Leftrightarrow U^*)$ очевидно, т.к.

0.0.17 OSN 17 Понятие архитектуры ЭВМ. Принципы фон Неймана. Компоненты компьютера: процессор, оперативная память, внешние устройства. Аппарат прерываний.

Компьютер – исполнитель алгоритма на языке машины.
Архитектура ЭВМ – совокупность узлов машины и взаимосвязей между ними, рассматриваемая на определённом уровне рассмотрения этой архитектуры.

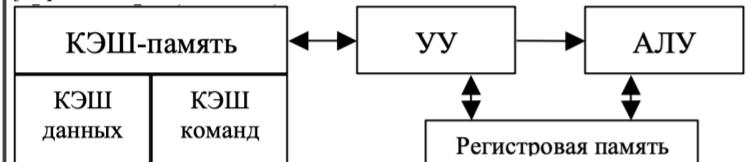
Принципы фон Неймана:

- 1. Принцип двоичного кодирования информации: вся информация, которая поступает и обрабатывается в компьютере, кодируется в двоичной системе счисления.

2. Принцип программного управления. Программа состоит из команд, в которых закодированы операции и операнды, над которыми должна выполняться данная операция. Выполнение компьютером программы – это автоматическое выполнение определенной последовательности команд, составляющих программу. В компьютере устройство, обеспечивающее выполнение команд, – Последовательность выполненных процессором команд последовательностью команд и данных, составляющих программу. То есть, по сути, второй принцип – это принцип последовательной обработки.

3. Принцип хранения программы. Для хранения команд и данных программы используется единое устройство памяти, которое представляется в виде вектора слов. Все слова имеют последовательную адресацию. Команды и данные представляются единым образом. Интерпретация информации памяти и, соответственно, ее идентификация как команды или как данных происходит неявно при выполнении очередной команды. К примеру, содержимое слова, адрес которого используется в команде перехода в качестве операнда, интерпретируется как команда. Если то же слово используется в качестве операнда команды сложения, то его содержимое интерпретируется как данные. То есть одна и та же область памяти в зависимости от команд в одном случае будет интерпретироваться как команда, в другом случае – как данные. Этот принцип фон Неймана замечателен тем, что он определяет возможность программной генерации команд с последующим их выполнением, то есть возможность компилиации программы, когда одна программа порождает другую программу, которая будет выполняться.

Процессор состоит из **устройств управления (УУ)** и **арифметико-логического устройства (АЛУ)**. АЛУ выполняет различные операции над данными, хранящимися на регистрах АЛУ. УУ выполняет команды языка машины, пославая управляемые сигналы к остальным устройствам.



Основная (оперативная) память хранит команды программы и обрабатываемые данные. ОЗУ состоит из ячеек, ячейка памяти – устройство, в котором размещается информация. Ячейка состоит из двух полей **тек** и **машинное слово**. Машинное слово – поле программно изменяемой информации. Здесь могут располагаться машинные команды или данные, с которыми будет оперировать программа. Имеет фиксированый для данной ЭВМ размер. Размер машинного слова – количество двоичных разрядов, размещаемых в машинном слове. Поле машинной информации (**тек**) – поле ячеек памяти, в котором схемами контроля процессорами и ОЗУ автоматически размещается информация, необходимая для осуществления контроля за целостностью и корректностью использования данных. Использование тега:

- Контроль за целостностью данных – одноразрядный тег, который использовался для контроля точности.
- Контроль доступа к командам/данным. (вся информация раскрашивается в 2 цвета - команды и данные)
- Контроль доступа к машинным типам данных. (в теге записывается код типа данных)

Ячейки памяти, расположенные не в основной памяти ЭВМ, а в других устройствах, называются регистрами. В процессе работы команды поступают на регистры в УУ, а данные – на регистры в АЛУ. АЛУ может обрабатывать данные только на своих регистрах, чтобы обработать данные, расположенные в основной памяти, их надо сначала считать на регистры в АЛУ.

Внешние устройства служат для обмена программами и данными между основной (оперативной) памятью и «внешним миром».

Аппарат прерываний – способность ЭВМ быстро и гибко реагировать на события, происходящие как внутри процессора и оперативной памяти, так и во внешних устройствах. Каждое такое событие порождает сигнал, приходящий на специальную электронную схему – контроллер прерываний.

Прерывания делятся на:

– **Внутренние (синхронные)**, источником которых являются выполняемые команды программы, их нельзя закрыть и не реагировать на них.

– **Внешние (асинхронные)**, которые вызываются событиями в периферийных устройствах. Эти прерывания можно временно закрыть, если в данный момент процессор занят другой срочной работой.

Аппаратная реакция на прерывание заключается в сохранении информации о считающейся в данное время программе (процесса) и переключение на выполнение другой программы (процедуры обработки прерываний, т.е. события, здесь включается режим блокировки прерываний). В некоторых архитектурах это называется переключением контекста. Этот механизм позволяет (при необходимости) продолжить (возобновить) выполнение прерванной программы с текущего места.

Программная реакция на прерывание производится процедурой обработчиком прерываний и делится на два этапа. Сначала происходит минимальная программная реакция, она производится в режиме с закрытыми прерываниями от внешних устройств. Это опасный режим, так как процессор не обращает внимание на все события в периферийных устройствах. Затем происходит полная программная реакция уже в режиме с открытыми прерываниями.



Короткое прерывание – обработка не требует дополнительных ресурсов ЦП и времени.

Фатальное прерывание – после него продолжить выполнение программы невозможно.

0.0.18 OSN 19 Системы программирования. Основные компоненты систем программирования, схема их функционирования. Общая схема работы компилятора. Основные методы, используемые при построении компиляторов.

- **Системой программирования** называется комплекс программных средств, предназначенный для поддержки программного продукта на протяжении всего жизненного цикла этого продукта.
- **Этапы жизненного цикла** программного продукта: разработка, сопровождение, эксплуатация.
- **Этапы разработки программного продукта** анализ требований, проектирование, написание текста программы ("кодирование"), трансляция, компоновка/интеграция (связывание частей программы в единую систему), верификация (процесс проверки на правильность), тестирование (обнаружение дефектов посредством сравнения с эталоном) и отладка (процесс поиска причин дефектов и их устранение), документирование, внедрение, тиражирование, сопровождение (этот этап является повторением всех предыдущих).

1. Основные компоненты системы программирования:

1. **Транслятор** переводит программы на исходном языке программирования в некоторый целевой язык. В случае, когда транслятор является компилятором, целевой язык – язык ассемблера, машинный код или байт-код некоторой виртуальной машины.
2. **Интерпретатор** выполняет программы без необходимости предварительной компиляции в машинный код. Может содержать **Just-in-Time (JIT)** компилятор, который транслирует программы в машинный код во время выполнения для оптимизации часто используемых участков кода. Если интерпретатор выполняет не исходный текст, а некоторое промежуточное представление (называемое байт-кодом), то интерпретатор называется виртуальной машиной. В этом случае для получения байт кода необходим отдельный транслайтер.
3. **Макрогенератор** или **макропроцессор** выполняет преобразование текста программы, выполняя замену вызовов макроопределений их определениями. Если макропроцессор входит в состав транслятора, его называют **Пропроцессором**. В этом случае он выполняет непосредственно транслицию кода в целевой язык.
4. **Редактор текстов** используется для написания и редактирования исходного текста программ на языке программирования.
5. **Редактор связей** или **компоновщик** используется для связывания между собой (по внешним данным) объектных модулей, посыпаляемых компилятором, а также файлов библиотек (которые являются наборами объектных модулей внутри одного файла), входящих в состав СП.
6. **Отладчик** используется для проверочных запусков программы и исправления ошибок. В нем обычно присутствуют такие возможности как интроспекция (получение типов данных) и анализ данных программы в время выполнения, остановка выполнения в определенной точке или при определенном условии, пошаговое выполнение программы и сопоставление машинного кода программы ее исходного кода при выполнении.
7. **Библиотеки стандартных программ** облегчают работу программиста, используются на этапе трансляции и исполнения.

• Дополнительные компоненты систем программирования:

1. Система контроля версий для версионирования исходного текста ПП (git).
2. Средства конфигурирования помогают создавать различные конфигурации ПП в зависимости от конкретных параметров системного окружения.
3. Система сборки позволяет автоматизировать сборку ПО (maven)
4. Средства тестирования помогают при составлении набора тестов, автоматического выполнения тестов.

5. Профилировщик используется для анализа поведения программы и поиска критических участков кода, на которые затрачивается наибольшее количество ресурсов (пример: анализ затрачиваемого на выполнение каждой функции времени, возможно в процентах от полного времени выполнения программы). Используется для оптимизации программы.
6. Справочная система содержит справочные материалы по языку программирования и компонентам СП.
7. Инструменты для статического анализа кода позволяют найти дефекты в программном коде без выполнения программы с помощью формальных методов.
8. Средства навигации по коду позволяют более эффективно ориентироваться в коде и поддерживать, например, переход от вызова функции к ее определению.
9. Инструменты подготовки документации. (Sphinx)

10. Система управления разработкой.

В другой терминологии интерпретаторы также называют трансляторами. В системе программирования должен обязательно присутствовать транслятор или интерпретатор. Также могут присутствовать оба. В этом случае они либо взаимозаменяются (Например, tiny с compiler может либо интерпретировать Си, либо компилировать), либо, если интерпретатор это виртуальная машина, должны использоваться одновременно (Например, java: java+javac).

Схема функционирования компилятора и часто применяемые алгоритмы (методы):

1. **Лексический анализ**. Лексический анализатор читает поток символов исходного текста программы и группирует эти символы в значащие последовательности – **лексемы**. Используется разбор с использованием регулярных грамматик для преобразования потока символов в поток лексем.
2. **Синтаксический анализ**. Синтаксический анализатор формирует из последовательности лексем (токенов) промежуточное представление программы (синтаксическое дерево, AST, дерево, DSA). Используются алгоритмы разбора грамматик, наиболее эффективные для данной грамматики, например, рекурсивный спуск, LL(1), LR(1) или, для отдельных частей грамматики, регулярные выражения.

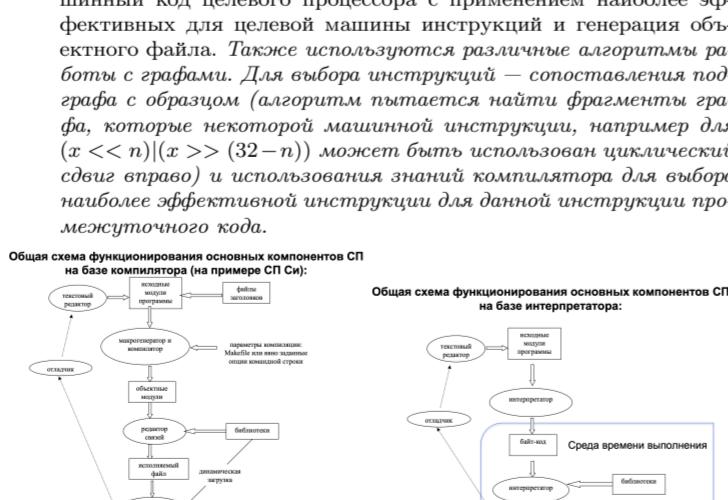
3. **Семантический анализ**. Семантический анализатор проверяет исходную программу на семантическую согласованность с определением языка (например, проверка типов). Используются различные обходы графов (DFS, BFS) и их погрешности.

4. **Генерация промежуточного кода**. Переход AST в некоторое промежуточное представление, на котором удобнее производить оптимизации. Часто применяется **SSA-форма** (single static assignment), в которой каждый переменной можно присвоить значение лишь единожды. Для ее построения используется обход графа для генерации промежуточного кода и алгоритм преобразования генерации ф-функций для преобразования в SSA.

5. **Машинно-независимая оптимизация**. Серия трансформаций промежуточного кода с целью увеличения скорости его работы на целевом процессоре с сохранением семантики работы программы. Используются различные алгоритмы работы с графиками, см. библиотеки дор25, дор26.
6. **Машинно-зависимая оптимизация и генерация кода**. Трансляция промежуточного представления компилятора в машинный код целевого процессора с применением наиболее эффективных для целевой машины инструкций и генерации объектного файла. Также используются различные алгоритмы работы с графиками. Для выбора инструкций – сопоставления подграфа с образцом (алгоритм пытается найти фрагменты графа, которые некоторой машинной инструкции, например для $(x < n)(x > (32 - n))$ может быть использован циклический сдвиг вправо) и использование знаний компилятора для выбора наиболее эффективной инструкции для данной инструкции промежуточного кода.

Общая схема функционирования основных компонентов СП на базе компилятора (на примере СП Си):

Общая схема функционирования основных компонентов СП на базе интерпретатора:



0.0.19 OSN 21 Базы данных. Основные понятия реляционной модели данных. Реляционная алгебра. Средства языка запросов SQL.

Основные понятия реляционной модели данных:

- Понятие **типа данных** в реляционной модели данных полностью соответствует понятию типа данных в языках программирования, состоит из трех основных компонентов: определение множества значений данного типа; определение набора операций, применимых к значениям типа; определение способа внешнего представления значений типа (литералов).
- **Домен** – допустимое потенциальное, ограниченное подмножество значений данного типа.
- Для уточнения термина отношение выделяются понятия заголовка отношения, значения отношения и переменной отношения. Пусть дана совокупность типов данных T_1, T_2, \dots, T_n , называемых также доменами, не обязательно различных. Тогда **n-арным отношением** R , или отношением R степени n называется подмножество декартова произведения множеств T_1, T_2, \dots, T_n .

- **Тело отношения** – это множество кортежей вида $\{< A_1, T_1, v_1 >, < A_2, T_2, v_2 >, \dots, < A_n, T_n, v_n >\}$, где $v_i \in T_i$, A_i – столбцы (атрибуты) отношения.

Алгебра Кодда – основные реляционные операции

- При выполнении операции **объединение (UNION)** двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.
- Операция **пересечение (INTERSECT)** двух отношений производится отношение, включающее все кортежи, входящие в оба отношения-операнды.
- Отношение, являющееся **разностью (MINUS)** двух отношений, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, являющееся вторым операндом.

- При выполнении **декартова произведения (TIMES)** двух отношений производится отношение, кортежи которого являются конкатнацией (специением) кортежей первого и второго операндов.

- Результатом **ограничения (WHERE)** отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющие этому условию.

- При выполнении **проекции (PROJECT)** отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого производятся путем взятия соответствующих значений из кортежей-операндов.

- При соединении (JOIN) двух отношений по некоторому условию образуется результатирующее отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию.

- У операции **реляционного деления (DIVIDE BY)** два операнда – бинарное и unary отношения. Результатирующее отношение состоит из unaryных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута включает множество значений второго операнда.

- Операция **переименования (RENAME)** производит отношение, тело которого совпадает с телом операнда, но имена атрибутов могут быть изменены.

- Операция **присваивания (=)** позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Algebra Kodda – специальные реляционные операции

Имеются важные частные случаи соединения – **экви соединение (EQUJOIN)** и **естественное соединение (NATURAL JOIN)**.

- Операция соединения называется операцией **экви соединения**, если условие соединения имеет вид $(a = b)$, где a и b – атрибуты разных операндов соединения.

0.0.21 OSN 25 Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Вронского.

Линейным дифференциальному оператором n -го порядка называется оператор \mathcal{L} , линейным дифференциальному уравнением n -го порядка называется $f(t)$:

$$\mathcal{L}y = f(t) = a_0(t)y^{(n)}(t) + a_1(t)y^{(n-1)}(t) + \dots + a_{n-1}(t)y'(t) + a_n(t)y(t),$$

где $\mathcal{L}y(t) \in C[a, b]$, $f(t) -$ комплекснозначная функция, $a_k(t) \in C[a, b]$, $a_k(t) \neq 0$, $t \in [a, b]$,

$y(t) \in C^{(n)}[a, b]$ (n раз диф-ма на отрезке $[a, b]$),

Если $f(t) = 0$ на $[a, b]$, то уравнение называется однородным, иначе неоднородным.

Теорема 1: Если функции $y_k(t)$, $k = 1..m$ являются решениями уравнений $\mathcal{L}y_k = f_k(t)$, то функция $y(t) = \sum_{k=1}^m c_k y_k(t)$, где $c_k -$ комплексные постоянные, – является решением уравнения $\mathcal{L}y = f(t)$, где $f(t) = \sum_{k=1}^m c_k f_k(t)$.

$$\Delta \mathcal{L}y = \sum_{k=1}^m c_k \mathcal{L}y_k(t) = \sum_{k=1}^m c_k f_k(t) = f(t), t \in [a, b] \blacksquare$$

Следствие: Линейная комбинация решений однородного уравнения является решением однородного уравнения. Разность двух решений неоднородного уравнения с одинаковой правой частью есть решение однородного уравнения.

Теорема 2 Решение задачи Коши $\mathcal{L}y = f(t)$, $y^{(k)}(t_0) = y_{0k}$, $k = 0, n-1$ представимо в виде $y(t) = v(t) + w(t)$, где функция $v(t)$ является решением задачи Коши для неоднородного уравнения $\mathcal{L}v = f(t)$ с нулевыми начальными условиями $v^{(k)}(t_0) = 0$, $k = \overline{0, n-1}$, а функция $w(t)$ является решением задачи Коши для однородного уравнения $\mathcal{L}w = 0$ с ненулевыми начальными условиями $w^{(k)}(t_0) = y_{0k}$, $k = 0, n-1$.

▲ Сумма $y(t) = v(t) + w(t)$ удовлетворяет неоднородному ур-ю в силу теоремы 1. Для начальных условий имеем равенства $y^{(k)}(t_0) = v^{(k)}(t_0) + w^{(k)}(t_0) = 0 + y_{0k}$, $k = 1..n-1$ ■

Скалярные производные $\varphi_1(t), \dots, \varphi_m(t)$ называются линейно зависимыми на отрезке $[a, b]$, если найдутся такие комплексные константы

$$c_k \in \mathbb{C}, k = 1, m, \sum_{k=1}^m |c_k| > 0, \text{ что справедливо равенство } \sum_{k=1}^m c_k \varphi_k(t) = 0, \forall t \in [a, b].$$

Если равенство выполнено только для $c_k = 0$, $k = 1..n$, то функции линейно независимы.

Определителем Вронского (вронсианом) системы функций $\varphi_1(t), \dots, \varphi_m(t)$, где $\varphi_i(t) \in C^{(m-1)}[a, b]$, называется зависимый от переменной $t \in [a, b]$ определитель

$$W[\varphi_1, \dots, \varphi_m](t) = \begin{vmatrix} \varphi_1(t) & \varphi_2(t) & \dots & \varphi_m(t) \\ \varphi_1'(t) & \varphi_2'(t) & \dots & \varphi_m'(t) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1^{(m-1)}(t) & \varphi_2^{(m-1)}(t) & \dots & \varphi_m^{(m-1)}(t) \end{vmatrix}$$

Теорема 3: если система скалярных функций $\varphi_1(t), \dots, \varphi_m(t)$, где $\varphi_i(t) \in C^{(m-1)}[a, b]$, является линейно зависимой на отрезке $[a, b]$, то определитель Вронского этой системы тождественно равен нулю на этом отрезке: $W[\varphi_1, \dots, \varphi_m](t) \equiv 0, \forall t \in [a, b]$.

▲ Так как функции $\varphi_k(t)$ линейно зависимы на $[a, b]$, то существует нетривиальный набор констант c_1, \dots, c_m , для которого на отрезке $[a, b]$ справедливо равенство выше. В этом равенстве допустимо починение дифференцирование до порядка $m-1$ включительно:

$$c_1\varphi_1^{(k)}(t) + \dots + c_m\varphi_m^{(k)}(t) = 0, k = 0, m-1, t \in [a, b].$$

Отсюда следует, что вектор-столбцы определителя Вронского линейно зависимы для всех $t \in [a, b]$. Следовательно, этот определитель равен нулю для всех $t \in [a, b]$. ■

Замечание. Из ЛНЗ не следует, что $W \neq 0$, контриимер: $\varphi_1(t) = t^2$, $\varphi_2(t) = -t^2, t < 0, t^2, t \geq 0$, $W \equiv 0$, но на $[-1, 1]$ функции ЛНЗ.

Теорема 4: Для решений $y_1(t), \dots, y_n(t)$ линейного однородного уравнения на отрезке $[a, b]$ справедлива следующая альтернатива:

либо $W[y_1, \dots, y_n](t) \equiv 0$ на $[a, b]$ и функции $y_1(t), \dots, y_n(t)$ линейно независимы на этом отрезке;

либо $W[y_1, \dots, y_n](t) \neq 0, \forall t \in [a, b]$ и функции $y_1(t), \dots, y_n(t)$ линейно независимы на $[a, b]$.

Фундаментальной системой решений линейного однородного дифференциального уравнения n -го порядка на отрезке $[a, b]$ называется система из n линейно независимых на данном отрезке решений этого уравнения.

Общим решением линейного однородного (неоднородного) дифференциального уравнения n -го порядка называется зависящее от n произвольных постоянных решения этого уравнения, так, что любое другое решение уравнения может быть получено из него в результате выбора некоторых значений этих постоянных.

Теорема 5: У любого линейного однородного уравнения $\mathcal{L}y = 0$ существует фундаментальная система решений на $[a, b]$.

▲ Рассмотрим постороннюю матрицу B с элементами $b_{ij}, i, j = 1, 2, \dots, n$ такую, что $det B \neq 0$. Обозначим через $y_j(t)$ решения задачи Коши для уравнения $\mathcal{L}y = 0$ с начальными условиями:

$$y_j(t_0) = b_{1j}, y'_j(t_0) = b_{2j}, \dots, y_{n-1}(t_0) = b_{nj}, j = 1, 2, \dots, n.$$

По теореме существования и единственности решения задачи Коши для линейного дифференциального уравнения n -го порядка функции $y_j(t)$ существуют и определены однозначно. Составленный из них определитель Вронского $W[y_1, \dots, y_n](t)$, в силу начальных условий, таков, что $W[y_1, \dots, y_n](t_0) = det B \neq 0$. Следовательно, по предыдущей теореме он не равен нулю ни в одной точке отрезка $[a, b]$. Значит, они образуют фундаментальную систему решений уравнения $\mathcal{L}y = 0$. ■

Теорема 6: Пусть $y_1(t), y_2(t), \dots, y_n(t)$ – фундаментальная система решений линейного однородного уравнения $\mathcal{L}y = 0$ на отрезке $[a, b]$. Тогда общее решение этого уравнения на рассматриваемом отрезке имеет вид:

$$y(t) = c_1y_1(t) + c_2y_2(t) + \dots + c_ny_n(t), \forall c_j \in \mathbb{C}. \quad (7)$$

▲ Так как линейная комбинация решений однородного уравнения $\mathcal{L}y = 0$ является решением этого уравнения, то при любых значениях постоянных c_k функция $y(t)$, определяемая формулой (7), является решением линейного однородного дифференциального уравнения $\mathcal{L}y = 0$. Покажем теперь, что любое решение уравнения $\mathcal{L}y = 0$ может быть получено из (7) в результате выбора значений постоянных c_k . Пусть $\tilde{y}(t)$ – некоторое решение уравнения $\mathcal{L}y = 0$. Рассмотрим систему алгебраических уравнений относительно неизвестных c_k :

$$\begin{cases} c_1y_1(t_0) + c_2y_2(t_0) + \dots + c_ny_n(t_0) = \tilde{y}(t_0) \\ c_1y'_1(t_0) + c_2y'_2(t_0) + \dots + c_ny'_n(t_0) = \tilde{y}'(t_0) \\ \dots \\ c_1y_{n-1}^{(n-1)}(t_0) + c_2y_{n-2}^{(n-1)}(t_0) + \dots + c_ny_n^{(n-1)}(t_0) = \tilde{y}^{(n-1)}(t_0) \end{cases} \quad (8)$$

где t_0 – некоторая точка отрезка $[a, b]$. Определитель этой системы равен определителю Вронского в точке t_0 и не равен 0, так как решения $y_1(t), y_2(t), \dots, y_n(t)$ линейно независимы. Следовательно, система (8) имеет единственное решение $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n$.

Рассмотрим функцию $\tilde{y}(t) = \sum_{k=1}^n \tilde{c}_k y_k(t)$.

Эта функция является решением уравнения $\mathcal{L}y = 0$. Так как постоянные $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n$ представляют собой решение системы (8), то функция $\tilde{y}(t)$ такова, что $\tilde{y}^{(k)}(t_0) = \tilde{y}'^{(k)}(t_0)$, $k = 0, 1, \dots, n-1$.

Следовательно, функции $\tilde{y}(t)$ и $\tilde{y}'(t)$ являются решениями уравнения $\mathcal{L}y = 0$ и удовлетворяют одни и тем же начальными условиям в точке t_0 . По теореме о существовании и единственности решения задачи Коши для n функций должны совпадать: $\tilde{y}(t) = \tilde{y}'(t) = \sum_{k=1}^n \tilde{c}_k y_k(t)$ ■

Теорема 7: Пусть $y_1(t), y_2(t), \dots, y_n(t)$ – фундаментальная система решений линейного однородного уравнения $\mathcal{L}y = 0$ на отрезке $[a, b]$, $y_n(t)$ – некоторое (частное) решение неоднородного уравнения $\mathcal{L}y = f(t)$. Тогда общее решение линейного неоднородного уравнения $\mathcal{L}y = f(t)$ на рассматриваемом отрезке имеет вид:

$$y(t) = y_H(t) + y_O(t) = y_H(t) + c_1y_1(t) + c_2y_2(t) + \dots + c_ny_n(t), \quad (9)$$

где c_1, c_2, \dots, c_n – произвольные комплексные постоянные.

▲ Для любого набора констант $c_j \in \mathbb{C}$ формула (9) определяет решение линейного неоднородного уравнения $\mathcal{L}y = f(t)$ в силу линейности уравнения. Согласно определению общего решения осталось показать, что выбором констант в (9) можно получить любое наперед заданное решение $\mathcal{L}y = f(t)$, то есть для любого решения $\tilde{y}(t)$ неоднородного уравнения $\mathcal{L}y = f(t)$ найдутся константы $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n$ такие, что на отрезке $[a, b]$ будет выполнено равенство

$$\tilde{y}(t) = y_H(t) + \tilde{c}_1y_1(t) + \tilde{c}_2y_2(t) + \dots + \tilde{c}_ny_n(t). \quad (10)$$

Пусть $\tilde{y}(t)$ – решение неоднородного уравнения $\mathcal{L}y = f(t)$. Разность $y(t) = \tilde{y}(t) - y_H(t)$ двух решений линейного неоднородного уравнения $\mathcal{L}y = f(t)$ является решением однородного уравнения $\mathcal{L}y = 0$. По теореме об общем решении линейного однородного уравнения найдутся комплексные константы \tilde{c}_j такие, что на рассматриваемом отрезке выполнено равенство $\tilde{y}(t) = \tilde{c}_1y_1(t) + \tilde{c}_2y_2(t) + \dots + \tilde{c}_ny_n(t)$, а вместе с ним и искомое равенство (10). ■

[А. М. Денисов, *Обыкновенные дифференциальные уравнения, часть 1*, page 65-73]

0.0.22 OSN 22 Виды параллельной обработки данных, их особенности. Компьютеры с общей и распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.

Параллельная обработка данных имеет две разновидности: конвейерность и параллельность.

● **Параллельная обработка.** Увеличение количества независимо работающих устройств. Если некое устройство выполняет одну операцию за единицу времени, то тысячу операций оно выполнит за тысячу единиц времени. Система из N устройств ту же работу выполнит за $1000/N$ единиц времени (в идеальном случае).

● **Конвейерная обработка.** Усложнение самого устройства, чтобы на разных этапах могли находиться разные данные. Идея заключается в выделении отдельных этапов выполнения общей операции, причем каждый этап, выполнив свою работу, передавал бы результат следующему, одновременно принимая новую порцию входных данных. Получаем выигрыш в скорости обработки за счет сокращения прежде разнесенных во времени операций. Существует некоторая задержка (время задержки) между этапами конвейера.

Классификация многопроцессорных сетей по Флинну.

В контексте машины можно выделить два потока информации: поток управления (для передачи управляющих воздействий на конкретное устройство) и поток данных (циклический между оперативной памятью и внешними устройствами). В связи с этим выделяют 4 основных класса: SIMD (1 поток команд, 1 поток данных), "Традиционный" (последовательный компьютер), SIMD (1 поток команд, много потоков данных, пример - векторные компьютеры), MIMD (много потоков команд, много потоков данных). Среди MIMD можно выделить системы с общей ОП и системы с распределенной памятью.

● **Компьютеры с общей памятью.**

В системе присутствует несколько равноправных процессоров, имеющих одинаковый доступ к единой памяти. Всё, кроме процессоров, в одном экземпляре: образ операционной системы, память, подсистема ввода-вывода и т.д. Все процессоры работают с единным адресным пространством. (+) относительная простота параллельного программирования; (-) сложность увеличения числа процессоров (рост производительности).

UMA – системы с однородным доступом к памяти (все процессоры имеют одинаковый доступ к памяти). SMP – общая шина, соединенная с всеми процессорами и с ОП.

NUMA (Non Uniform Memory Access) – память физически распределена, но логически общедоступна. Каждый вычислительныйузел компьютера содержит процессор, локальную память, контроллер памяти, быть может, некоторые устройства ввода/вывода. Контроллер памяти определяет, является ли запрос к памяти локальным или его необходимо передать удаленному узлу через коммутатор/шину. Проблема – синхронизация кэша.

сeNUMA (cache coherent NUMA). На аппаратном уровне решает проблему когерентности кэшей. Но остаются ограничения, связанные с централизацией – использование системой шиной, возникающие при возникновении логической конфликта.

сeNUMA (cache coherent NUMA). На аппаратном уровне решает проблему когерентности кэшей. Но остаются ограничения, связанные с централизацией – использование системой шиной, возникающие при возникновении логической конфликта.

сeNUMA (cache coherent NUMA). На аппаратном уровне решает проблему когерентности кэшей. Но остаются ограничения, связанные с централизацией – использование системой шиной, возникающие при возникновении логической конфликта.

сeNUMA (cache coherent NUMA). На аппаратном уровне решает проблему когерентности кэшей. Но остаются ограничения, связанные с централизацией – использование системой шиной, возникающие при возникновении логической конфликта.

0.0.25 OSN 26 Теоремы существования и единственности решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной.

Пусть функция $f(t, y)$ определена и непрерывна в прямоугольнике $\Pi = \{(t, y) : |t - t_0| \leq T; |y - y_0| \leq A\}$. Рассмотрим на отрезке $[t_0 - T; t_0 + T]$ дифференциальное уравнение с условием:

$$y'(t) = f(t, y(t)) \quad (11)$$

$$y(t_0) = y_0 \quad (12)$$

Требуется определить функцию $y(t)$, удовлетворяющую уравнению (11) и условию (12).

Эта задача называется **задачей с начальным условием или задачей Коши**. Рассмотрим отрезок $[t_1, t_2]$ такой, что $t_0 - T \leq t_1 < t_2 \leq t_0 + T$, $t_0 \in [t_1, t_2]$.

Оп. Функция $y(t)$ называется **решением задачи Коши** (11), (12) на отрезке $[t_1, t_2]$, если $y(t) \in C[t_1, t_2]$, $|y(t) - y_0| \leq A$ для $t \in [t_1, t_2]$, $y(t)$ удовлетворяет уравнению (11) для $t \in [t_1, t_2]$ и (12).

Рассмотрим на отрезке $[t_0 - T, t_0 + T]$ уравнение относительно неизвестной функции $y(t)$:

$$y(t) = y_0 + \int_{t_0}^t f(\tau, y(\tau)) d\tau \quad (13)$$

Лемма 1. Функция $y(t)$ является решением задачи Коши (11), (12) на отрезке $[t_1, t_2] \iff$ когда $y(t) \in C[t_1, t_2]$, $|y(t) - y_0| \leq A$ для $t \in [t_1, t_2]$ и $y(t)$ удовлетворяет уравнению (13) для $t \in [t_1, t_2]$.

▲ (\Rightarrow) Пусть функция $\bar{y}(t)$ является решением задачи с начальным условием (11), (12) на отрезке $[t_1, t_2]$. Из определения решения следует, что $\bar{y}(t) \in C[t_1, t_2]$, $|\bar{y}(t) - y_0| \leq A$ для $t \in [t_1, t_2]$. Покажем, что $\bar{y}(t)$ удовлетворяет уравнению (13) для $t \in [t_1, t_2]$. Интегрируя (11) от t_0 до t получим:

$$\int_{t_0}^t \bar{y}'(\tau) d\tau = \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2]$$

Учитывая условие (12), имеем:

$$\bar{y}(t) = y_0 + \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2]$$

Следовательно, функция $\bar{y}(t)$ удовлетворяет интегральному уравнению (13) при $t \in [t_1, t_2]$.

(\Leftarrow) Пусть функция $\bar{y}(t)$ такова, что $\bar{y}(t) \in C[t_1, t_2]$, $|\bar{y}(t) - y_0| \leq A$ для $t \in [t_1, t_2]$ и $\bar{y}(t)$ удовлетворяет уравнению (13) для $t \in [t_1, t_2]$, то есть:

$$\bar{y}(t) = y_0 + \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2] \quad (14)$$

Покажем, что $y(t)$ является решением задачи с начальным условием (11), (12).

Положив (14) $t = t_0$, получим, что $\bar{y}(0) = y_0$. Следовательно условие (12) выполнено. Так как функция $\bar{y}(t)$ непрерывна на $[t_1, t_2]$, то правая часть равенства (14) непрерывно дифференцируема на $[t_1, t_2]$ как интеграл с переменным верхним пределом t от непрерывной функции $f(\tau, \bar{y}(\tau)) \in C[t_1, t_2]$. Следовательно, $\bar{y}(t)$ непрерывно дифференцируема на $[t_1, t_2]$. Дифференируя (14), получим, что $\bar{y}'(t)$ удовлетворяет (11). ■

Оп. Функция $f(t, y)$, заданная прямоугольнике Π , удовлетворяет в Π условию Липшица по y , если $|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|$, $\forall (t, y_1), (t, y_2) \in \Pi$, где L — положительная постоянная.

Лемма Гроновиля-Беллмана. Пусть функция $z(t) \in C[a, b]$ и таково, что $0 \leq z(t) \leq c + d \left| \int_t^b z(\tau) d\tau \right|$, $t \in [a, b]$, где постоянная c неотрицательна, постоянная d положительна, а t_0 — произвольное фиксированное число на отрезке $[a, b]$. Тогда $z(t) \leq ce^{d|t-t_0|}$, $t \in [a, b]$.

Теорема (единственности). Пусть функция $f(t, y)$ непрерывна в Π и удовлетворяет в Π условию Липшица по y . Если $y_1(t)$, $y_2(t)$ — решения задачи Коши (11), (12) на отрезке $[t_1, t_2]$, то $y_1(t) = y_2(t)$ для $t \in [t_1, t_2]$.

▲ Так как $y_1(t)$ и $y_2(t)$ — решения задачи Коши (11), (12), то из Леммы 1 следует, что они являются решениями интегрального уравнения (13). То есть:

$$y_1(t) = y_0 + \int_{t_0}^t f(\tau, y_1(\tau)) d\tau, \quad t \in [t_1, t_2],$$

$$y_2(t) = y_0 + \int_{t_0}^t f(\tau, y_2(\tau)) d\tau, \quad t \in [t_1, t_2].$$

Вычитая второе уравнение из первого и оценивая разность по модулю и используя условие Липшица, получаем: $|y_1(t) - y_2(t)| = \left| \int_{t_0}^t f(\tau, y_1(\tau)) d\tau - \int_{t_0}^t f(\tau, y_2(\tau)) d\tau \right| \leq \left| \int_{t_0}^t |f(\tau, y_1(\tau)) - f(\tau, y_2(\tau))| d\tau \right| \leq L \left| \int_{t_0}^t |y_1(\tau) - y_2(\tau)| d\tau \right|$

Обозначив $z(t) = |y_1(t) - y_2(t)|$, перепишем последнее неравенство следующим образом:

$$0 \leq z(t) \leq L \left| \int_{t_0}^t z(\tau) d\tau \right|, \quad t \in [t_1, t_2].$$

Применяя лемму Гроновиля-Беллмана с $c = 0$ и $d = L$, имеем $z(t) = 0$, $t \in [t_1, t_2]$. Следовательно, $y_1(t) = y_2(t)$, $t \in [t_1, t_2]$. ■

Теорема (существования) ((локальная)). Пусть функция $f(t, y)$ непрерывна в Π , удовлетворяет в Π условию Липшица по y и $|f(t, y)| \leq M$, $(t, y) \in \Pi$. Тогда на отрезке $[t_0 - h, t_0 + h]$, где $h = \min\{T, \frac{A}{M}\}$, существует функция $y(t)$ такая, что $y(t) \in C^1[t_0 - h, t_0 + h]$, $|y(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$, $y'(t) = f(t, y(t))$, $t \in [t_0 - h, t_0 + h]$, $y(t_0) = y_0$.

Следует отметить, что мы можем доказать теорему существования не на всем исходном отрезке $[t_0 - T, t_0 + T]$, а на некотором, вообще говоря, меньшем. Поэтому эта теорема часто называется **локальной теоремой существования решения задачи Коши**.

[А. М. Денисов, *Обыкновенные дифференциальные уравнения, часть I*, page 25-30]

0.0.26 OSN 28 Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых методом Шеннона.

Орграф — это ориентированный граф. Вершины орграфа, в которые не входит ни одной дуги, называются **истоками**. Орграф называется **акиклическим**, если в нем нет ориентированных циклов. Система Б: g_1, g_2, \dots, g_m , где все g_i — функции алгебры логики, будем называть **базисом функциональных элементов**. Орграф называется **упорядоченным**, если для каждой вершины v_i , в которую входит k_i дуг, задан порядок e_1, e_2, \dots, e_{k_i} этих дуг.

Схемой из функциональных элементов в базисе Б называется акиклический упорядоченный орграф, в котором:

• каждому истоку присвоена некоторая переменная, причем разными истоками приписаны разные переменные (истоки при этом называются входами схемы, а присвоенные им переменные — входными переменными);

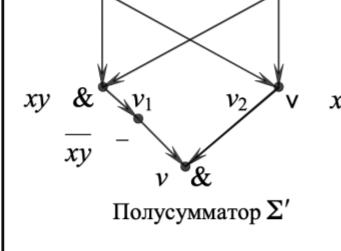
• каждой вершине, в которую входят $k \geq 1$ дуг, присвоена функция из базиса Б, зависящая от k переменных (вершина с присвоенной функцией при этом называется **функциональным элементом**);

• некоторые вершины выделены как **выходы**.

Сложностью схемы из функциональных элементов называется число функциональных элементов в схеме (число внутренних вершин).

Пример:

Полусумматор Пусть v и v_1 — выходы на рисунке, $f_v = x \wedge y \wedge (x \vee y) = xy$; $f_{v_1} = x \wedge y$. Сложность (число элементов) полусумматора равна 4.



Полусумматор Σ'

Важные ФАЛ и системы ФАЛ:

• Мультиплексор ФАЛ μ_n порядка n

$$\mu(x_1, \dots, x_n, y_0, \dots, y_{2^n-1}) = \bigvee_{\alpha=(\alpha_1, \dots, \alpha_n)} x_1^{\alpha_1} \dots x_n^{\alpha_n} y_{\nu(\alpha)},$$

где $\nu(\alpha)$ — перевод двоичного числа α в десятичное.

• Универсальная система $\vec{P}_2(n)$ порядка n — содержит все ФАЛ от n переменных. Реализуется мультиплексором.

• Акиклическая система $\vec{P}_2(n)$ порядка n — содержит все ФАЛ от n переменных. Реализуется универсальным многополосником.

Лемма. Для каждого натурального n существует СФЭ над базисом B , $B \subseteq U_B^C$ (множество всех схем на базисом B), которая реализует систему ФАЛ $\vec{P}_2(n)$ и сложность которой равна $2^{2^n} - n$.

▲ В силу полноты базиса в B существует система СФЭ Σ от БП x_1, \dots, x_n , реализующая систему ФАЛ $\vec{P}_2(n)$. Искомая СФЭ U_n является строгое призведённой СФЭ, которая эквивалентна Σ и получается из неё в результате операции присоединения эквивалентных вершин и удаления висячих вершин. Действительно, из построения следует, что число всех вершин СФЭ U_n , включая n её входов, равно 2^{2^n} и поэтому $L(U_n) = 2^{2^n} - n$ (вычитаем n входов, которые автоматически реализуют функции x_1, \dots, x_n). ■

Следствие. $L_B^C(\vec{P}_2(n)) \leq 2^{2^n} - n$.

Базис $B_0 = \{\&, \vee, \neg\}$

Определения сложности.

Сложность ФАЛ: $L_B(f) = \min_{\text{СФЭ } \Sigma \in U_B^C \text{ реализующие } f} L(\Sigma)$

Функция Шеннона: $L_B(n) = \max_{f \in \vec{P}_2(n)} L_B(f)$

Синтез по совершенной ДНФ

Совершенная ДНФ $f(x_1, \dots, x_n) = \bigvee_{\sigma \in N_f} x_1^{\sigma_1} \wedge \dots \wedge x_n^{\sigma_n}$, где N_f — все наборы σ , на которых $f(\sigma) = 1$.

Совершенная ДНФ — формула в B_0 , значит существует СФЭ Σ_f над базисом B_0 , которая реализует f .

Тогда $L(\Sigma_f) \leq \sum_{i=1}^n (n-1) + \sum_{j=1}^n + 2^n - 1$, где 1 — верхняя оценка $|N_f|$,

т.е. количество дизъюнктов в совершенной ДНФ, 2 — количество конъюнкций в каждом дизъюнкте, 3 — верхняя оценка количества отрицаний в дизъюнкте, 4 — оценка количества дизъюнкций между дизъюнктами. СФЭ — частный случай квазидеревьев, которые эквивалентны формулам, поэтому $L^C(n) \leq L^B(n)$. Так получаем верхнюю оценку функции Шеннона: $L^C(n) \leq L(\Sigma_f) \leq n \cdot 2^{n+1}$.

Метод Шеннона.

Выбираем параметр q , $1 \leq q \leq n$.

Использование разложения Шеннона:

$$f(x_1, \dots, x_q, x_{q+1}, \dots, x_n) = \bigvee_{\sigma''=(\sigma_{q+1}, \dots, \sigma_n)} x_{q+1}^{\sigma_{q+1}} \dots x_n^{\sigma_n} \cdot f_{\sigma''}(x_1, \dots, x_q, \sigma_{q+1}, \dots, \sigma_n)$$

Для любой ФАЛ $f \in P_2(n)$ строим СФЭ Σ_f как суперпозицию $\Sigma''(\Sigma')$, где $\Sigma' =$ мультиплексор, $\Sigma'' =$ универсальный многополосник.

Сложность многополосника от q переменных: $L(\Sigma') \leq 2^{2^q} - q$ (следует из леммы)

Сложность мультиплексора от $n - q$ переменных: $L(\Sigma'') \leq 2^{n-q+2} - 3$

Полагая $q = \lceil \log_2(n - 2 \log n) \rceil$ получаем в результате преобразований:

$$L(\Sigma_f) \leq 2^{2^q} + 4 \cdot 2^{n-q} \leq \frac{8 \cdot 2^n}{n - 2 \log n} + O\left(\frac{2^n}{n^2}\right)$$

Таким образом, верна оценка сложности СФЭ: $L^C(n) \lesssim 8 \cdot \frac{2^n}{n}$

[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

0.0.27 OSN 30 Квадратурные

0.0.29 OSN 33 Задача Коши для уравнения колебания струны. Формула Даламбера.

Задача Коши для уравнения колебания струны.

$$\begin{cases} u_{tt} = a^2 u_{xx} + f(x, t) \\ u(x, 0) = \varphi(x) \\ u_t(x, 0) = \psi(x) \end{cases}$$

где $t > 0$, $a > 0$, $u(x, t) \in C^2(t > 0, x \in \mathbb{R}) \cap C^1(t \geq 0, x \in \mathbb{R})$.

Физическая интерпретация: уравнение малых поперечных колебаний струны. $u(x, t)$ – положение точки струны с координатой x в момент времени t . Если концы струны закреплены, то $u(0, t) = 0$, $u(l, t) = 0$. Так как процесс колебаний струны зависит от ее начальной формы и распределения скоростей, то задают начальные условия. $a = \sqrt{\frac{T_0}{\rho}}$, где T_0 – величина натяжения (не зависит от x , ρ – линейная плотность струны). $f(x, t)$ – положение точки струны с координатой x в момент времени t . Если концы струны закреплены, то $u(0, t) = 0$, $u(l, t) = 0$. Так как процесс колебаний струны зависит от ее начальной формы и распределения скоростей, то задают начальные условия. $a = \sqrt{\frac{T_0}{\rho}}$, где

T_0 – величина натяжения (не зависит от x , ρ – линейная плотность струны). $f(x, t)$ – положение точки струны с координатой x в момент времени t . Если концы струны закреплены, то $u(0, t) = 0$, $u(l, t) = 0$. Так как процесс колебаний струны зависит от ее начальной формы и распределения скоростей, то задают начальные условия. $a = \sqrt{\frac{T_0}{\rho}}$, где

далее будем рассматривать $f(x, t) = 0$.

(Представим что) $\frac{\partial^2 u}{\partial t^2} = \frac{d^2 u}{dt^2}$, $u \frac{\partial^2 u}{\partial x^2} = \frac{d^2 u}{dx^2}$, $\frac{d^2 u}{dt^2} = a^2 \frac{d^2 u}{dx^2} \Rightarrow d^2 u dx^2 = a^2 d^2 t^2 \Rightarrow dx^2 = a^2 dt^2$. Характеристическое уравнение: $dx^2 - a^2 dt^2 = 0$. $dx - adt = 0 \Rightarrow x - at = C_1$, $x + at = C_2 = const$. Сделаем замену переменных:

$$x + at = \xi, \quad x - at = \eta$$

$$\xi_x = 1, \quad \xi_{xx} = 0, \quad \eta_x = 1, \quad \eta_{xx} = 0,$$

$$\xi_t = a, \quad \xi_{tt} = 0, \quad \eta_t = -a, \quad \eta_{tt} = 0.$$

Тогда:

$$u_x = u_\xi \cdot \xi_x + u_\eta \cdot \eta_x,$$

$$u_t = u_\xi \cdot \xi_t + u_\eta \cdot \eta_t,$$

$$u_{xx} = u_{\xi\xi} \cdot \xi_x^2 + 2u_{\xi\eta} \cdot \xi_x \cdot \eta_x + u_\xi \cdot \xi_{xx} + u_{\eta\eta} \cdot \eta_x^2 + u_\eta \cdot \eta_{xx},$$

$$u_{tt} = u_{\xi\xi} \cdot \xi_t^2 + 2u_{\xi\eta} \cdot \xi_t \cdot \eta_t + u_\xi \cdot \xi_{tt} + u_{\eta\eta} \cdot \eta_t^2 + u_\eta \cdot \eta_{tt}.$$

Подставляем в уравнение $u_{tt} = a^2 u_{xx}$:

$$\begin{aligned} u_{\xi\xi} \cdot \xi_x^2 + 2u_{\xi\eta} \cdot \xi_x \cdot \eta_x + u_\xi \cdot \xi_{xx} + u_{\eta\eta} \cdot \eta_x^2 + u_\eta \cdot \eta_{xx} \\ = 0 \\ = a^2(u_{\xi\xi} \cdot \xi_x^2 + 2u_{\xi\eta} \cdot \xi_x \cdot \eta_x + u_\xi \cdot \xi_{xx} + u_{\eta\eta} \cdot \eta_x^2 + u_\eta \cdot \eta_{xx}) \\ = 0 \end{aligned}$$

Преобразуем:

$$a^2 u_{\xi\xi} - 2a^2 u_{\xi\eta} + a^2 u_{\eta\eta} = a^2 u_{\xi\xi} + 2a^2 u_{\xi\eta} + a^2 u_{\eta\eta}$$

$$4a^2 u_{\xi\eta} = 0, \quad a > 0$$

Получаем: $u_{\xi\eta} = 0$

Найдем общий интеграл этого уравнения: $u_\eta(\xi, \eta) = f^*(\eta)$, где $f^*(\eta)$ – некоторая функция только переменного η .

Интегрируя это равенство по η при фиксированном ξ , получим:

$$u(\xi, \eta) = \int f^*(\eta) d\eta = f_1(\xi) + f_2(\eta) \quad (19)$$

Обратно, каковы бы ни были дважды дифференцируемые функции f_1 и f_2 , функция $u(\xi, \eta)$, определяемая формулой (19), представляет собой решение уравнения $u_{tt} = 0$. Так как всякое решение уравнения $u_{tt} = 0$ может быть представлено в виде (19) при соответствующем выборе f_1 и f_2 , то формула (19) является общим интегралом этого уравнения. Следовательно, функция

$$u(x, t) = f_1(x + at) + f_2(x - at)$$

является общим интегралом уравнения $u_{tt} = a^2 u_{xx}$.

Удовлетворим начальным условиям:

$$\begin{cases} u(x, 0) = f_1(x) + f_2(0) = \varphi(x) \\ u_t(x, 0) = -af'_1(x) + af'_2(0) = \psi(x) \end{cases}$$

Проинтегрируем второе равенство, получим:

$$\begin{cases} f_1(x) + f_2(0) = \varphi(x) \\ f_1(x) - f_2(0) = \frac{1}{a} \int_{x_0}^x \psi(\alpha) d\alpha + C \end{cases}$$

Сложим и вычтем два полученных равенства:

$$\begin{cases} f_1(x) = \frac{1}{2} \varphi(x) + \frac{1}{2a} \int_{x_0}^x \psi(\alpha) d\alpha + \frac{C}{2} \\ f_2(x) = \frac{1}{2} \varphi(x) - \frac{1}{2a} \int_{x_0}^x \psi(\alpha) d\alpha - \frac{C}{2} \end{cases}$$

Подставим в $u(\xi, \eta) = f_1(x + at) + f_2(x - at)$ найденные выражения для f_1, f_2 :

$$u(x, t) = \frac{\varphi(x + at) + \varphi(x - at)}{2} + \frac{1}{2a} \left(\int_{x_0}^{x+at} \psi(\alpha) d\alpha - \int_{x_0}^{x-at} \psi(\alpha) d\alpha \right)$$

Окончательно:

$$u(x, t) = \frac{\varphi(x + at) + \varphi(x - at)}{2} + \frac{1}{2a} \int_{x-at}^{x+at} \psi(\alpha) d\alpha$$

Полученная формула – формула Даламбера.

Теорема о применимости формулы Даламбера. Пусть $\varphi(x) \in C^2(-\infty, \infty)$, $\psi \in C^1[0, +\infty)$, $a > 0$. Тогда формула Даламбера представляет единственное решение задачи Коши для уравнения колебания струны.

[А. Н. Тихонов, Уравнения математической физики, page 50-52]

0.0.30 OSN 31 Методы Ньютона и секущих для решения нелинейных уравнений.

• ф-но $f(x)$, $x \in \mathbb{R}$, и ур-не $f(x) = 0$.

□ $x^* \in \mathbb{R}$ – корень уравнения, и определено его окрестность радиуса a , не содержащая других корней уравнения: $U_a(x^*) = \{x : |x - x^*| < a\}$, причем заданная функция $f(x)$ определена на этой окрестности. Считаем, что начальное приближение $x^0 \in U_a(x^*)$ задано. Тогда для нахождения численного решения уравнения в рассматриваемой окрестности необходимо построить последовательность $\{x^n\}$, сходящуюся к корню x^* уравнения: $\lim_{n \rightarrow \infty} f(x^n) = f(x^*) = 0$.

Численное решение нелинейных уравнений можно разбить на 2 этапа:

1. Локализация корня, т.е. определение окрестности $U_a(x^*)$.

2. Задание итерационного процесса – построение последовательности $\{x^n\}$, сходящейся к корню уравнения.

Метод Ньютона

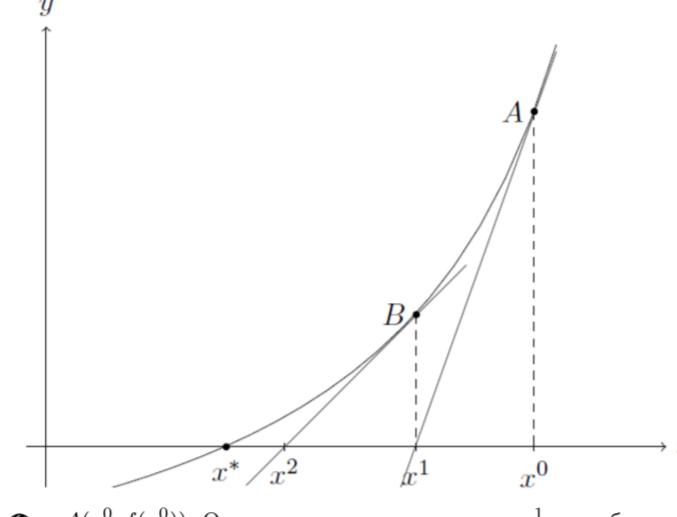
□ В $U_a(x^*)$ существует и не обращается в ноль непрерывная первая производная функции $f(x)$: $f'(x) \neq 0, \quad x \in U_a(x^*)$.

Разложим $f(x^*)$ по формуле Тейлора в малой окрестности точки $x \in U_a(x^*)$: $f(x^*) = f(x) + (x^* - x)f'(x) + \dots$, и отбросим в этом разложении величины, имеющие второй и выше порядок малости по $(x^* - x)$. Заменив x^* на x^{n+1} и x на x^n , получим ур-ие $f(x^n) + (x^{n+1} - x^n)f'(x^n) = 0$, $n \in \mathbb{Z}_+$.

Учитывая, что $f'(x^n) \neq 0$, имеем:

$$x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)}, \quad n \in \mathbb{Z}_+. \quad (20)$$

Итерационный процесс поиска корня уравнения $f(x) = 0$, задаваемый формулой (20), называется **итерационным методом Ньютона**.



• т. A($x^0, f(x^0)$). Определим первое итерацию x^1 как абсциссу точки пересечения с осью Oх касательной к $f(x)$, проведенной через A.

Аналогично получаем значение x^2 . Продолжая, на n -ом шаге получим значение x^n , приближающее корень x^* уравнения $f(x) = 0$ с заданной точностью.

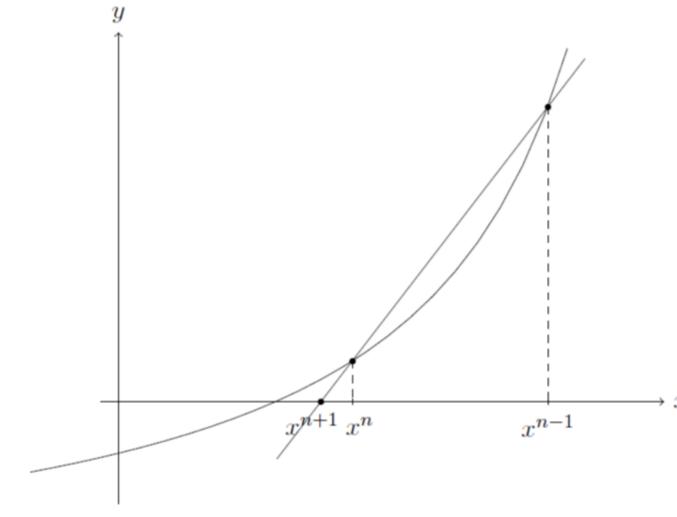
Зам. При решении задач на практике часто рассматривается модифицированный метод Ньютона, задаваемый формулой $x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)}$, $n \in \mathbb{Z}_+$ (чтобы считать пр-ю только один раз).

Метод секущих

В методе Ньютона (20) заменим $f'(x)$ на его дискретный аналог $\frac{f(x^n) - f(x^{n-1})}{x^n - x^{n-1}}$, получаем итерационный метод:

$$x^{n+1} = x^n - \frac{(x^n - x^{n-1})f(x^n)}{(f(x^n) - f(x^{n-1}))} \quad (21)$$

Итерационный процесс 21 задает двухшаговый метод решения уравнений, называемый **методом секущих**.



Сходимость метода Ньютона

Если рассмотреть итерационный метод Ньютона как метод простой итерации $x^{n+1} = S(x^n)$, $n \in \mathbb{Z}_+$ с функцией $S(x) = x - \frac{f(x)}{f'(x)}$.

• $|S'(x)| < 1$ при $x \in U_a(x^*)$, то он сходится. Предполагая, что функция $f(x)$ дифференцируема достаточно число раз, проидифференцируем формулю $S(x)$:

$$S'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}.$$

Теорема: \exists такая константа $M > 0$, для которой выполнена оценка $\frac{1}{2} |S''(x)| \leq M$, $x \in U_a(x^*)$. Тогда если начальное приближение x^0 выбрать в соответствии с условием $|x^0 - x^*| < \frac{1}{M}$, то итерационный метод Ньютона сходится, и имеет место оценка: $|x^n - x^*| \leq M|x^0 - x^*|^2$.

• Погрешность приближенного решения: $x^n = x^* - x^*$.

• выражение для x^{n+1} : $x^{n+1} = x^n - x^* = S(x^n + x^*) - S(x^*)$.

Разложим $S(x^n + x^*)$ по формуле Тейлора и учитывая $S'(x^*) = 0$:

$$z^{n+1} = S(x^n) + S'(x^n)z^n + \frac{1}{2}S''(x^n)(z^n)^2 - S(x^*) = \frac{1}{2}S''(x^n)(z^n)^2,$$

$$x^n = x^* + \theta z^n, \quad \theta \in \mathbb{R}, \quad |\theta| < 1.$$

• функция $f(x)$ трижды непрерывно дифференцируема в окрестности $U_a(x^*)$. Тогда $S''(x) = \left(\frac{f(x)f''(x)}{(f'(x))^2} \right)'$.

• \exists постоянная $M > 0$ такая, что для любого $x \in U_a(x^*)$ выполняется неравенство $M \geq \frac{1}{2} |S''(x)|$.

Из этого неравенства и уравнения z^{n+1} следует оценка $|z^{n+1}| \leq M|(z^n)^2|$.

Домножим это неравенство на M и обозначим $v^n = M|z^n|$. Тогда получим, что $v^{n+1} \leq (v^n)^2$.

Отсюда следует, что $v^n \leq (v^0)^{2^n}$, значит, $M|z^n| \leq (M|z^0|)^{2^n}$, $|z^n| \leq \frac{1}{M}(M|z^0|)^{2^n}$.

Введем обозначение $q = M|z^0|$. Если $0 < q < 1$, то последовательность $\{z^n\}_{n=0}^\infty$ стремится к нулю: $z^n \xrightarrow{n \rightarrow \infty} 0$, и итерационный метод Ньютона сходит. Условие на q ($0 < q < 1$) будет выполнено, если $0 < |z^0| < \frac{1}{M}$, то есть $|x^0 - x^*| < \frac{1}{M}$. ■

</

0.0.33 DOP 1 Теорема Поста о полноте систем функций в алгебре логики.

Полная система (P_2) — множество \mathcal{A} ФАЛ такое, что любую ФАЛ можно выразить формулой над \mathcal{A} .

$\Box \mathcal{A} \subseteq P_2$. Тогда замыкание \mathcal{A} (обозн. $[A]$) — множество всех ФАЛ, которые можно выразить формулами над \mathcal{A} .

Класс \mathcal{A} называется замкнутым, если $\mathcal{A} = [A]$.

Говорят, что набор $\tilde{\alpha} = (\alpha_1, \dots, \alpha_n)$ предшествует набору $\tilde{\beta} = (\beta_1, \dots, \beta_n)$, $\tilde{\alpha} \preceq \tilde{\beta}$ ($\tilde{\alpha}$ не больше $\tilde{\beta}$), если $\alpha_i \leq \beta_i$ для всех $i = 1, \dots, n$.

Если $\tilde{\alpha} \preceq \tilde{\beta}$ и $\tilde{\alpha} \neq \tilde{\beta}$, то говорят, что набор $\tilde{\alpha}$ строго предшествует набору $\tilde{\beta}$, $\tilde{\alpha} < \tilde{\beta}$.

Наборы $\tilde{\alpha}$ и $\tilde{\beta}$ называются сравнимыми, если $\tilde{\alpha} \preceq \tilde{\beta}$ либо $\tilde{\beta} \preceq \tilde{\alpha}$.

Например, векторы $[0, 1]$ и $[1, 0]$ — не сравнимы.

Стандартные замкнутые классы:

- $T_0 = \{f \in P_2 | f(0, \dots, 0) = 0\}$; — сохраняющие 0

- $T_1 = \{f \in P_2 | f(1, \dots, 1) = 1\}$; — сохраняющие 1

- $L = \{f(x_1, \dots, x_n) = a_0 + a_1x_1 + \dots + a_nx_n\}$ — линейные функции;

- $S = \{f(x_1, \dots, x_n) = f^*(x_1, \dots, x_n) = \bar{f}(x_1, \dots, x_n)\}$ — самодвойственные функции;

- $M = \{\alpha \leq \beta \mid f(\alpha) \leq f(\beta)\}$ — монотонные функции;

Вспомогательные леммы:

1. Если булева функция f немонотона, то из нее подстановкой вместо аргументов констант 0 и 1 и переменной x можно получить \bar{x} .

▲ Немонотона $\Rightarrow \exists$ два набора $\alpha < \beta$, $f(\alpha) = 1 > f(\beta) = 0$. Будем заменять в α 0 на 1 по одному, чтобы получился β , промежуточные назовем α_k , $\alpha_0 = \alpha$, $\alpha_r = \beta$. В какой-то момент получим $f(\alpha_k) = 1$, $f(\alpha_{k+1}) = 0$, получили два соседних набора α_k и α_{k+1} . Пусть различаются в i -й переменной. Заменим в α_k i -ую переменную на x , получим $\tilde{\alpha}_k$, $f(\tilde{\alpha}_k) = \bar{x}$ ■

2. Если булева функция несамодвойственна, то из неё подстановкой вместо аргументов переменной x и её отрицания \bar{x} можно получить \bar{x} .

▲ Несамодвойственна, то \exists т.ч. $f(\alpha) = f(\bar{\alpha}) = C$. Рассмотрим $\phi(x) = f(x \oplus \alpha_1, \dots, x \oplus \alpha_n)$. Тогда в зависимости от x в аргументах функции набор α или $\bar{\alpha}$, в любом случае $\phi(x) = C$. ■

3. Если булева функция нелинейна, то из неё подстановкой вместо аргументов констант, переменных x , y , их отрицаний \bar{x} , \bar{y} можно получить $x \cdot y$ или $\bar{x} \cdot \bar{y}$.

▲ Рассмотрим полином Жегалкина P_f функции f (представление в виде \oplus из конъюнкций). В нем найдется слагаемое, которое конъюнкциями двух или более переменных, пусть $x_1 \cdot x_2 \cdot \dots \cdot x_r$.

$P_f = x_1 \cdot x_2 \cdot g_1(x_3, \dots, x_n) \oplus x_1 \cdot g_2(x_3, \dots, x_n) \oplus x_2 \cdot g_3(x_3, \dots, x_n) \oplus g_4(x_3, \dots, x_n)$

Либо $g_1 = 1$ (если $r = 2$), либо $\exists \alpha$ т.ч. $g_1(\alpha) = 1$. Обозначим $g_2(\alpha) = a$, $g_3(\alpha) = b$, $g_4(\alpha) = c$.

$\phi(x, y) = f(x \oplus b)(y \oplus a) \oplus a(x \oplus b) \oplus b(y \oplus a) \oplus c = \{\text{раскройте сами}\} xy \oplus d$, d -константа. ■

Теорема Поста. Система ФАЛ $\mathcal{A} = \{f_1, f_2, \dots\}$ является полной в P_2 \Leftrightarrow она не содержится целиком ни в одном из следующих классов: T_0 , T_1 , S , L , M .

▲ Необходимость. Пусть \mathcal{A} — полная система, N — любой из классов T_0 , T_1 , S , L , M , и (от противного) пусть $\mathcal{A} \subseteq N$. Тогда $[\mathcal{A}] \subseteq [N] = N \neq P_2$, то есть $[\mathcal{A}] \neq P_2$. Полученное противоречие завершает обоснование необходимости.

Достаточность. Пусть \mathcal{A} не является подмножеством ни одного из этих классов. Тогда в \mathcal{A} существуют функции $f_0 \notin T_0$, $f_1 \notin T_1$, $f_L \notin L$, $f_M \notin M$, $f_S \notin S$. Пусть $B = \{f_0, f_1, f_M, f_L, f_S\}$. Достаточно показать, что $[\mathcal{A}] \supseteq [B] = P_2$.

Выразим формулами над B все функции из полной системы $\{0, 1, \bar{x}, \cdot, \bar{\cdot}, \wedge, \vee, \neg\}$.

• Получение отрицания. Рассмотрим функцию $f_0(x_1, \dots, x_n) \notin T_0$ и введём функцию $\varphi_0(x) = f_0(x, x, \dots, x)$. Так как функция f_0 не сохраняет пуль, $\varphi_0(0, 0, \dots, 0) = 1$. Возможны два случая: либо $\varphi_0(x) = \bar{x}$, либо $\varphi_0(x) \equiv 1$.

Рассмотрим функцию $f_1(x_1, \dots, x_n) \notin T_1$ и введём функцию $\varphi_1(x) = f_1(x, x, \dots, x)$. Так как функция f_1 не сохраняет единицу, $\varphi_1(1) = f_1(1, 1, \dots, 1) = 0$. Возможны два случая: либо $\varphi_1 = \bar{x}$, либо $\varphi_1(x) \equiv 0$.

Если хотя бы в одном случае получилось искомое отрицание, пункт завершен. Если же в обоих случаях получились константы, то согласно лемме о немонотонной функции, подставляя в функцию вместо всех переменных константы и тождественные функции, можно получить отрицание.

Отрицание получено.

• Получение констант 0 и 1. Имеем $f_S \notin S$. Согласно лемме о несамодвойственной функции, подставляя вместо всех переменных функции f_S отрицание (которое получено в пункте 1) и тождественную функцию, можно получить константы: $[f_S, \bar{x}] \supseteq [0, 1]$.

Константы получены.

• Получение конъюнкции. Имеем функцию $f_L \notin L$. Согласно лемме о илинейной функции, подставляя в функцию вместо всех переменных константы и отрицания (которые были получены на предыдущих шагах доказательства), можно получить либо конъюнкцию, либо отрицание конъюнкции. Однако на первом этапе отрицание уже получено, следовательно, всегда можно получить конъюнкцию: $[f_L, 0, 1, \bar{x}] \supseteq [x \cdot y, \bar{x} \cdot \bar{y}]$.

Конъюнкция получена.

В результате получено, что $[f_0, f_1, f_L, f_S, f_M] \supseteq [0, 1, \bar{x}, x \cdot y] = P_2$. Что и требовалось доказать. ■

[Презентации к госам от маткиба]

0.0.34 DOP 3 Логика 1-го порядка. Выполнимость и общезначимость. Общая схема метода резолюций.

Базовые символы:

- Предметные переменные $Var = \{x_1, x_2, \dots, x_k, \dots\}$;
- Предметные константы $Const = \{c_1, c_2, \dots, c_l, \dots\}$;
- Функциональные символы $Func = \{f_1^{n_1}, f_2^{n_2}, \dots, f_r^{n_r}, \dots\}$;
- Предикатные символы $Pred = \{P_1^{m_1}, P_2^{m_2}, \dots, P_s^{m_s}, \dots\}$.

Тройка $(Const, Pred, Func)$ называется сигнатурой алфавита.

Логические связи и кванторы:

Конъюнкция \wedge

Дизъюнкция \vee

Отрицание \neg

Импликация \rightarrow

Квантор всеобщности \forall

Квантор существования \exists

Определение терма: Терм — это

x , если $x \in Var$, x — переменная;

c , если $c \in Const$, c — константа;

$f^n(t_1, t_2, \dots, t_n)$, если $f^n \in Func$, t_1, t_2, \dots, t_n — термы, — составной терм.

Терм — множество термов заданного алфавита. Var_T — множество переменных, входящих в состав терма t .

t_{Var} — запись обозначающая терм t , у которого $Var_t \subseteq \{x_1, x_2, \dots, x_n\}$.

Формула — это либо атомарная формула $P_m(t_1, t_2, \dots, t_m)$, если $P_m \in Pred$, $\{t_1, t_2, \dots, t_m\} \subseteq Term$; либо составная формула $\phi, \psi \vee \phi$ и т.д., если ψ, ϕ — формулы;

Квантор связывает ту переменную, которая следует за ним. Вхождение переменной в области действия квантора, связывающего эту переменную, называется связанным. Вхождение переменной в формулу, не являющейся связанным, называется свободным. Переменная называется свободной, если она имеет свободное вхождение в формулу. $\psi(x_1, x_2, \dots, x_n)$ — запись, обозначающая формулу ψ , у которой $Var_\psi \supseteq \{x_1, x_2, \dots, x_n\}$. Если $Var_\psi = 0$, то формула ψ называется замкнутой формулой, или предложением. $CForm$ — множество всех замкнутых формул.

• Интерпретация сигнатуры — это $(D_1, Const, Func, Pred)$, где

• D_1 — непустое множество, которое называется областью интерпретации, предметной областью, универсумом.

• $Const$ — оценка констант — сопоставляет каждой константе предмет из области интерпретации.

• $Func$ — оценка функциональных символов — сопоставляет n -местной функции f функцию из области интерпретации.

• $Pred$ — оценка предикатных символов — сопоставляет каждому предикатному символу отношение на области интерпретации.

Отношение выполнимости обозначается как \models . Пусть I — интерпретация.

$I \models \varphi(x_1, \dots, x_n)[d_1, \dots, d_n]$, т.е. формула выполнима в интерпретации I на наборе d_1, \dots, d_n , если

1. $\varphi(x_1, \dots, x_n) = P(t_1, \dots, t_m) \wedge \bar{P}(t_1[d_1, \dots, d_n], \dots, t_n[d_1, \dots, d_n]) = True$,

2. φ имеет вид $\psi_1 \wedge \psi_2$ и обе формулы выполнимы на наборе,

3. φ имеет вид $\psi_1 \vee \psi_2$ и хотя бы одна из формул выполнима на наборе,

4. φ имеет вид $\psi_1 \rightarrow \psi_2$ и на наборе либо выполнима ψ_2 , либо невыполнима ψ_1 ,

5. $\neg \varphi$ невыполнима на наборе,

6. если $\varphi(x_1, \dots, x_n) = \exists x_0 \psi(x_0, x_1, \dots, x_n)$ и для некоторого элемента d_0 , $d_0 \in D_1$ имеет место $I \models \psi(x_0, \dots, x_n)[d_1, \dots, d_n]$,

7. если $\varphi(x_1, \dots, x_n) = \forall x_0 \psi(x_0, x_1, \dots, x_n)$ и для любого элемента d_0 , $d_0 \in D_1$ имеет место $I \models \psi(x_0, \dots, x_n)[d_1, \dots, d_n]$.

Формула выполнима в интерпретации, если существует хотя бы один набор элементов интерпретации, на котором формула выполнима.

Формула истинна в интерпретации, если она выполнима на любом наборе элементов интерпретации.

Формула выполнима, если существует интерпретация, в которой она выполнима.

Формула общезначима, если она истинна в любой интерпретации.

Формула без свободных переменных называется замкнутой формулой (предложением).

Пусть Γ — некоторое множество замкнутых формул, $\Gamma \subseteq CForm$. Тогда каждая интерпретация I , в которой выполняются все формулы множества Γ , называется моделью для множества Γ .

Пусть Γ — некоторое множество замкнутых формул, и ψ — замкнутая формула. Формула ψ называется логическим следствием Γ , если каждая модель для множества предложений (базы знаний) Γ , если каждая модель для множества формул Γ является моделью для формулы ψ , т. е. для любой интерпретации I верно $I \models \Gamma \iff I \models \psi$.

Запись $\Gamma \models \psi$ обозначает, что ψ — логическое следствие Γ .

Теорема о логическом следствии. Пусть $\Gamma = \{\psi_1, \dots, \psi_n\} \subseteq CForm$, $\phi \in CForm$. Тогда $\Gamma \models \phi \iff \models \psi_1 \wedge \dots \wedge \psi_n \rightarrow \phi$.

▲ (\Rightarrow) Пусть I — произвольная интерпретация. Если $I \not\models \psi_1 \wedge \dots \wedge \psi_n$, то $I \models \psi_1 \wedge \dots \wedge \psi_n \rightarrow \phi$.

0.0.37 DOP 8 Зависимости в реляционных отношениях: функциональные, многозначные, проекции/соединения. Проектирование реляционных БД на основе принципов нормализации отношений. Нормальные формы.

Зависимости в реляционных отношениях

задана переменная отношения R ($-$ таблица), и X и Y являются произвольными подмножествами заголовка R ($<$ составными> атрибутами, наборами столбцов).

Атрибут Y **функционально зависит** от атрибута $X \iff$ каждому значению X соответствует в точности одно значение Y . В этом случае говорят также, что атрибут X **функционально определяет** атрибут Y (X является дeterminантом (определителем) для Y , а Y является зависимым от X). (далее, FD (Functional Dependency) – функциональная зависимость).

$FD A \Rightarrow B$ – **тривиальная**, если $A \supseteq B$. Любая тривиальная FD всегда выполняется.

Армстронг предложил правила вывода новых FD на основе существующих. Аксиомы Армстронга:

1. Если $A \supseteq B$, то $A \Rightarrow B$ (рефлексивность).
2. Если $A \Rightarrow B$, то $(A \cup C) \Rightarrow (B \cup C)$ (полонение).
3. Если $A \Rightarrow B$ и $B \Rightarrow C$, то $A \Rightarrow C$ (транзитивность).

В переменной отношении R с атрибутами A, B, C (в общем случае, составными) имеется **многозначная зависимость** B от A ($A \Rightarrow B$) \iff множество значений атрибута B , соответствующее паре значений атрибутов A и C , зависит от значения A и не зависит от значения C .

Декомпозиция отношения R называется замена R на совокупность отношений $\{R_1, R_2, \dots, R_n\}$ такую, что каждый из них есть проекция R , и каждый атрибут R входит хотя бы в одну из проекций декомпозиции. То есть все R_i состоят только из атрибутов R и любой атрибут R есть хотя бы в одном R_i .

$\exists R' = NATURAL JOIN (R_1, \dots, R_n)$. Декомпозиция $\{R_1, R_2, \dots, R_n\}$ называется **декомпозицией без потерь**, если $R' = R$

\exists некоторое отношение r со скемой R , а также два произвольных подмножества атрибутов $A, B \subseteq R$. $\exists C = R \setminus (A \cup B)$. В этом случае B **многозначно зависит** от A , тогда и только тогда, когда множество значений атрибута B , соответствующее заданной паре $[a : A; c : C]$ отношения r , зависит от a и не зависит от c .

$\exists R$ – переменная отношения, а A, B, \dots, Z – некоторые подмножества множества её атрибутов. Если декомпозиция любого допустимого значения R на отношения, состоящие из множества атрибутов A, B, \dots, Z , является декомпозицией без потерь, говорят, что переменная отношение R удовлетворяет зависимости соединения $*\{A, B, \dots, Z\}$.

Проектирование реляционных БД

Теорема Хита. \exists Пусть $\{A, B, C\}$ – отношение, состоящее из множества атрибутов A и C . Если в R есть функциональная зависимость $A \Rightarrow B$, то R равно соединению его проекций $\{A, B\}$ $\{A, C\}$.

Нормальные формы

Переменная отношения находится в 1НФ тогда и только тогда, когда в любом допустимом значении отношения каждого его кортежей содержит только одно значение для каждого из атрибутов. Реляционное отношение уже находится в 1НФ.

Замыкание множества FD S – множество FD S^+ , включающее все FD, логически выводимые из множества S .

FD с минимальным детерминантом (удаление любого атрибута из детерминанты приводит к изменению замыкания S^+ , т. е. порождению множества FD, неэквивалентного S) называется **минимальной связью**.

Атрибут B в **минимально зависит** от атрибута A , если выполняется минимальная слева $A \Rightarrow B$.

Потенциальный ключ – в реляционной модели данных – подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости).

• Уникальность означает, что нет и не может быть двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают (равны).

Свойство уникальности определяется не для конкретного значения переменной отношения в тот или иной момент времени, а по всем возможным значениям, то есть следует из внешнего знания о природе и закономерностях данных, которые могут находиться в переменной отношении.

• Минимальность (несократимость) означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности. Иными словами, если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности.

Первичный ключ – в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию).

Суперключ отношения r – любое подмножество K заголовка r , включающее, по меньшей мере, хотя бы один возможный ключ r .

FD $A \Rightarrow C$ называется **транзитивной**, если существует такой атрибут B , что имеются $F3$ и $A \Rightarrow B$ и $B \Rightarrow C$ и отсутствует $F3 \Rightarrow C$.

Переменная отношения находится в 2НФ тогда и только тогда, когда она находится в 1НФ, и каждый неключевой атрибут минимально функционально зависит от первичного ключа.

Данные находятся в 1НФ но не в 2НФ:

| PK: Модель | PK: Фирма | Цена | Скидка |
|------------|-----------|-------|--------|
| M5 | BMW | 50k\$ | 5% |
| X5M | BMW | 51k\$ | 5% |
| GT-R | Nissan | 47k\$ | 10% |

Цена машины зависит от модели и фирмы. Скидка зависит от фирмы, то есть зависимость от первичного ключа неполна. Исправляется это путем декомпозиции на два отношения, в которых не ключевые атрибуты зависят от ПК.

| PK: Модель | PK: Фирма | Цена | PK: Модель | PK: Фирма | Цена |
|------------|-----------|-------|------------|-----------|------|
| M5 | BMW | 50k\$ | | | |
| X5M | BMW | 51k\$ | | | |
| GT-R | Nissan | 47k\$ | | | |

Переменная отношения находится в 3НФ тогда и только тогда, когда она находится в 2НФ и каждый неключевой атрибут не транзитивно функционально зависит от первичного ключа.

Данные находятся в 2НФ но не в 3НФ:

| PK: Модель | Магазин | Телефон |
|------------|---------------|----------|
| BMW | Salon Lan-bin | 18-05-00 |
| Audi | Salon Lan-bin | 18-05-00 |
| Nissan | Cherap-avto | 07-04-99 |

В отношении атрибут «Модель» является первичным ключом. Личных телефонов у автомобилей нет, и телефон зависит исключительно от магазина. Таким образом, в отношении существуют следующие функциональные зависимости: Модель \rightarrow Магазин, Магазин \rightarrow Телефон, Модель \rightarrow Телефон. Зависимость Модель \rightarrow Телефон является транзитивной, следовательно, отношение не находится в 3НФ. В результате разделения исходного отношения получаются два отношения, находящиеся в 3НФ:

| PK: Модель | Магазин | Телефон |
|------------|---------------|----------|
| BMW | Salon Lan-bin | 18-05-00 |
| Audi | Salon Lan-bin | 18-05-00 |
| Nissan | Cherap-avto | 07-04-99 |

BCNF, 4НФ Между 3 и 4 нормальной формой есть еще и промежуточная нормальная форма, она называется – **Нормальная форма Бойса-Кодда** (BCNF). Иногда ее еще называют «Усиленная третья нормальная форма».

1. Таблица должна находиться в третьей нормальной форме. Здесь все как обычно, т.е. как и у всех остальных нормальных форм, первое требование заключается в том, чтобы таблица находилась в предыдущей нормальной форме, в данном случае в третьей нормальной форме.

2. Ключевые атрибуты составного ключа не должны зависеть от неключевых атрибутов

Отсюда следует, что требования нормальной формы Бойса-Кодда предъявляются только к таблицам, у которых первичный ключ составной. Таблицы, у которых первичный ключ простой, и они находятся в третьей нормальной форме, автоматически находятся и в нормальной форме Бойса-Кодда.

Требование **четвертой нормальной формы (4NF)** заключается в том, чтобы в таблицах отсутствовали нетривиальные многозначные зависимости и были в 3НФ. В таблицах многозначная зависимость выглядит следующим образом: **Таблица должна иметь как минимум три столбца**, допустим A, B, C , при этом B и C между собой никак не связаны и не зависят друг от друга, но по отдельности зависят от A , и для каждого значения A есть множество значений B , а также множество значений C .

В данном случае многозначная зависимость обозначается вот так: $A \Rightarrow B, A \Rightarrow C$ Если подобная многозначная зависимость есть в таблице, то она не соответствует четвертой нормальной форме.

[IT-сообщество, [Какие-то ссылки про БД](#)]

0.0.37 DOP 8 Зависимости в реляционных отношениях: функциональные, многозначные, проекции/соединения. Проектирование реляционных БД на основе принципов нормализации отношений. Нормальные формы.

Зависимости в реляционных отношениях

задана переменная отношения R ($-$ таблица), и X и Y являются произвольными подмножествами заголовка R ($<$ составными> атрибутами, наборами столбцов).

Атрибут Y **функционально зависит** от атрибута $X \iff$ каждому значению X соответствует в точности одно значение Y . В этом случае говорят также, что атрибут X **функционально определяет** атрибут Y (X является дeterminантом (определителем) для Y , а Y является зависимым от X). (далее, FD (Functional Dependency) – функциональная зависимость).

$FD A \Rightarrow B$ – **тривиальная**, если $A \supseteq B$. Любая тривиальная FD всегда выполняется.

Армстронг предложил правила вывода новых FD на основе существующих. Аксиомы Армстронга:

1. Если $A \supseteq B$, то $A \Rightarrow B$ (рефлексивность).
2. Если $A \Rightarrow B$, то $(A \cup C) \Rightarrow (B \cup C)$ (полонение).
3. Если $A \Rightarrow B$ и $B \Rightarrow C$, то $A \Rightarrow C$ (транзитивность).

В переменной отношении R с атрибутами A, B, C (в общем случае, составными) имеется **многозначная зависимость** B от A ($A \Rightarrow B$) \iff множество значений атрибута B , соответствующее паре значений атрибутов A и C , зависит от значения A и не зависит от значения C .

Декомпозиция отношения R называется замена R на совокупность отношений $\{R_1, R_2, \dots, R_n\}$ такую, что каждый из них есть проекция R , и каждый атрибут R входит хотя бы в одну из проекций декомпозиции. То есть все R_i состоят только из атрибутов R и любой атрибут R есть хотя бы в одном R_i .

$\exists R' = NATURAL JOIN (R_1, \dots, R_n)$. Декомпозиция $\{R_1, R_2, \dots, R_n\}$ называется **декомпозицией без потерь**, если $R' = R$

\exists некоторое отношение r со скемой R , а также два произвольных подмножества атрибутов $A, B \subseteq R$. $\exists C = R \setminus (A \cup B)$. В этом случае B **многозначно зависит** от A , тогда и только тогда, когда множество значений атрибута B , соответствующее заданной паре $[a : A; c : C]$ отношения r , зависит от a и не зависит от c .

$\exists R$ – переменная отношения, а A, B, \dots, Z – некоторые подмножества множества её атрибутов. Если декомпозиция любого допустимого значения R на отношения, состоящие из множества атрибутов A, B, \dots, Z , является декомпозицией без потерь, говорят, что переменная отношение R удовлетворяет зависимости соединения $*\{A, B, \dots, Z\}$.

Проектирование реляционных БД

Теорема Хита. \exists Пусть $\{A, B, C\}$ – отношение, состоящее из множества атрибутов A, B, C . Если в R есть функциональная зависимость $A \Rightarrow B$, то R равно соединению его проекций $\{A, B\}$ $\{A, C\}$.

Нормальные формы

Переменная отношения находится в 1НФ тогда и только тогда, когда в любом допустимом значении отношения каждого его кортежей содержит только одно значение для каждого из атрибутов. Реляционное отношение уже находится в 1НФ.

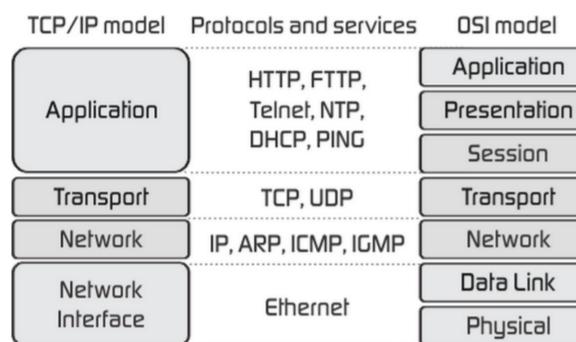
Замыкание множества FD S – множество FD S^+ , включающее все FD, логически выводимые из множества S .

FD с минимальным детерминантом (удаление любого атрибута из детерминанты приводит к изменению замыкания S^+ , т. е. порождению множества FD, неэквивалентного S) называется **минимальной связью**.

0.0.45 DOP 15 Основные принципы построения и архитектура сети Интернет. Алгоритмы и протоколы внешней и внутренней маршрутизации. Явление перегрузки и методы борьбы с ней.

Основные принципы построения сети Интернет

- Децентрализация управления — нет единого центра управления для сети Интернет.
- Выход из строк одного компьютера или участка сети не приводит к неработоспособности всей сети.
- Коммутация пакетов — способ динамического распределения ресурсов сети связи за счёт передачи и коммутации оцифрованной информации в виде частей небольшого размера, так называемых **пакетов**, которые передаются по сети, в общем случае, независимо друг от друга (действиями) либо последовательно друг за другом по виртуальным соединениям. Узел-приёмник из пакетов собирает сообщение. В таких сетях по одной физической линии связи могут обмениваться данными много узлов.
- Инкапсуляция — последовательное вложение протокольной единицы (PDU) вышележащего уровня в протокольную единицу нижележащего уровня. PDU вышележащего уровня не доступны на нижележащем уровне (нажелезажий уровень не понимает структуры данных вышележащего уровня).
- Каждый уровень имеет строго определенный интерфейс с нижележащим и вышележащим уровнями и набор сервисов, которые может использовать вышележащий уровень.



ISO/OSI:

Физический — передача последовательности битов через физическую линию.

Канальный — превращение несовершенной физической среды передачи данных в надежный канал, свободный от ошибок передачи.

Сетевой — построение маршрута между отправителем и получателем.

Транспортный — получение данных с вышеуказанным уровнем, разделение на более мелкие единицы (если требуется), передача на сетевой уровень, обеспечение целостности данных при доставке до адресата.

Сессии — установка сессий между пользователями. Сессия позволяет передавать данные, как это может делать транспортный уровень, и имеет более сложный сервис, полезный в некоторых приложениях. Например, можно осуществить вход в удаленную систему.

Представления — обеспечение решения проблем, связанных с представлением данных, в основном — проблемы синтаксиса и семантики передаваемой информации.

Приложений — обеспечение работы часто используемых приложений, например — передача файлов.

TCP/IP:

Прикладной уровень. На прикладном уровне (Application layer) работает большинство сетевых приложений. Эти программы имеют свои собственные протоколы обмена информацией, например, интернет браузер для протокола HTTP, ftp-клиент для протокола FTP (передача файлов).

Транспортный уровень (как упаковывать?). Протоколы транспортного уровня (Transport layer) могут решать проблему негарантированной доставки сообщений («дошли ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных.

Сетевой (межсетевой) уровень (кому отправлять?). Межсетевой уровень (Network layer) изначально разработан для передачи данных из одной сети в другую. На этом уровне работают маршрутизаторы, которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по маске сети.

Канальный уровень (как кодировать в среде?). Канальный уровень (англ. Link layer) описывает способ кодирования данных для передачи пакета данных в физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также обеспечивающие помехоустойчивость).

Алгоритмы и протоколы внешней и внутренней маршрутизации

Алгоритмы маршрутизации применяются для определения наилучшего пути пакетов от источника к приемнику и являются основой протокола маршрутизации. Для формулирования алгоритмов маршрутизации сеть рассматривается как граф. При этом маршрутизаторы являются узлами, а физические линии между маршрутизаторами — ребрами соответствующего графа. Каждой грани графа присваивается определенное число — стоимость, зависящая от физической длины линии, скорости передачи данных по линии или стоимости линии. Основными алгоритмами являются алгоритмы Беллмана-Форда и Дейкстры.

Алгоритм Беллмана-Форда (пример алгоритма по вектору расстояний):

- Пусть каждый маршрутизатор знает стоимость линии к каждому своему непосредственному соседу
- Маршрутизатор R_s рассчитывает стоимость C_i для достижения каждого известного ему R_i
- Вектор $C_8 = (C_1, C_2, \dots, C_7)$ — вектор расстояния до R_s
- Изначально $C = (\infty, \infty, \dots, \infty)$

1. Каждые T секунд R_s шлет C_i всем своим соседям
2. Если R_i нашёл более дешёвый путь, то он обновляет C_i у всех своих соседей
3. Вернуться к 1

- Длина вектора C устанавливает администратор

Алгоритм Дейкстры (пример алгоритма по состоянию канала):

1. Определение топологии сети: каждый маршрутизатор передаёт линии всем своим соседям состояния своих линий и строят топологию сети (1. Периодически, 2 Когда изменяется состояние линии)

2. Вычисление по алгоритму Дейкстры: каждый маршрутизатор независимо запускает алгоритм Дейкстры наискратчайшего пути. Каждый маршрутизатор находит соединяющее дерево с минимальной стоимостью до каждого другого маршрутизатора.

Протоколы внутренней маршрутизации: RIP (на основе алгоритма Б-Ф, редко используется), OSPF (на основе алгоритма Д, широко используется).

Протоколы внешней маршрутизации: BGP-4 — алгоритм разработан для того, чтобы решить проблему того, что Интернет представляет из себя смесь разнообразных автономных сетей. Необходимо учитывать ряд условий, связанных с политикой автономной сети, например, пакеты из Министерства обороны не должны проходить через недружественные страны.

Перегрузка — падение производительности в транспортной среде из-за слишком большого числа пакетов. (Если на маршрутизаторе в очереди скапливается много пакетов, то они просто сбрасываются — это неэффективно)

Управление перегрузками — организация потоков в транспортной среде, при которой потоки соответствуют пропускной способности подсети и не превышают ее.



0.0.46 DOP 13

Качество программного обеспечения и методы его контроля. Тестирование и другие методы верификации.

Основные определения

- **Качество ПО** в стандарте ISO 9126 — вся совокупность его характеристик, относящихся к возможности удовлетворять высказанные или подразумеваемые потребности всех заинтересованных лиц.
- Внутреннее качество связано с характеристиками ПО самого по себе, без учета его поведения; внешнее качество характеризует ПО с точки зрения его поведения; качество ПО при использовании в различных контекстах — качество, которое ощущается пользователями при конкретных сценариях работы ПО.

- **Верификация** означает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.

- **Методы обеспечения качества** представляют собой техники, гармонизирующие достижение определенных показателей качества при их применении.

Методы верификации

- Методы и техники выяснения свойств ПО во время его работы. Это, прежде всего, все виды тестирования.

- Методы и Техники определения качества на основе симуляции работы ПО с помощью моделей: проверка на моделях (model checking), прототипирование (макетирование) для оценки качества принимаемых решений).

- Методы и Техники выявления нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации: инспектирование кода.

- Методы и Техники обычного или формализованного анализа проектной документации и исходного кода для выявления их свойств: методы анализа архитектуры ПО.

Виды тестирования

- Стressовое (нагрузочное) тестирование — проверяет производительность ПО в условиях повышенных нагрузок.

- Тестирование черного ящика (тестирование на соответствие) — проверка требований спецификаций, стандартов, ограничений.

- Функциональное тестирование — его частный случай, проверка требований к функциональности.

- Аттестационное тестирование — для получения документа о соответствии.

- Тестирование белого ящика — на основе знаний о структуре системы.

- Тестирование на отказ — попытка вывести систему из строя в том числе некорректными данными.

- Тестирование с помощью набора мутантов — программ, отличных от тестируемой в отдельных точках.

- Модульное тестирование — проверка отдельных модулей. Важная часть отладочного тестирования. **Предусловия** описывают для каждой операции, на каких входных данных она предназначена работать, **постусловия** — как должны соотноситься входные данные с возвращающими ими результатами, **инварианты** — определяют критерии целостности внутренних данных модуля.

- Интеграционное тестирование — проверка правильности взаимодействия некоторого набора модулей друг с другом.

- Системное тестирование — проверка правильности работы системы в целом, ее способности правильно решать поставленные пользователем задачи в различных ситуациях.

- Тестирование пользовательского интерфейса — имитация действий пользователя над элементами этого интерфейса. Частные случаи — тестирование графического интерфейса, тестирование интерфейса Web-приложений.

[Пушкин-Залупкин, Напиши источник!, page 69-96]

0.0.47 DOP 11 Функции FIRST и FOLLOW. LL(1)-грамматики. Конструирование таблицы предсказывающего анализатора.

LL(1)-анализатор — наиходящий алгоритм синтаксического разбора. Цифра 1 говорит, что для определения пути разбора нужна всего одна лексема.

LL(1) Используется для разбора кода в ряде языков программирования, таких, как Pascal и Python (до 3.8). Очень быстр в исполнении и имеет характерное сообщение об ошибке вида «ожидался такой-то символ».

Определения

- При построении таблицы предсказывающего анализатора нам потребуются две функции FIRST и FOLLOW.

- Пусть $G = (N, T, P, S)$ — КС-грамматика. Для α — произвольной цепочки, состоящей из символов грамматики, определим $FIRST(\alpha)$ как множество терминалов, с которых начинаются строки, выводимые из α . Если $\alpha \Rightarrow^* e$, то e также принадлежит $FIRST(\alpha)$.

- Определим $FOLLOW(A)$ для нетерминала A как множество терминалов a , которые могут появиться непосредственно справа от A в некоторой синтаксической форме грамматики, то есть множество терминалов a таких, что существует вывод вида $S \Rightarrow^* \alpha A a \beta$ для некоторых $\alpha, \beta \in (N \cup T)^*$. Заметим, что между α и a в процессе вывода могут находиться нетерминальные символы, из которых выводится e . Если A может быть самым правым символом некоторой синтаксической формы, то e также принадлежит $FOLLOW(A)$.

- Грамматика $G = (N, T, P, S)$ является LL(1)-грамматикой тогда и только тогда, когда для каждой пары правил $A \rightarrow \alpha, A \rightarrow \beta$ из P (то есть правил с одинаковой левой частью) выполняются следующие 2 условия:

1. $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$.
2. Если $e \in FIRST(\alpha)$, то $FIRST(\beta) \cap FOLLOW(A) = \emptyset$.

Вычисление FIRST для символов КС грамматики

Вход: КС грамматика $G = (N, T, P, S)$.

Выход: Множество $FIRST(X)$ для каждого символа $X \in (N \cup T)$.

1. Если X — терминал, то положить $FIRST(X) = \{X\}$, если нетерминал — $FIRST(X) = \emptyset$.
2. Если в P есть правило $X \rightarrow e$, то добавить в $FIRST(X)$ e .
3. Выполнять алгоритм на рисунке ниже.

Краткое описание: для каждого правила $X \leftarrow Y_1 Y_2 \dots$ в $FIRST(X)$ добавлять $FIRST(Y_1)$ и, если $e \in FIRST(Y_1)$, добавлять $FIRST(Y_2)$ и, если $e \in \dots$

do { continue = false;

 Для каждого нетерминала X

 Для каждого правила $X \rightarrow Y_1 Y_2 \dots Y_k$

i=1; nonstop = true;

while (i < k && nonstop)

{добавить FIRST(Y_i) n {e} к FIRST(X);

if (Были добавлены новые элементы)

continue = true;

if (e != FIRST(Y_i)) nonstop = false;

else i++ = 1;

}

if (nonstop) {добавить e к FIRST(X);

continue = true;

while (continue);

Вычисление FIRST для цепочки

Вход: КС грамматика $G = (N, T, P, S)$.

Выход: Множество $FIRST(X_1, X_2, \dots, X_n)$, $X_i \in (N \cup T)$.

1. Для всех $X \in (N \cup T)$ вычислить $FIRST(X)$.

2. Положить $FIRST(X_1, X_2, \dots, X_n) = \emptyset$.

| |
|--|
| <p>0.0.49 DOP 16 Теоретические основы передачи данных, физический уровень стека протоколов. Системы передачи данных Ethernet и Wi-Fi: алгоритмы работы, управление множественным доступом к каналу</p> <p>Теоретические основы</p> <p>Данные — описание фактов, явлений.</p> <p>Сигнал — представление данных при передаче.</p> <p>Передача — процесс взаимодействия передатчика и приёмника, с целью передачи сигнала. Данные и сигналы могут быть аналоговыми (непрерывными) и цифровыми. Для цифровой передачи данных сигнал нужно разбивать на уровни (для кодирования).</p> <p>Полосы пропускания канала — спектр частот, которые канал пропускает без существенного понижения мощности сигнала.</p> <p>Скорость передачи зависит от способа кодирования данных на физическом уровне и сигнальной скорости — скорости изменения значения сигнала.</p> <p>Пропускная способность канала — максимальная скорость, с которой канал способен передавать данные.</p> <p>Теорема Найквиста-Котельникова: $R_{max_data_rate} = 2 * D * \log_2 L$ (bps (bits per second))</p> <ul style="list-style-type: none"> - $R_{max_data_rate}$ — максимальная пропускная способность канала - D — ширина полосы пропускания канала (максимальная частота сигнала в спектре). - L — количество уровней (значений) сигнала. <p>Теорема Котельникова — Аналоговый сигнал $u(t)$, не содержащий частот выше $F_{max}(\Gamma_U)$, полностью определяется последовательностью своих значений в моменты времени, отстоящие друг от друга на $\frac{1}{2F_{max}}$</p> <p>Шум в канале измеряется, как соотношение мощности полезного сигнала к мощности шума: S/N (измеряется в децибелах $1dB = 10 \log_{10} S/N$)</p> <p>Т</p> |
|--|

0.0.53 DOP 23 Промежуточные представления программы: абстрактное синтаксическое дерево; последовательность трехадресных инструкций. Базовые блоки и граф потока управления.

Промежуточное представление программы (Intermediate Representation, IR) — форма представления программы, ориентированная на удобство дальнейшей обработки компилятором. Различают следующие IR:

- HIR (высокий уровень) — абстрактное синтаксическое дерево (АСД) — конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья — с соответствующими операндами. По сути результат парсинга программы на языке программирования, по АСД можно однозначно восстановить программу, по остальным уже нельзя.

- MIR (средний уровень) — включает в себя:

1. Инструкции
 - присваивания: $x \leftarrow op y z$; $x \leftarrow op y$; $x \leftarrow y$; $x \leftarrow y[i]$;
 - переходы: $goto L$; $ifTrue x goto L$; $ifFalse x goto L$;
 - дополнительное: $param x$; $call x n$; $return y$;
2. таблицу символов,
- переменные, их имена в программе и атрибуты, такие как область видимости, тип, для имен функций - число параметров, и т. п.

- LIR (низкий уровень) — фактически машинные инструкции — используется для машинно-зависимых задач, например, распределение регистров и выбор подходящих команд.

Базовые блоки и граф потока управления

Базовым блоком (ББ или линейным участком) называется последовательность следующих одна за другой инструкций MIR, обладающая следующими свойствами:

- Поток управления может входить в базовый блок только через его первую инструкцию (т.е. в программе нет переходов в середину базового блока).

- Поток управления покидает базовый блок без останова или ветвления, кроме, возможно, последней инструкции базового блока.

Грубо говоря, базовый блок содержит инструкции, которые будут выполнены (или не выполнены) обязательно все вместе, независимо ни от чего. Поэтому он базовый.

Чтобы выделить базовые блоки, достаточно найти все их начала (НББ):

- первая инструкция программы
- инструкция, на которой есть метка
- следующая инструкция после перехода

Граф потока управления (ГПУ) — граф, обладающий следующими свойствами:

- Вершины — базовые блоки.

- Дуга соединяет выход одного блока со входом другого, если второй блок может выполняться сразу следом за первым.

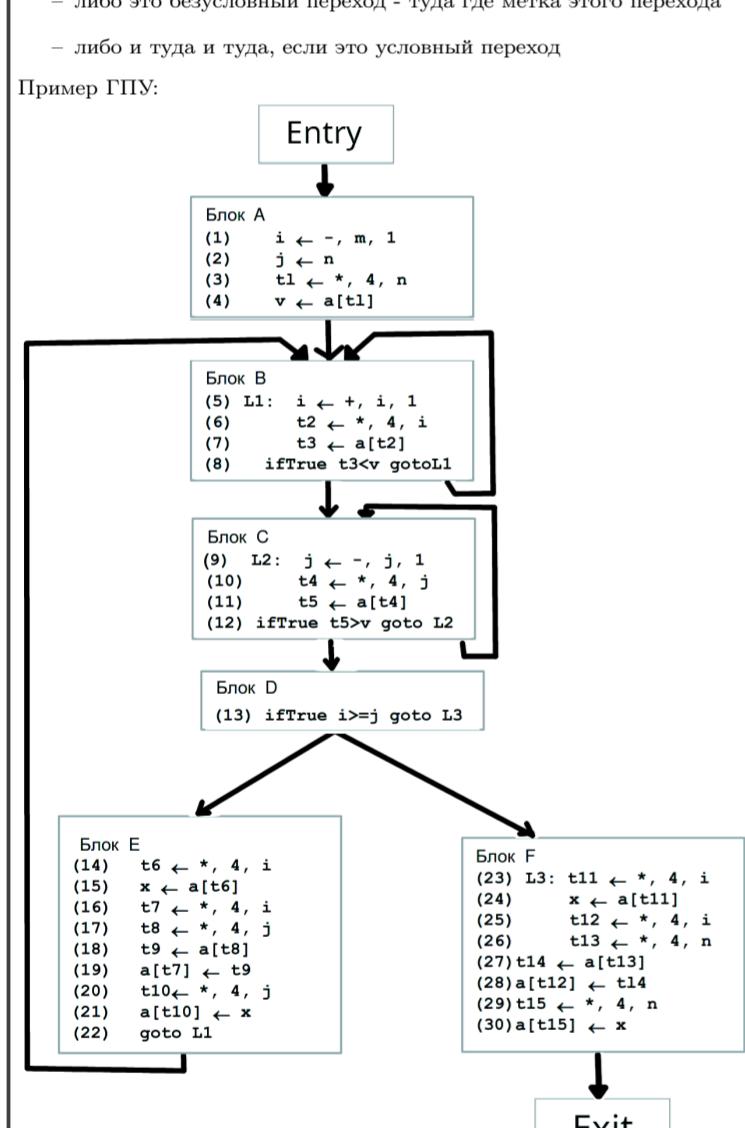
Если последняя инструкция базового блока — условный переход, то из этого блока будут выходить две дуги.

Если первая инструкция базового блока имеет метку, то в этот блок будет входить дуги из всех базовых блоков, у которых последняя инструкция — переход на эту метку.

Чтобы построить ГПУ, надо:

1. выделить базовые блоки;
2. пропустить дугу из блока туда, куда может пойти управление после этого блока. Определяется по последней инструкции:
 - либо просто следующий блок
 - либо это безусловный переход - туда где метка этого перехода
 - либо и туда и туда, если это условный переход

Пример ГПУ:



[Презентации Гайсааряна, slide 7-28]

0.0.54 DOP 21 Отказоустойчивость в распределенных системах. Типы отказов. Фиксация контролльных точек и восстановление после отказа. Репликация и протоколы голосования. Надежная групповая рассылка.

Основные определения

- **Отказом системы** называется поведение системы, не удовлетворяющее ее спецификации. Отказы могут быть случайными, периодическими или постоянными. Случайные отказы (сбои) при повторении операции исчезают. Отказы по характеру своего проявления: <визитские> (система активна, но некорректно работает), 2) <пропажа признаков жизни> (частичная или полная).

Восстановление:

1. Прямое — основано на своевременном обнаружении сбоя и ликвидации его последствий путем приведения некорректного состояния системы в корректное. Такое восстановление возможно только для определенного набора заранее предусмотренных сбоев.
2. Возвратное — возврат процесса (или системы) из некорректного состояния в некоторое из предшествующих корректных состояний.



На рисунке показаны три процесса (X,Y,Z), взаимодействующие через сообщения. Вертикальные черточки показывают на временной оси моменты запоминания состояния процесса для восстановления в случае отказа. Стрелочки соответствуют сообщениям и показывают моменты их отправления и получения. Предположим теперь, что процесс Z сломается и будет восстановлен в состоянии z2. Это приведет к откату процесса Y в у1, а затем и процессов X и Z в начальных состояниях x1 и у1. Этот эффект известен как эффект домино.

- Множество контролльных точек называется **строго консистентным**, если во время его фиксации никаких обменов между процессами не было. Оно соответствует понятию строго консистентного глобального состояния, когда все посланные сообщения получены и нет никаких сообщений в каналах связи.
- Множество контролльных точек называется **консистентным**, если для любой зафиксированной операции приема сообщения, соответствующая операция посылки также зафиксирована (нет сообщений-сирот).

Методы фиксации контролльных точек

1. **Простой метод** — фиксация локальной контролльной точки после каждой операции посыпки сообщения. При этом посыпка сообщения и фиксация должны быть единой неделимой операцией (транзакцией). Множество последних локальных контролльных точек состояния, когда все посланные сообщения получены и нет никаких сообщений в каналах связи.
2. **Лямбда-выражения** (3 строчки) — более общая конструкция, чем лямбда-операторы, могут быть преобразованы в стандартный тип деревьев выражений. Параметры анонимных делегатов типизированы (тип возвращаемого значения выводится из типа выражения в return). А лямбда-конструкции — нетипизированы.

Java

0.0.55 DOP 19 Основные характеристики функциональных языков программирования. Использование понятий функционального программирования (замыкания, анонимные функции) в современных объектно-ориентированных языках.

Свойства функциональных ЯП:

1. Язык динамический — связывания происходят во время выполнения.
2. Нет понятия состояния и присваивания.
3. Главная операция — вызов функции.
4. Главная абстракция — определение функции.
5. Функции — объекты 1 класса, то есть могут быть значениями, вычисляться, передаваться как параметры и возвращаемые значения и т.п.
6. Структуры данных — списки (последовательности).
7. Простая типовая структура.
8. Понятие переменной соответствует математическому смыслу — переменная отождествляется со значением, а не хранит его.

Понятия функционального программирования

- Замыкание — это конструкция, которая связывает функцию (функциональное значение) с переменными из объемлющей области видимости. Про такие переменные говорят, что они "захвачены", их область видимости (scope) не совпадает с областью действия (extent), последняя шире. Пример:

```
function initAdder(x) {
  function adder(y) {return x + y}
  return adder
}
```

- Анонимная функция (лямбда-функция) — это "чистое" функциональное значение без имени. Его можно передавать как параметр другой функции, возвращая как результат другой функции, в языках с процедурными конструкциями — присваивать.

Использование понятий ФП в современных ОО языках C#

```
delegate(int x, int y) {return x+y;}
(x,y)=> {return x+y;}
(x,y)=>x+y;
```

Лямбда-выражения (3 строчки) — более общая конструкция, чем лямбда-операторы, могут быть преобразованы в стандартный тип деревьев выражений. Параметры анонимных делегатов типизированы (тип возвращаемого значения выводится из типа выражения в return). А лямбда-конструкции — нетипизированы.

Пример замыкания:

```
Function<Integer, Integer> initAdder(int x) {
  return (Integer y) -> x + y;
}
```

Пример на Python:

```
square = lambda n: n * n # lambda expression
print(square(4)) # 16
```

[Головин, Материалы по языкам программирования к госэкзамену]

0.0.56 DOP 17 Базисные типы данных в языках программирования. Основные проблемы, связанные с базисными типами и способы их решения в различных языках. Понятие абстрактного типа данных и способы его реализации в современных языках программирования.

Простые типы данных и их свойства

Целевые типы:

- Универсальность (насколько полно учтены машинные типы).
- Наличие (или отсутствие) беззнаковых типов (в Java их нет, в C++, C# есть).
- Представление (размер значения, диапазоны значений).
- Надежность (какие ошибки могут возникать при выполнении операций с целыми значениями — переполнение типа)
- Набор операций (почти во всех языках одинаково, в Java нет беззнакового типа, поэтому есть логический сдвиг).

Вещественные типы:

- $(-1)^S * M * 2^P$, где S — бит знака, M — нормализованнаяmantissa ($0.5 \leq M < 1$), p — порядок.
- Неточные (например, float — точность сложения 2^{-23} , если операнды между 0.5 и 1.

Символьные типы:

- Включают в себя как символы алфавитов естественных языков, так и символы, управляющие работой устройств ввода/вывода, и специальные символы.

• Главной проблемой символьного типа является выбор кодировки. Современное решение — Unicode.

Порядковые типы:

- Перечислимые типы — перечень именованных значений констант, типы диапазона.

• Перечислимый тип C#: числовые значения констант — всегда int, преобразования enum в int — неявно, int в enum — только явно, константы перечислимого типа имеют ту же область действия, что и имя перечислимого типа.

• Перечислимый тип C# — константы типа доступны через <имя типа>.<имя константы>, только явные преобразования между enum и int.

• Java: аналогично C# и являются классами.

• Тип диапазона позволяет ограничить значения целого типа. Есть в Паскале, в C++, C#, Java — нет.

Указательные и ссылочные типы данных:

- Указатель — абстракция понятия машинного адреса, лишенная недостатков указателя. Ссылки C# и Java отличаются от ссылок C++ — тем, что ссылка C++ "навсегда", то есть в течение всего времени жизни ссылки, полностью ассоциирована с объектом. Однако, и ссылки в C++, и ссылки в C# и Java похожи в том смысле, что после установления ассоциации с объектом ссылка идентична самому объекту, поэтому не требуется никакая операция размежевания.

Составные типы и их свойства

1. Одномерные массивы.

Массив — это непрерывная последовательность элементов одного типа. Атрибуты массива: базовый тип (T), тип индекса (I), диапазон индекса (L и R — нижня и верхняя граница) и связанная с ним характеристика: длина массива. В C# и Java массивы — полноправные объекты.

2. Многомерные массивы.

В большинстве языков рассматриваются как массивы массивов. В Java все многомерные массивы — ступенчатые (то есть внутренние массивы не обязаны иметь одну длину), в C# есть примоугольный массив (для более эффективного доступа к элементам и возможности обрабатывать массивы, совместимые с моделью данных языков типа C).

3. Динамические строки.

Последовательность символов произвольной длины. Необходимо введение специального типа вместо массива: строки реализуются как неизменяемый объект (главный аргумент), набор операций для строк существенно шире и специфичнее набора операций для обычных массивов.

4. Записи (структурные типы) — это совокупность объявлений переменных, которые объединены в отдельный объект.

Абстрактные типы данных

Абстрактный тип данных (ATD) — тип, в котором внутренняя структура данных полностью инкапсулирована, то есть тип представлен только множеством операций. Класс является абстрактным типом данных, если открытые членами являются только методы.

Говорят, что совокупность открытых членов класса составляет интерфейс класса.

Реализация:

- Скрытие членов-данных, доступ через селекторы (геттеры-сеттеры) или их абстракцию — свойство (в C#).

• Абстрактный класс — класс, который предназначен исключительно для того, чтобы быть базовым классом.

• Интерфейс — класс, состоящий только из абстрактных методов.

[Головин, Материалы по языкам программирования к госэкзамену]

[Курс распределенных систем]

0.0.57 DOP 24 Локальная оптимизация при компиляции программы. Ориентированный ациклический граф и метод нумерации значений.

Базовый блок (ББ или линейным участком) называется последовательность следующих одна из другой инструкций MIR, обладающая следующими свойствами:

- Поток управления может входить в базовый блок только через его первую инструкцию (т.е. в программе нет переходов в середину базового блока).

• Поток управления покидает базовый блок без останова или ветвления, кроме, возможно, последней инструкции базового блока.

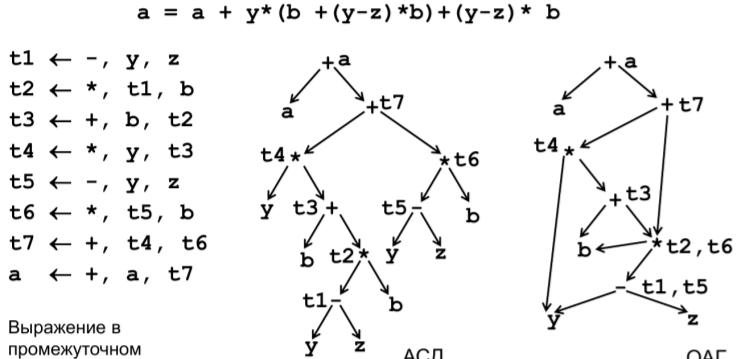
Локальная оптимизация — это оптимизация, которая выполняется в пределах одного базового блока (ББ). Возможные локальные оптимизации:

1. Удаление общих подвыражений (инструкций, повторно вычисляющих уже вычисленные значения).
2. Удаление мертвого кода (инструкций, вычисляющих значения, которые впоследствии не используются).
3. Сворачивание констант (вычисление константных выражений).
4. Изменение порядка инструкций, там, где это возможно, чтобы сократить время хранения временного значения на регистре.
5. Снижение стоимости вычислений (замена более дорогих операций более дешевыми).

ОАГ и метод нумерации значений

Все указанные преобразования для локальной оптимизации можно выполнить за один просмотр ББ, если представить его в виде ориентированного ациклического графа (ОАГ). Суть ОАГ заключается в том, что узлы, представляющие однинаковые значения, присутствуют в единственном экземпляре.

Пример. Выражение в исходном коде:



ОАГ можно представить в виде таблицы значений (вычислять таблицу просто для понимания). Пример таблицы ниже.

Каждая строка таблицы значений представляет один узел ОАГ. Строки содержат:

1. свой номер (номер значения)
2. сигнатуру операции
- Для обычных операций <op, #left, #right>, где op — код операции, a #left и #right — номера значений левого и правого operandов (у unaryных операций #right равен 0)
- Унарные операции id и mm определяют соответственно имена переменных константы (листовые узлы).
3. Имена переменных, в которых хранится это значение.

Алгоритм (на псевдокоде) построения ОАГ для базового блока B, содержащего n инструкций вида $t_i \leftarrow op_i, l_i, r_i$.

Функция `#val(s)` определяет номер значения, определяемого сигнатурой s = (Op, #val(1), #val(r)) .

```
for each "ti ← opi, li, ri" do
    si = (opi, #val(li), #val(ri))
    if (T3 содержит si == si)
        then
            вернуть j в качестве значения #val(si)
        else
            завести в T3 новую строку T3k
            записать сигнатуру si в строку T3k
            вернуть k в качестве значения #val(si)
    t1 ← -, y, z      t15 ← -, y3, z4      t1 ← -, y, z
    t2 ← *, t1, b    t26 ← *, t15, b2    t2 ← *, t1, b
    t3 ← +, b, t2    t37 ← +, b2, t26    t3 ← +, b, t2
    t4 ← *, y, t3    t48 ← *, y3, t37    t4 ← *, y, t3
    t5 ← -, y, z      t59 ← -, y3, z4    t5 ← t1
    t6 ← *, t5, b    t610 ← *, t59, b2    t6 ← t2
    t7 ← +, t4, t6    t711 ← +, t48, t610    t7 ← +, t4, t6
    a ← +, a, t7      a12 ← +, a1, t711    a ← +, a, t7
```

| | 1 | id | ссылка в ТС | a |
|------------|-----|-------------|-------------|---------------------------------|
| 2 | id | ссылка в ТС | b | |
| 3 | id | ссылка в ТС | y | |
| 4 | id | ссылка в ТС | z | |
| 5 | - | 3 | 4 | t1, t5 |
| 6 | * | 5 | 2 | t2, t6 |
| 7 | + | 2 | 6 | t3 |
| 8 | * | 3 | 7 | t4 |
| 9 | + | 8 | 6 | t7 |
| 10 | + | 1 | 9 | a |
| # значения | KOP | # операнда | # операнда | Присоединенные переменные |
| | | | | Определение значения (сигнтура) |

При нумерации значений (и восстановлении базового блока из ОАГ) автоматически получаем **удаление общих подвыражений**. Для значений, которым соответствует несколько переменных достаточно вычислить одну из них, а во вторую скопировать результат.

Сворачивание констант также можно провести во время нумерации значений. Если оба операнда — константы, их результат можно сразу вычислить и записать в таблицу значений как константу.

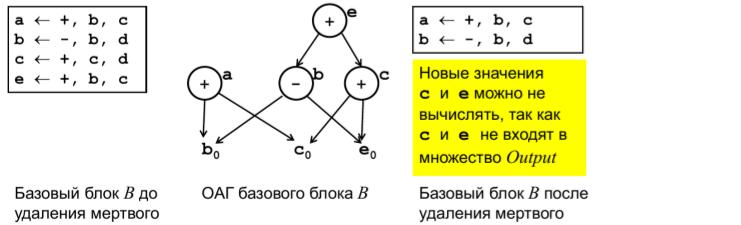
Удаление мертвого кода — более сложная оптимизация, которая требует знаний о других блоках. Для описания алгоритма расширим понятие базового блока

Базовым блоком (ББ) называется тройка ($B = P, Input, Output$), где

- P последовательность инструкций,
- $Input$ — множество переменных, определенных до входа в блок B,
- $Output$ — множество переменных, используемых после выхода из блока B.

Живыми называются переменные, значения которых, вычисленные в рассматриваемом базовом блоке, используются в других базовых блоках.

Пример. Рассмотрим базовый блок $B = \{P, \{a, b, c, d\}, \{a, b\}\}$



Базовый блок B до удаления мертвого кода

ОАГ базового блока B

Базовый блок B после удаления мертвого кода

Восстановление базового блока по его ОАГ

- Для каждого узла с одной или несколькими связанными переменными строится трехадресная инструкция, которая вычисляет значение одной из этих переменных.

- Если у узла несколько присоединенных живых переменных, то следует добавить команды копирования, которые присвоят корректное значение каждой из этих переменных.

[Презентации Гайсаряна, slides 29-45]

0.0.58 DOP 26 Постановка задачи дискретной оптимизации. Метод ветвей и границ. Задача целочисленного линейного программирования.

Постановка задачи дискретной оптимизации: найти $\min_{x \in X} f(x)$, где X — конечно или счетно (ми-во допуст. значений перм. x), в постановке м.б. и нахождение и т.пх

Метод ветвей и границ

Две процедуры: ветвление и нахождение оценок (границ), т.е. для того чтобы МВГ работал, нужны:

- 1) функции оценки решения.

Процедура ветвления состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров.

Процедура можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое **деревом поиска** или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества) множества значений переменной x . Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

В основе МВГ лежит следующая идея: если нижняя граница значений функции на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Обычно минимальную из всех верхних верхних оценок записывают в глобальную переменную m ; любой узел дерева поиска, нижняя граница которого больше значения m , может быть исключен из дальнейшего рассмотрения (закрыт).

Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска меньше или равна m , то A может быть исключена из дальнейшего рассмотрения (закрыта). Если нижняя граница для узла дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (закрыта).

| | | | | | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0.0.61 OSN 24** Линейные методы в машинном обучении: линейная и гребневая регрессии, метод опорных векторов. Регуляризация в линейных методах. **Линейная регрессия** Метод определения целевых значений по формуле $a(X_1, \dots, X_n) = w_0 + w_1 X_1 + \dots + w_n X_n$ называется «линейной регрессией». Здесь X_1, \dots, X_n — значения признаков объекта. Одномерный случай $a(X_1) = w_0 + w_1 X_1$: Предполагая что целевые значения задаются линейно представим следующую систему: $$\begin{cases} w_0 + w_1 x_1 = y_1, \\ \dots \\ w_0 + w_1 x_m = y_m. \end{cases}$$ Но вряд ли система решается точно (она может быть несоставна, особенно при довольно большом объёме обучающей выборки m). Формулы для «невязки» (отклонения/ошибок): $$\begin{cases} e_1 = y_1 - w_0 - w_1 x_1, \\ \dots \\ e_m = y_m - w_0 - w_1 x_m. \end{cases}$$ Задача обучения линейной регрессии — задача минимизации суммы квадратов отклонений: $e_1^2 + \dots + e_m^2 \rightarrow \min$ Или же задача минимизации эмпирического риска по параметрам $w = (w_0, w_1)$: $$L(w) = \sum_{i=1}^m (y_i - a_w(x_i))^2 = \sum_{i=1}^m (y_i - w_0 - w_1 x_i)^2 \rightarrow \min$$ Геометрический смысл — квадраты невязок соответствуют площадям нарисованных розовых квадратов. Явное решение данной задачи: $w_1 = \frac{\sum_{i=1}^m (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^m (x_i - \bar{x})^2} = \frac{\text{cov}(x_i, y_i)}{\text{var}(x_i)}$, $w_0 = \bar{y} - w_1 \bar{x}$, где $\bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$ и $\bar{y} = \frac{1}{m} \sum_{i=1}^m y_i$ Общий случай $a(X_1, \dots, X_n) = w_0 + w_1 X_1 + \dots + w_n X_n$: Здесь $w = (w_0, w_1, \dots, w_n)^T$ — вектор параметров (весов) линейной модели, а $x = (X_0, X_1, \dots, X_n)^T$ — признаковое описание объекта. $X_0 \equiv 1$ — фиктивный признак. Система уравнений: $$\begin{cases} x_1^T w = y_1, \\ \dots \\ x_m^T w = y_m \end{cases}$$ Или же $Xw = y$. Задача оптимизации выглядит так $\|Xw - y\|_2^2 = \sum_{i=1}^m (x_i^T w - y_i)^2 \rightarrow \min_w$ $$\|Xw - y\|_2^2 = (Xw - y)^T (Xw - y) = w^T X^T Xw - w^T X^T y - y^T Xw + y^T y \quad (22)$$ $$\nabla \|Xw - y\|_2^2 = 2X^T Xw - 2X^T y = 0 \quad (23)$$ $$X^T Xw = X^T y \quad (24)$$ $$w = (X^T X)^{-1} X^T y \quad (25)$$ Решение существует, если столбцы матрицы X линейно независимы. Матрица $(X^T X)^{-1} X^T$ называется псевдообратной матрицей Муравьёва. Правило для запоминания: исходное матричное уравнение умножить на X^T (слева и справа) Обобщённая линейная регрессия $a(X_1, \dots, X_n) = w_0 + w_1 \phi_1(X_1, \dots, X_n) + \dots + w_k \phi_k(X_1, \dots, X_n)$: ϕ_1, \dots, ϕ_n — фиксированные базисные функции, не зависят от данных. $$a(x) = \sum_{i=1}^k w_i \phi_i(x) = \phi(X)^T w$$ — решение. $\|\phi(X)^T w - y\|_2^2 \rightarrow \min_w$ — задача оптимизации. **Регуляризация** В $(X^T X)^{-1} X^T$ происходит обращение матрицы которая может оказаться вырожденной. Эту проблему решает регуляризация. Упрощённое объяснение её смысла в линейной модели $a(X_1, \dots, X_n) = w_0 + w_1 X_1 + \dots + w_n X_n$: Если есть два похожих объекта, то должны быть похожи и их метки. Пусть они отличаются в j -м признаке, тогда ответы модели отличаются на $\epsilon_j w_j$. Поэтому не должно быть аномально больших по модулю весов у признаков, по которым могут отличаться похожие объекты, а значит и на всех признаках X_1, \dots, X_n , поскольку заранее не известно, на каких объектах модель будет работать. Заметим также, что константного признака это рассуждение не касается. Поэтому вместе с задачей $\|Xw - y\|_2^2 \rightarrow \min$ обычно стараются решить задачу $\|w\|_2^2 = w_1^2 + \dots + w_n^2 \rightarrow \min$ (минимизация нормы весов). Регуляризация Иванова: $$\begin{cases} \|Xw - y\|_2^2 \rightarrow \min, \\ \|w\|_2^2 \leq \lambda \end{cases}$$ Регуляризация Тихонова (такой вид регуляризации называется также L2-регуляризацией): $$\begin{cases} \|Xw - y\|_2^2 + \lambda \|w\|_2^2 \rightarrow \min, \\ 0 \leq \lambda \end{cases}$$ Они эквивалентны. Решение указанной задачи регуляризации Тихонова задаётся формулой: $$\text{argmin}(\|Xw - y\|_2^2 + \lambda \|w\|_2^2) = (X^T X + \lambda I)^{-1} X^T y \quad (26)$$ Для доказательства достаточно взять градиент и привести к критерию L1-регуляризации: $$\begin{cases} \sum_{i=1}^m (x_i^T w - y_i)^2 + \lambda \sum_{j=1}^n |w_j| \rightarrow \min, \\ 0 \leq \lambda \end{cases}$$ **Гребневая регрессия** Регрессия с коэффициентами, определяемыми формулой 26 называется гребневой регрессией (Ridge Regression). Смысл гребневой регрессии — борьба с вырожденностью (плохой обусловленностью) матрицы $X^T X$. Коэффициент λ называется коэффициентом регуляризации. При $\lambda = 0$ — классическое решение, при $\lambda \rightarrow \infty$ матрица которую приходится обращать становится заведомо хорошо обусловленной, метод меньше «затачивается» на данные». Отметим, что для ridge-регрессии нужна правильная нормировка признаков (как правило, стандартизация), при масштабировании (умножении признаков на скаляры) результат может отличаться. **Градиентный метод обучения** На практике не применяется аналитическое решение. Для оптимизации $\frac{1}{2} \sum_{i=1}^m (a(x_i|w) - y_i)^2 \rightarrow \min$, где $a(x|w) = w^T x$ используют итерационный метод (стохастический градиентный спуск): $$w^{(t+1)} = w^{(t)} - n \nabla L_i(w^{(t)}) \quad (27)$$ $$w^{(t+1)} = w^{(t)} - n(a(x_i|w^{(t)}) - y_i)x_i \quad (28)$$ Здесь n — размер шага (скорость обучения). **Линейные классификаторы** Пусть $X = R^n$ и $Y = \{-1, 1\}$, рассмотрим линейную модель классификатора: $$a(x) = \text{sgn}(w^T x + b) = \begin{cases} +1 & , w^T x + b \geq 0 \\ -1 & , w^T x + b < 0 \end{cases}$$ Геометрический смысл линейного классификатора — пространство делится гиперплоскостью на два полупространства: При обучении минимизируем число ошибок: $$L(X_{t+1:n}, a) = \sum_{i=t+1}^n L(y_i, a(x_i)) \rightarrow \min \quad (29)$$ | **0.0.63 DOP 27** Комбинаторные методы нахождения оптимального пути в графе. **Определения** - Взвешенный орграф $G = (V, E, c)$ называется **сетью** (c — функция, определяющая вес ребра). - Оrientированый маршрут называется **путем**. - Пусть P — некоторый (v, w) -путь: $v = v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k = w$. Тогда $l(P) = c(e_1) + c(e_2) + \dots + c(e_k)$ называется длиной пути P , где e_i — ребро перехода от вершины v_{i-1} к вершине v_i . - (u, w) -путь с наименьшей длиной называется **кратчайшим**. - Задача о кратчайшем пути между фиксированными вершинами: в заданной сети G с двумя выделенными вершинами s и t найти кратчайший (s, t) -путь. **Алгоритм Bellman-Ford'a.** Сложность $\mathcal{O}(|V| + |E|)$. Пусть мы хотим найти длину кратчайшего пути из s в остальные вершины. Обозначим через a_{kp} длину кратчайшего пути из s в p , состоящего из k дуг, или \inf , если такого пути не существует. **Инициализация.** $a_{0s} = 0$, $a_{0p} = \inf$ для всех вершин $p \neq s$. **Шаг алгоритма.** Зная все минимальные длины путей из $k-1$ дуг, посчитать минимальную длину пути из k дуг можно, перебрав все вершины, из которых имеются дуги, идущие в данную. Для всех вершин p : $a_{kp} = \min\{a_{k-1,p} + c((p, p)) \mid p' \in V, (p', p) \in E\}$. Шаг повторяется $|V| - 1$ раз, так как если путь содержит больше дуг, то в нем точно имеется цикл, который можно выбросить. На практике нет необходимости хранить всю матрицу целиком, нужно хранить лишь три строки — ранее вычисленную a_{k-1} , вычисляемую сейчас a_k , и строку с ответами, которая обновляется на каждом шаге: $a_{ksp} = \min\{a_{kp}, a_{ksp}\}$. **K = 0** **K = 1** **K = 2** **K = 3** **Алгоритм Дейкстры.** Сложность варьируется (см. ниже). Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до s . Алгоритм работает пошагово — на каждом шаге он посещает одну вершину и пытается уменьшить метки. Работа алгоритма завершается, когда все вершины посещены. **Инициализация.** Метка самой вершины s полагается равной 0, метки остальных вершин — \inf . Это отражает то, что расстояния от s до других вершин пока неизвестны. Все вершины графа помечаются как неиспользованные. **Шаг алгоритма.** Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещенных вершин выбирается вершина u , имеющая минимальную метку. Обновим метки соседей u и вершин v : $\forall s : (u, s) \in E : \text{label}_s = \min\{\text{label}_s, \text{label}_u + c(u, s)\}$. Пометим вершину u и посещенной. Сложность алгоритма варьируется в зависимости от того как хранить множество неиспользованных вершин для удобства получения вершины с минимальной меткой. При хранении множества как списка, упорядоченного по убыванию меток, сложность алгоритма составляет $\mathcal{O}(|V|^2)$. При использовании двойичной кучи сложность уменьшается до $\mathcal{O}(|E| + |V| \log |V|)$. **Начальное состояние** **Первый шаг** **Второй шаг** **Третий шаг** **Четвертый шаг:** + серенький последний круг [Пупкин-Залупкин, *Напиши источник!*, page 69-96] **Достигающие определения** | Область определения | Множества определений | |----------------------|--| | Направление | Прямое | | Передаточная функция | $\text{gen}_B \cup (x - \text{kill}_B)$ | | Границочное условие | $\text{OUT}[\text{ВХОД}] = \emptyset$ | | Оператор сбора | \wedge | | Уравнения | $\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \wedge_{P \in \text{Pred}_B} \text{OUT}[P]$ | | Инициализация | $\text{OUT}[B] = \emptyset$ | **Глобальная оптимизация** — оптимизация в пределах процедуры (шире чем в базовом блоке). К глобальным оптимизациям относятся, например: - Удаление мертвого кода (вычисляющего неиспользуемые переменные). - Устранение общих подвыражений (одинаковых по тексту выражений, у которых не изменились значения переменных, а значит и результат). - Вынос инвариантов цикла. Для выполнения таких преобразований необходима информация, полученная с помощью **анализа потоков данных**. Он позволяет извлечь различные свойства, вычисленные вдоль путей программы, используя при этом общий алгоритм. Общие понятия для всех анализов: - Точки программы** $(\dots, p_j, p_{j+1}, p_{j+2}, \dots)$ расположены между её инструкциями $(\dots, I_j, I_{j+1}, \dots)$ и характеризуют соответствующие состояния программы. - Состояние программы** — множество значений всех переменных программы, включая переменные в кадрах стека времени выполнения, находящихся ниже текущей вершины стека. - Инструкция программы** I_j описывается парой состояний: состоянием в точке программы p_j перед инструкцией I_j (**входным состоянием**, $\text{In}[I_j]$) и состоянием в точке программы p_{j+1} после инструкции (**выходным состоянием**, $\text{Out}[I_j]$). - Считается, что с каждой инструкцией I_j связаны две **передаточные функции**: передаточная функция прямого обхода ГПУ (от входного до выходного состояния) f_{I_j} и передаточная функция обратного обхода (от выходного до входного состояния) $f_{I_j}^b$. Передаточная функция определяет как изменяется состояние программы, когда встречается эта инструкция. Т.е.: $\text{Out}[I_j] = f_{I_j}(\text{In}[I_j])$ при прямом обходе и $\text{In}[I_j] = f_{I_j}^b(\text{Out}[I_j])$ при обратном. • Для ББ B из инструкций I_1, \dots, I_n по определению $\text{In}[B] = \text{In}[I_1], \text{Out}[B] = \text{Out}[I_n]$. **Передаточная функция** f_B блока B по определению равна композиции передаточных функций его инструкций: $f_B(x) = f_{I_n}(f_{I_{n-1}}(\dots f_{I_1}(x))) = (f_{I_1} \circ f_{I_2} \circ \dots \circ f_{I_n})(x)$, или $f_B = f_{I_1} \circ f_{I_2} \circ \dots \circ f_{I_n}$, и, соответственно, $f_B^b = f_{I_1}^b \circ f_{I_2}^b \circ \dots \circ f_{I_n}^b$. Итого, при прямом обходе $\text{Out}[B] = f_B(\text{In}[B])$; при обратном обходе: $\text{In}[B] = f_B^b(\text{Out}[B])$. (Осторожнее с порядком функций в операции композиции \circ , есть разные мнения на этот счет. Здесь записано мнение Гайсаряна. В Википедии наоборот.) **Анализ достигающих определений** Один из видов анализа потоков данных. - Определением переменной** x называется инструкция, которая присваивает значение переменной x . - Использованием переменной** x называется инструкция, одним из операндов которой является переменная x . - Определение d **достигает** точки p , если существует путь от точки, непосредственно следующей за d , к точке p , такой, что вдоль этого пути d остается живым. - Передаточные функции достигающих определений.** Рассмотрим инструкцию I : $d : u = v + w$, расположенную между точками p_1 и p_2 программы. По определению передаточной функции $y = f_I(x)$ инструкция I сначала убывает все предыдущие определения u , а потом порождает d — новое определение u . Следовательно, $f_I(x) = \text{gen}_I \cup (x - \text{kill}_I)$, где x — состояние во входной точке инструкции I . Пусть базовый блок B содержит n инструкций, каждая из которых имеет передаточную функцию $f_i(x) = \text{gen}_i \cup (x - \text{kill}_i)$, $i = 1, 2, \dots, n$. Тогда передаточная функция для базового блока B может быть записана как $f_B(x) = \text{gen}_B \cup (x - \text{kill}_B)$, где $\text{kill}_B = \text{kill}_1 \cup \text{kill}_2 \cup \dots \cup \text{kill}_n$, и $\text{gen}_B = \text{gen}_1 \cup (\text{gen}_{n-1} - \text{kill}_n) \cup \dots \cup (\text{gen}_1 - \text{kill}_1 - \text{kill}_2 - \dots - \text{kill}_n)$. Таким образом, для каждого базового блока B_i можно выписать уравнение: $\text{Out}[B_i] = f_{B_i}(\text{In}[B_i])$. В случае анализа достигающих определений: $\text{Out}[B_i] = \text{gen}_{B_i} \cup (\text{In}[B_i] - \text{kill}_{B_i})$. **Pred(B)** — множество всех вершин ГПУ, которые непосредственно предшествуют вершине B . Следовательно, $\text{In}[B_i] = \bigcup_{P \in \text{Pred}(B_i)} \text{Out}[P]$. Произведя подстановку, получим систему уравнений: $$\text{In}[B_i] = \bigcup_{P \in \text{Pred}(B_i)} (\text{gen}_P \cup (\text{In}[P] - \text{kill}_P)), i = 1, 2, \dots, n$$ **Монотонные и дистрибутивные передаточные функции** - Полурешетка это множество L , на котором определена бинарная операция «если», такая, что $\forall x, y, z \in L$: - $x \wedge x = x$ (идемпотентность) - $x \wedge y = y \wedge x$ (коммутативность) - $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ (ассоциативность) - Для всех пар $x, y \in L$ определим отношение \leq : $x \leq y$ тогда и только тогда, когда $x \wedge y = x$. - Структурой потока данных** называется четверка $\langle D, F, L, \wedge \rangle$, где D — направление анализа (Forward или Backward), F — семейство передаточных функций, L — поток данных (множество элементов полурешетки), \wedge — реализация операции сбора. - Структура потока данных для **анализа достигающих определений**: $\langle \text{Forward}, GK, Def, \wedge \rangle$, где GK — семейство передаточных функций вида gen-kill ; \wedge — множество определений переменных. - Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **монотонной** (определение эквивалентно предыдущему), если $\forall x, y \in L, \forall f \in F (x \leq y) \Rightarrow f(x) \leq f(y)$. - Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **дистрибутивной**, если $\forall x, y \in L, \forall f$ |

