

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 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 $A \in (a_1, \dots, a_m)$, 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 $\Rightarrow \exists \varepsilon > 0: \forall \delta > 0 \exists M \in \{M\}: 0 < \rho(M, A) < \delta, |f(M) - b| \geq \varepsilon \Rightarrow \{M_n\} \rightarrow A \Rightarrow$ по Гейне $\{f(M_n)\} \rightarrow b \Rightarrow$ противоречие с $|f(M_n) - b| \geq \varepsilon$.

(\Leftarrow) \Rightarrow Гейне $\exists b$ – предел $u = f(M)$ в т. A по Коши и $\{M_n\} \rightarrow A$. Фиксируем $\varepsilon > 0$, по 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 $\forall \varepsilon > 0 \exists N \in \mathbb{N}: \forall m \geq N, p \in \mathbb{N}$ 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 $\forall \varepsilon > 0 \exists M', M'' \in U(M)$, удовлетворяющих $0 < \rho(M', M) < \delta, 0 < \rho(M'', M) < \delta$, следует $|f(M') - f(M'')| < \varepsilon$

Критерий Коши \exists **предела ф-ции**. Чтобы функция $f(x)$ имела конечное 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 $\frac{\varepsilon}{2} > 0$, $\forall M \in \{M\}: 0 < \rho(M, A) < \delta, 0 < \rho(M', A) < \delta$

$\Rightarrow |f(M') - b| < \frac{\varepsilon}{2}, |f(M'') - b| < \frac{\varepsilon}{2}$. 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) \Rightarrow $f(M)$ удовл. в т. A усл. Коши, $\{M_n\} : \{M_n\} \rightarrow A, M_n \neq A$.

Выберем $\varepsilon > 0$, соотв. $\delta > 0$, какое, что выше усл. Коши, для этого δ .

$\exists N \in \mathbb{N}: \forall n \geq N \Rightarrow 0 < \rho(M_{n+p}, A) < \delta$ (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 $\Rightarrow \{f(M_n)\}$ сход., к некоторому b .

□ (\Rightarrow) $\forall M \in \{M\} \rightarrow A, \{M_n\} \rightarrow A, \{f(M_n)\} \rightarrow b, \{f(M'_n)\} \rightarrow b'$. Тогда $f(M_1), f(M'_1), \dots, f(M_n), f(M'_n), \dots$ сходятся \Rightarrow все сб 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 $\{M\}$ – на множестве $\{N\}$ пространства E_k **определенна 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. $a = (a_1, \dots, a_k)$, а функция $u = f(x_1, \dots, x_m)$ непрерывна в точке $b = (b_1, \dots, b_m)$. Тогда сложная функция $f(x(t))$ непрерывна в точке 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 (пужка 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 $x_{n_1} = x_{n_2} = x_{n_3}$. Если $\{X\}$ – бесконечно, то по принципу Больцано-Вейерштрасса (у любого отсека мн-ва есть хотя бы 1 предельная точка) у $\{X\}$ есть предельная точка $\Rightarrow \exists$ 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 $\{x_{n_k}\}$, предел с которой в $[a, b]$. Очевидно, что посл-ть $\{f(x_{n_k})\}$ беск. большая, но в силу непр-ти функции в т. с эта посл-ть должна сходиться 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 $[a, b] \Rightarrow \exists M, m - \text{ТВГ и ТНГ}$ $f(x)$ 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 $|f(x'_n) - f(x''_n)| \geq \varepsilon$.

$\square x_n - x''_n < \frac{1}{n} \forall n \in \mathbb{N}$, но $|f(x'_n) - f(x''_n)| \geq \varepsilon$.

$\square \exists \{x'_{n_k}\} \subset [a, b]: \exists \lim_{k \rightarrow \infty} x'_{n_k} = c. \square \{x''_{n_k}\} : |x''_{n_k} - c| \leq$

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

[И.В. Садовничая, *Определенный интеграл*, page 14-23]

[И.С. Ломов, *Рукописные лекции по математическому анализу*]

[0.0.2 Определенный интеграл, его свойства. Основная формула интегрального исчисления.]

[Определения:]

$\square f(x)$ задана на $[a, b], a < b, T$ – разбиение $[a, b]: a = x_0 < x_1 < \dots < x_n = b$ на частичные сегменты $[x_0, x_1], \dots, [x_{n-1}, x_n]$. $\square \xi_i$ – 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 $\forall \varepsilon > 0 \exists \delta = \delta(\varepsilon) > 0: \forall \delta$ разбиение 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 $|I\{x_i, \xi_i\} - I| < \varepsilon$.

$$I = \lim_{\Delta \rightarrow$$

0.0.5 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 ряда числовую последовательность:

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

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 $|x| < \frac{L}{\varepsilon}$, 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. 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 начальную 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 $\sqrt[n]{|a_n x^n|} = |x|^n \sqrt[n]{|a_n|} < L + \frac{\varepsilon}{2} < 1$, 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 $\{\sqrt[n]{|a_n|}\}$, ($k = 1, 2, \dots$), сходящуюся к L . Но это означает, что, начиная с некоторого номера k , справедливо неравенство $L - \varepsilon < \sqrt[n]{|a_n|} < L + \varepsilon$.

Таким образом, начиная с этого номера k , справедливо неравенство $\sqrt[n]{|a_n x^n|} = |x|^n \sqrt[n]{|a_n|} > L - \varepsilon = 1$, 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 $\frac{1}{2|x|} < \frac{1}{2|x|}$ го $\sqrt[n]{|a_n|} < \frac{1}{2|x|}$. Стало быть, начиная с указанного номера, $\sqrt[n]{|a_n x^n|} = |x|^n \sqrt[n]{|a_n|} < \frac{1}{2} < 1$, 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 z_0 . Разложение 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 ■

Доказательство ф. Грина для простой области \square область D — криволинейная трапеция (области, правильная в направлении OY): $D = \{(x, y) | a \leq x \leq b, y_1(x) \leq y \leq y_2(x)\}$

Для кривой C , ограничивающей область D зададим направление обхода по часовой стрелке. Тогда:

0.0.6 Криволинейный интеграл, формула Грина.

$\square \varphi(t), \psi(t)$ непр. на $[a, b]$. Если рассматривать t как время, эти функции определяют закон движения по плоскости точки M с координатами $x = \varphi(t), y = \psi(t)$, $\alpha < t < \beta$. Множество $\{M\}$ всех точек M , координаты x, y , которых определяются уравнениями $\varphi(t), \psi(t)$, называется простой плоской кривой L , если различным значениям параметра t из $[\alpha, \beta]$ отвечают различные точки этого множества.

$\square \varphi(t), \psi(t) \in C[t]$. Уравнения $x = \varphi(t), y = \psi(t)$ задают параметрически кривую L , если \exists такая система сегментов $\{t_{i-1}, t_i\}$, разбивающих множество $\{t\}$, что для значений t из каждого данного сегмента этой системы все уравнения определяют простую кривую.

Справляемася кривая — кривая, имеющая конечную длину.

Длина кривой — это предел последовательностей длин ломаных, вписанных в эту линию, при условии, что длина наибольшего звена $\rightarrow 0$.

$\square x = \varphi(t), y = \psi(t) \in C[\alpha, \beta]$. Тогда кривая L , определяемая x, y , спрямляема и длину l ее дуть можно вычислить по формуле

$$l = \int_{\alpha}^{\beta} \sqrt{\varphi'^2(t) + \psi'^2(t)} dt$$

\square произвольную спрямляемую кривую L на плоскости Oxy , не имеющую точек самопересечения и самоналегания, называемую ограниченную точками A, B , описывающую параметрическими ур-ями:

$$\begin{cases} x = \varphi(t) \\ y = \psi(t) \end{cases}, t \in [a, b], A = (\varphi(a), \psi(a)), B = (\varphi(b), \psi(b))$$

\square на кривой L определены три непрерывные вдоль этой кривой функции $f(x, y) = P(x, y), Q(x, y) = Q(M)$.

\square разбиение отрезка $[a, b] : a = t_0 < t_1 < \dots < t_n = b$, $\Delta t_k = t_k - t_{k-1}$, $M_k = M_k(\varphi(t_k), \psi(t_k))$.

Выберем точки $N_k(\xi_k, \eta_k) \in M_{k-1} M_k$, $\xi_k = \varphi(t_k)$, $\eta_k = \psi(t_k)$, $t_k \in [t_{k-1}, t_k]$.

\square $\sigma_k = x_k - x_{k-1}$, $x_k = \varphi(t_k)$, $\Delta y_k = y_k - y_{k-1}$, $y_k = \psi(t_k)$

• Три интегральных суммы:

$$1. \sigma_1 = \sum_{k=1}^n f(N_k) \Delta t_k$$

$$2. \sigma_2 = \sum_{k=1}^n P(N_k) \Delta x_k$$

$$3. \sigma_3 = \sum_{k=1}^n Q(N_k) \Delta y_k$$

Число I_s , $s = 1, 2, 3$ называется пределом интегральной суммы σ_s при $\Delta \rightarrow 0$, если $\forall \varepsilon > 0 \exists \delta > 0 : \Delta < \delta \Rightarrow |I_s - \sigma_s| < \varepsilon$ независимо от выбора точек $N_k \in M_{k-1} M_k$.

Если существует предел I_1 интегральной суммы σ_1 при $\Delta \rightarrow 0$, то он называется криволинейным интегралом 1 рода от функции f по кривой L .

\square Если для всех номеров k , по крайней мере начиная с некоторого номера, справедливо неравенство $\frac{p_{k+1}}{p_k} < q < 1$ ($\frac{p_{k+1}}{p_k} \geq 1$), то ряд $\sum_{k=1}^{\infty} p_k$ сходится (расходится).

\square Если $\frac{p_{k+1}}{p_k} \geq 1$, то $p_{k+1} \geq p_k$, а значит $\lim_{k \rightarrow \infty} u_k \neq 0$, не выполнено необходимое условие сходимости ряда и ряд расходится.

Рассмотрим ряд из элементов геом. прогрессии:

$$\sum_{k=1}^{\infty} q^k = q + q^2 + \dots + \frac{1}{1-q}, |q| < 1.$$

Если $\frac{p_{k+1}}{p_k} \leq q = \frac{q^{k+1}}{q^k}$, то ряд $\sum_{k=1}^{\infty} p_k$ сходится по признаку сравнения, так как сходится ряд $\sum_{k=1}^{\infty} q^k$. ■

\square Если для всех номеров k , по крайней мере начиная с некоторого номера, справедливо неравенство $\frac{p_{k+1}}{p_k} < L < 1$ и $\lim_{k \rightarrow \infty} p_k = 0$, то ряд сходится при $L < 1$ и расходится при $L > 1$ (для $L = 1$ признак не работает).

$\square \forall \varepsilon > 0 \exists N \forall k \geq N : L - \varepsilon < p_{k+1} < L + \varepsilon$. Выберем $\varepsilon = \frac{1}{2}|L - 1|$.

Если $L < 1$, то $\frac{p_{k+1}}{p_k} < 0.5L + 0.5 = q < 1$, свели к 1 части, сходится.

Если $L > 1$, то $\frac{p_{k+1}}{p_k} > 0.5L + 0.5 > 1$, свели к 1 части, расходится. ■

Интегральный признак Коши-Маклорена. Пусть при $x \geq 1$ функция $f(x) \geq 0$ и не возрастает. Тогда ряд $\sum_{k=1}^{\infty} f(k)$ сходится или расходится одновременно с несобственным интегралом $\int_1^{\infty} f(x) dx$.

\square $\forall k \in \mathbb{N} \forall x \in [k, k+1]$, то $f(k) \geq f(x) \geq f(k+1) \Rightarrow f(k) \geq \int_k^{k+1} f(x) dx \geq f(k+1)$, $k = 1, \dots, n-1$, ($n \geq 2$)

$\square f(1) + f(2) + \dots + f(n-1) \geq \int_1^n f(x) dx \geq f(2) + \dots + f(n)$

$S_n - p_1 \leq \int_1^n f(x) dx \leq S_{n-1}$

Если $\int_1^{+\infty} f(x) dx$ сходится, то $\int_1^n f(x) dx \leq M \Rightarrow S_n \leq M + p_1 \Rightarrow$ сходится

Если $\int_1^{+\infty} f(x) dx$ расходится, то $\{f(x) \geq 0\} \int_1^n f(x) dx \rightarrow +\infty \Rightarrow S_{n-1} \rightarrow +\infty \Rightarrow$ расходится ■

Признак Лейбница. Пусть последовательность $\{u_k\}$, $u_k > 0 \forall k \in \mathbb{N}$ является невозрасташей и бесконечно малой. Тогда знакочередующийся ряд $\sum_{k=1}^{\infty} (-1)^k u_k$ сходится.

$\square S_{2n} = (u_1 - u_2) + (u_3 - u_4) + \dots + (u_{2n-1} - u_{2n}) = (> 0) + (> 0) + \dots + (> 0)$. Поэтому в силу невозрасташности последовательности $\{u_k\}$ последовательность $\{S_{2n}\}$ не убывает. С другой стороны, $S_{2n} = u_1 - (u_2 - u_3) - \dots - (u_{2n-2} - u_{2n-1}) - u_{2n} = u_1 - (> 0) - \dots - (> 0) - u_{2n}$. Поэтому в силу невозрасташности последовательности $\{u_k\}$ и того, что $u_{2n} > 0$, последовательность $\{S_{2n}\}$ ограничена сверху числом u_1 . Следовательно, $\{S_{2n}\}$ сходится к некоторому числу S . Но из того, что $S_{2n-1} = S_{2n} - u_{2n}$ и $\lim_{n \rightarrow \infty} u_{2n} = 0$ (из необх. условий сходимости), вытекает сходимость при $n \rightarrow \infty$ последовательности S_{2n-1} к тому же S . ■

■

■

■

■

■

0.0.9 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 $f_k = \langle f, \psi_k \rangle$, $k = 1, 2, \dots$

$S_n = \sum_{k=1}^n f_k \psi_k$ 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 частичная сумма 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

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

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 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 перпендикуляры, то задана **прямоугольная система координат**.

Пусть O_{xy} – 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 полученном 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

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

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. паралл. прям.

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

Классификация ПОВЕРХНОСТЕЙ второго порядка

Под общим уравнением алгебраической поверхности второго порядка в системе координат O_{xyz} пространства понимают уравнение вида:

$F(x, y, z) = a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2a_{12}xy + 2a_{13}xz + 2a_{23}yz + 2b_1x + 2b_2y + 2b_3z + c = 0$, где не все коэффициенты a_{ij} равны нулю, $a_{ij} = a_{ji}$

Введем обозначения:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \\ B = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{12} & a_{22} & a_{23} & b_2 \\ a_{13} & a_{23} & a_{33} & b_3 \end{pmatrix} = \begin{pmatrix} A & b \\ b^T & c \end{pmatrix}$$

Тогда уравнение примет вид:

$$F(x, y, z) = X^T AX + 2b^T X + c = 0, \quad A \neq A^T, \quad A \neq O.$$

Теорема. С помощью ортогонального преобразования координат (т.е. простым вращением и простым отражением) и параллельного переноса уравнение можно привести к одному из следующих типов:

1. $\lambda_1 x^2 + \lambda_2 y^2 + \lambda_3 z^2 + a_0 = 0$, $\lambda_1 \lambda_2 \lambda_3 \neq 0$
2. $\lambda_1 x^2 + \lambda_2 y^2 + b_0 z = 0$, $\lambda_1 \lambda_2 b_0 \neq 0$
3. $\lambda_1 x^2 + \lambda_2 y^2 + c_0 = 0$, $\lambda_1 \lambda_2 \neq 0$
4. $\lambda_2 y^2 + p_0 x = 0$, $\lambda_1 p_0 \neq 0$
5. $\lambda_2 y^2 + q = 0$, $\lambda_1 \neq 0$

Теорема. Для любой алгебраической поверхности второго порядка существует прямоугольная декартова система координат, в которой уравнение этой поверхности имеет канонический вид:

I тип:

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

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

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

Основные свойства алгоритма:

1. **Определенность** (помимо). 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 защищается 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 алгоритмов данных.

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

Машин 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 P в его конец.

	a	b	c	Λ	комментарий
q_1	Λ, R, q_2	Λ, R, q_3	Λ, R, q_4	R, Λ	анализ I симв., удаление
q_2	R, Λ	R, Λ	R, Λ	$a, !$	запись вправо
q_3	R, Λ	R, Λ	R, Λ	$b, !$	запись вправо
q_4	R, Λ	R, Λ	R, Λ	$c, !$	запись вправо

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 одноковым множествам слов. Случай машины Тьюринга **алгоритм** — это то, что может быть реализовано МТ.

Нормальный алгоритм Маркова:

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

$A = \{ \rightarrow \} B_i$. Каждая пара — формула подстановки для замены подслов в преобразуемом слове. Ищется вхождение слова A_1 в исходное слово. Если нашли, то заменяют 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 применилась завершающая формула, в которой \rightarrow .

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 одноковым множествам слов. Машин 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 док-ва]

Задача останова

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

Теорема. Такого алгоритма X не существует. [Давалась без док-ва]

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 , или 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:

$$\begin{cases} DA \rightarrow DA \\ NET \mapsto NET \end{cases}$$

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 $\langle \text{запись } Z \rangle Z \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle \text{запись } Z \rangle D A$. Пусть Z не самоприменим, тогда $\langle \text{запись } Z \rangle Z \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle \text{запись } Z \rangle N E T$.

— Пусть Z несамоприменим, тогда $\langle \text{запись } Z \rangle Z \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle \text{запись } Z \rangle N E T$.

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, *Машина Тьюринга и алгоритмы Маркова. Решение задач. Учебно-методическое пособие*.]

0.0.14 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 $Q Q^T = Q^T Q = I$

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 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
- (\Leftarrow) $UU^* = I \Rightarrow \exists U^{-1}, UU^* = I \Rightarrow \{\text{домн. слева на } U^{-1}\} \Rightarrow U^* = U^{-1} \Rightarrow U^* U = I$
- $(1 \Leftrightarrow 3)$ аналогично
- $(3 \Leftrightarrow 4)$ (\Rightarrow) $(Ux, Uy) = (x, UU^*y) = (x, y)$ (\Leftarrow) $(Ux, Uy) = (x, y) \Rightarrow (Ux, Uy) = (x, Iy) \Rightarrow UU^* = I$
- $(4 \Leftrightarrow 5)$ (\Rightarrow) очевидно, т.к. $|x| = \sqrt{(x, x)}$ (\Leftarrow) а) V — евкл.: $(x, y) = \sqrt{|x|^2 - |x|^2 - |y|^2 / 2}$. V — унит.: $(x, y) = (|x| + |y|^2 - |x|^2 - |y|^2 / 2)$ б) V — унит.: $|x|^2 = \sum |x_i|^2$, $|Ux|^2 = \sum |x_i|^2 \Rightarrow U$ сохр. длину
- $(4 \Leftrightarrow 5)$ (\Rightarrow) очевидно, т.к. $x - o/h \Rightarrow (Ue_i, Ue_j) = (e_i, e_j) = \delta_{ij}$ (\Leftarrow) Пусть e_1, \dots, e_n — о.и. $\forall x \in V : x = \sum x_i e_i, Ux = \sum x_i Ue_i \Rightarrow |x|^2 = \sum |x_i|^2$, $|Ux|^2 = \sum |x_i|^2 \Rightarrow U$ сохр. длину
- $(6 \Leftrightarrow 7)$ (\Rightarrow) очевидно (\Leftarrow) доказано в предыдущем пункте (\Leftarrow) ■

Примеры ортогональных матриц.

- $Q \in \mathbb{R}^{1 \times 1} \Rightarrow Q = [\pm 1]$
- $Q \in \mathbb{R}^{2 \times 2} \Rightarrow Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix}$

$$QQ^T = Q^T Q = I \Rightarrow \begin{cases} q_{11}^2 + q_{21}^2 = 1 \\ q_{11}q_{12} + q_{21}q_{22} = 0 \\ q_{12}^2 + q_{22}^2 = 1 \end{cases} \Rightarrow$$

$$\begin{cases} q_{11} = \cos \varphi \\ q_{21} = \sin \varphi \\ q_{12} = -k \sin \varphi \\ q_{22} = k \cos \varphi \end{cases}$$

$$1. k = 1 \Rightarrow Q = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix} \quad (\text{поворот}).$$

При $\varphi = \pi k$ получаются матрицы $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$.

При $\varphi \neq \pi k$ матрица не диагонализируется, так как у неё нет вещественных собственных значений.

$$2. k = -1 \Rightarrow Q = \begin{bmatrix} \cos \varphi & \sin \varphi \\ \sin \varphi & -\cos \varphi \end{bmatrix} \quad (\text{поворот и отражение}).$$

В этом случае собственные значения матрицы равны ± 1 \Rightarrow

получаются матрицы $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$.

Таким образом, в случае $Q \in \mathbb{R}^{2 \times 2}$ для ортогонального оператора в евклидовом пространстве существует ортогональный базис, в котором он имеет матрицу

$$\text{либо } \begin{bmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{bmatrix},$$

$$\text{либо } \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix}, (\varphi \neq \pi k).$$

Теорема. \forall ортогонального оператора Q в евклидовом пространстве \exists ортогональный базис e , в котором матрица оператора имеет вид:

$$Q = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & -1 & \\ & & & \ddots & \\ & & & & \cos \varphi_1 & -\sin \varphi_1 \\ & & & & \sin \varphi_1 & \cos \varphi_1$$

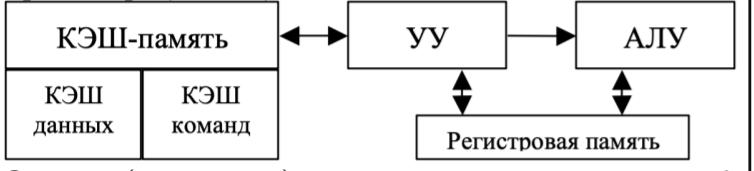
0.0.17 Понятие архитектуры ЭВМ. Принципы фон Неймана. Компоненты компьютера: процессор, оперативная память, внешние устройства. Аппарат прерываний.

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 выполнимых 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 единным 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 кампанде 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, посылая 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 процессора 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 прерывания 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.

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



Общая схема функционирования основных компонентов СП на базе интерпретатора:



Пример: Выбрать среднюю зарплату продавцов, которые обслуживают покупателей из штата 'CA'.

SELECT employee.last_name, employee.salary FROM employee JOIN job USING (job_id) JOIN customer ON employee_id = salesperson_id WHERE customer.state = 'CA' AND job.function = 'SALESPERSON';

[Кузнецов, Основы современных БД]

0.0.18 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 программ ("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).

• **Основные компоненты системы программирования:**

1. Транслятор переводит программы на исходном языке программирования в некоторый целевой язык. Вслучае, когда транслятор является компилятором, целевой язык — языки 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 транслятор.

3. **Макрогенератор или макропроцессор** выполняет преобразование текста программы, выполнив замену вызовов макропредопределений их определениями. Если макропроцессор входит в состав транслятора, его называют Препроцессором. В этом случае он выполняется непосредственно транслицией 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, порождаемых компилятором, а также файлов библиотек (которые являются наборами объектных модулей внутри одного файла), входящих в состав СП.

6. **Отладчик** используется для проверочных запусков программ и исправления ошибок. В нем обычно присутствуют такие возможности как интроспекция (получение типов данных) и анализ данных программы во 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 (make).

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 (например: 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, тью с компилятором 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 значения последовательности — лексемы. Используется различными обходами графов (DFS, BFS) и их комбинациями.

2. **Синтаксический анализ.** Синтаксический анализатор формирует из последовательности лексем (токенов) промежуточное представление программы (синтаксическое дерево, AST, дер24).

Используются алгоритмы разбора грамматики, 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 комбинации

4. Генерация промежуточного кода. Перевод AST в некоторое промежуточное представление, на котором удобнее производить оптимизации. Часто применяется SSA-форма (single static assignment), в которой каждая переменная может 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 генерация 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 $(x < n) \cdot (x > (32 - n))$ может быть использован циклический сдвиг вправо) и использования знаний компилятора для выбора наиболее эффективной инструкции для данной инструкции промежуточного кода.

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

Общая схема функционирования основных компонентов СП на базе интерпретатора:

Пример: Выбрать среднюю зарплату продавцов, которые обслуживают покупателей из штата 'CA'.

SELECT employee.last_name, employee.salary FROM employee JOIN job USING (job_id) JOIN customer ON employee_id = salesperson_id WHERE customer.state = 'CA' AND job.function = 'SALESPERSON';

[Кузнецов, Основы современных БД]

0.0.19 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, применяемых к значениям типа; определение способа внешнего представления значений типа (литералов).

• **Домен** — допустимое потенциальное, ограниченное подмножество значений данного типа.

• Для углубления термина отношение выделяются понятия заголовок отношения, значения отношения и переменной отношения. Пусть дана совокупность типов данных T_1, T_2, \dots, T_n , называемых также доменами, определение набора операций, применяемых к значениям типа; определение способа внешнего представления значений типа (литералов).

• **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 $A_i \in T_i$, A_i — столбцы (атрибуты) отношения.

Алгебра Кодда — основные реляционные операции

• При выполнении операции объединения (*UNION*) двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.

• Операция пересечения (*INTERSECT*) двух отношений производит отношение, включающее все кортежи, входящие в оба отношения-операндов.

• Отношение, являющееся разностью (*MINUS*) двух отношений, включает все кортежи, входящие в один из отношений-операндов, но не в другой.

• При выполнении декартова произведения (*TIMES*) двух отношений производится отношение, кортежи которого являются конкатенацией (сплелиением) кортежей первого и второго операндов.

• Редактор текстов используется для написания и редактирования исходного текста программ на языке программирования.

• Результатом ограничения (*WHERE*) отношения по некоторому условию является отношение, включающее все кортежи, входящие в это отношение.

• При выполнении проекции (*PROJECT*) отношения на заданное подмн

0.0.25 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:

$$y'(t) = f(t, y(t)) \quad (9)$$

$$y(t_0) = y_0 \quad (10)$$

Требуется определить функцию $y(t)$, удовлетворяющую уравнению (9) и условию (10).

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** (9), (10) 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 (9) для $t \in [t_1, t_2]$ и (10).

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

$$y(t) = y_0 + \int_{t_0}^t f(\tau, y(\tau)) d\tau \quad (11)$$

Лемма 1. Функция $y(t)$ является решением задачи Коши (9), (10) 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 (11) для $t \in [t_1, t_2]$.

▲ (⇒) Пусть функция $\bar{y}(t)$ является решением задачи с начальным условием (9), (10) 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 (11) для $t \in [t_1, t_2]$. Интегрируя (9) 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 (10), 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 (11) при $t \in [t_1, t_2]$.

(⇐) 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 (11) для $t \in [t_1, t_2]$, то есть:

$$\bar{y}(t) = y_0 + \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2] \quad (12)$$

Покажем, что $y(t)$ является решением задачи с начальным условием (9), (10).

Положив в (12) $t = t_0$, получим, что $\bar{y}(0) = y_0$. Следовательно условие (10) выполнено. Так как функция $\bar{y}(t)$ непрерывна на $[t_1, t_2]$, то правая часть равенства (12) 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 $[t_1, t_2]$. Дифференцируя (12), получим, что $\bar{y}'(t)$ удовлетворяет (9). ■

Оп. Функция $f(t, y)$, заданная прямоугольнике Π , удовлетворяет в П условию Лишица 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 П условию Лишица по y . Если $y_1(t), y_2(t)$ — решения задачи Коши (9), (10) 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 (9), (10), то из Леммы 1 следует, что они являются решениями интегрального уравнения (11). 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 Лишица, 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 П условию Лишица 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 1*, page 25-30]

0.0.26 Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шеннона.

Орграф — это ориентированный граф.

Вершины орграфа, в которые не входит ни одной дуги, называются истоками.

Орграф называется **акликлическим**, 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 акликлический 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 — выходами квадратурной формулы).

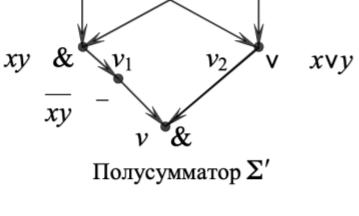
- каждой вершине, в которую входят $k \geq 1$ дуг, присвоена функция из базиса Б, зависящая от k переменных (вершина с присвоенной функцией при этом называется **функциональным элементом**);

• 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.



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.

- Аддитивная 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 , $B \in U_B^C$ (множество всех схем на базисе 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 строгого приобретённой 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 $f: L_B(f) = \min_{\text{СФЭ } \Sigma \in U_B^C \text{ реализующие } f} L(\Sigma)$

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 $L(\Sigma_f) \leq \underbrace{2^n}_{1} \underbrace{(n-1+n)}_{2} \underbrace{+n-1}_{4} = 2^{2^n} - 1$, 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 , $1 \leq q \leq n$.

Используется разложение 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 Σ' — мультиплексор, Σ'' — 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]

0.0.29 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

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

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

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

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

$$u(\xi, \eta) = \int f^*(\eta) d\eta = f_1(\xi) + f_2(\eta) \quad (17)$$

Обратно, каковы бы ни были дважды дифференцируемые функции f_1 и f_2 , функция $u(\xi, \eta)$, определяемая формулой (17), представляет собой решение уравнения $u_{tt} = a^2 u_{xx}$. Так как всякое решение уравнения $u_{tt} = 0$ может быть представлено в виде (17) при соответствующем выборе f_1 и f_2 , то формула (17) 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:

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

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]

Составим 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.

Составим 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.

Составим 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.

Составим 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.

Составим 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 :

$$u(x, t) = \frac{\varphi(x + at) + \varphi(x - at)}{2} + \frac{1}{$$

0.0.33 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 $\tilde{\beta}$, $\tilde{\alpha} \prec \tilde{\beta}$.

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 $[1, 0]$ — несравнимы.

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;

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

Вспомогательные леммы:

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

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

- 2. Если булева функция несамодвойственна, то из неё подстановкой вместо аргументов переменной x и её отрицания \tilde{x} можно получить либо константу 0, либо константу 1.

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

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

▲ Рассмотрим полином J Кегалкина 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 $g_2(\alpha) = a, g_3(\alpha) = b, g_4(\alpha) = c$.

$\phi(x, y) = f(x \oplus b)(y \oplus a) \oplus a(x \oplus b) \oplus b(y \oplus a) \oplus c = \{$ 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 $\{0, 1, \tilde{x}, x\}$.

• 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 нуль, $\varphi_0(0, 0, \dots, 0) = 1$. Возможны два случая: либо $\varphi_0(x) = \tilde{x}$, 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 $\varphi_1 = \tilde{x}$, 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: $[f_S, \tilde{x}] \supseteq [0, 1]$.

Константы получены.

• Получение конъюнкции. Имеем функцию $f_L \notin L$. Согласно лемме о нелинейной 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: $[f_L, 0, 1, \tilde{x}] \supseteq [x \cdot y, \tilde{x} \cdot \tilde{y}]$.

Конъюнкция получена.

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

[Презентации к госам от маткиба]

0.0.34 Логика 1-го порядка. Выполнимость и общезначимость. Общая схема метода резолюции.

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 .

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

t — 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 $Var_\psi \subseteq \{x_1, x_2, \dots, x_n\}$. Если $Var_\psi = 0$, то формула ψ называется замкнутой формулой, или предложением. $CForm$ — множество всех замкнутых формул.

• \neg — отрицание

• \wedge — конъюнкция

• \vee — дизъюнкция

• \rightarrow — импликация

• \forall — квантор всеобщности

• \exists — квантор существования

• \neg — отрицание

• $\neg\neg$ — двойное отрицание

• $\neg\neg\neg$ — тройное отрицание

• $\neg\neg\neg\neg$ — четверное отрицание

• $\neg\neg\neg\neg\neg$ — пятое отрицание

• $\neg\neg\neg\neg\neg\neg$ — шестое отрицание

• $\neg\neg\neg\neg\neg\neg\neg$ — седьмое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg$ — восьмое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — девятое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — десятое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — одиннадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — двенадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — тринадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — четырнадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — пятнадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — шестнадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — семнадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — восемнадцатое отрицание

• $\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg\neg$ — девятнадцатое отрицание

• \neg — двадцатое отрицание

• \neg — двадцать первое отрицание

• \neg — двадцать второе отрицание

• \neg — двадцать третье отрицание

• \neg — двадцать четвёртое отрицание

0.0.37 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>), 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 .

Декомпозицией 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 .

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

□ \exists некоторое отношение r со скелетом 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 .

□ 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. □ 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 неполная. 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	Cherger-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	Cherger-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 А, В и С, при этом В и С между собой никак не связаны и не зависят друг от друга, но по отдельности зависят от А, и для каждого значения А есть множество значений В, а также множество значений С.**

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](#)]

0.0.38 Язык ассемблера как машиннозависимый язык низкого уровня. Организация ассемблерной программы, секции кода и данных (на примере ассемблера пасм или тасм). Основные этапы подготовки к счёту ассемблерной программы: трансляция, редактирование внешних связей (компоновка), загрузка.

Язык ассемблера

Как правило, вычислительная система состоит из следующих основных компонентов: центральный процессор, оперативная память и внешние устройства.

Программа, предназначенная к выполнению, записывается в оперативную память в виде последовательности машинных инструкций (кодов), т.е. цифровых кодов, обозначающих те или иные операции.

Ассемблер – это программа, принимающая на вход текст, содержащий условные обозначения машинных программ, удобные для человека, и переводящий эти обозначения в последовательность соответствующих машинных команд понятных процессору. Язык таких условных обозначений называется **языком ассемблера**.

Программирование на языке ассемблера отличается от программирования на языках высокого уровня. В последних мы задаем лишь указания, а компилятор (программа, принимающая на вход программу на языке высокого уровня и выдающая эквивалентный машинный код) впоследствии сам определяет, какими ресурсами (регистрами и ячейками памяти) воспользоваться для хранения промежуточных результатов, какой алгоритм применить для решения какой-нибудь нетривиальной ситуации и т.д. В отличие от этого на языке ассемблера мы однозначно и недвусмысленно указываем, из каких машинных команд будет состоять наша программа, в этом понимании ассемблер не имеет практически никакой свободы.

Структура ассемблерной программы

Операционная система может установить пользовательской программе разные возможности по доступу к различным областям памяти. Область памяти может быть доступна для чтения, записи и исполнения.

В связи с этим, в современных языках ассемблера существует разделение виртуального адресного пространства на различные области памяти, так называемые секции. Эти секции определяются программистом в ходе создания программы. В результате перевода (трансляции) текста программы каждая секция будет превращена в последовательность байтов.

При запуске программы каждая такая последовательность байтов будет загружена в оперативную память и размещена в выделенных для нее ячейках памяти. В простых программах набор секций, как правило, ограничен тремя (не считая служебных):

- `.text` – секция кода.
- `.data` – секция статических инициализированных данных.
- `.bss` – секция статических неинициализированных данных, значения которых обнуляются операционной системой перед запуском программы.

Основные этапы подготовки к счёту ассемблерной программы

1. **Трансляция** – это перевод программы на язык ассемблера (в мемориках) в машинные коды. После трансляции получается

2. **Компоновка (связывание)**. На входе компоновщика один или несколько объективных файлов, на выходе – программа, готовая к исполнению (исполнимый файл). Компоновщик объединяет несколько файлов в один, пересчитывает адреса, связывает вызовы функций из разных файлов, для вызовов библиотечных функций (при статическом связывании) – добавляет их код в исполняемый файл и также связывает вызовы.

3. **Загрузка** – это загрузка программы секции из исполняемого файла в память и управление передается в точку входа программы. Также при загрузке происходит связывание с динамическими библиотеками.

Пример ассемблерной программы (язык пасм)

Программа, которая печатает 2 числа на экран:

```
; Equivalent C code
int main()
{
    int a=5;
    printf("a=%d, eax=%d\n", a, a+2);
    return 0;
}

extern printf      ; the C function, to be called
SECTION .data      ; Data section, initialized variables
a: dd 5            ; int a=5;
fmt: db "a=%d, eax=%d\n", 10, 0 ; The printf format, "\n", 10

SECTION .text      ; Code section.
global main        ; the standard gcc entry point
main:             ; the program label for the entry point
    push ebp       ; set up stack frame
    mov esp, [ebp] ; put a from store into register
    add eax, 2     ; a+2
    push ax         ; value of a+2
    push dword [a] ; value of variable a
    push dword fmt ; address of ctrl string
    call printf    ; Call C function
    add esp, 12     ; pop stack 3 push times 4 bytes
    mov esp, [ebp] ; takedown stack frame
    pop ebp         ; same as "leave" op
    mov eax, 0       ; normal, no error, return value
    ret
```

[Лекции асм 1 помока]

0.0.39 Логическое программирование. Декларативная семантика и операционная семантика; соотношение между ними. Стандартная стратег

0.0.41 Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задач на параллельных вычислительных системах.

Ускорение, получаемое при использовании параллельного алгоритма для p процессоров выражается: $S = \frac{T_1}{T_p}$, где T_1 — время выполнения задачи на одном процессоре, T_p — время параллельного выполнения задачи на p процессорах.

Закон Амдала: $S \leq \frac{1}{f + \frac{1-f}{p}}$, где

f — доля операций, которые обязаны быть выполнены последовательно ($0 \leq f \leq 1$),

p — число процессоров.

Следствие 1. Для того чтобы ускорить программу в q раз, необходимо не менее, чем в q раз, ускорить не менее, чем $(1 - \frac{1}{q})$ -ю часть программы. Это необходимо, но не достаточно условие.

Следствие 2. (при большом числе процессоров): $S \approx \frac{1}{f}$.

Граф алгоритма

Будем представлять программы с помощью графов.

Граф алгоритма — ориентированный граф, состоящий из вершин, соответствующих операциям алгоритма, и направленных дуг, соответствующих передаче данных (результаты одних операций передаются в качестве аргументов другим операциям) между ними. Не следует путать его с графиками управления программы и тем более с её блок-схемой. Вершины: процедуры, циклы, линейные участки, операторы, итерации циклов, срабатывания операторов.

Операции — одна вершина для каждой операции.

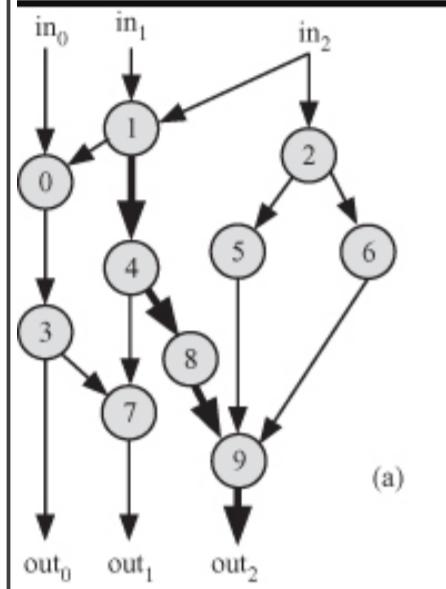
Срабатывания операторов — столько вершин, сколько раз каждый оператор сработал.

Дуги: отражают связь (отношение) между вершинами.

Особенностью графа алгоритма являются:

- его ацикличность, нет кратких дуг;

- невозможность, в общем случае, его описания простым перечислением, в силу того, что его составляющие могут зависеть от внешних параметров решаемой им задачи (например, алгоритм, реализующий метод Гаусса — от размера матрицы)



Выделяют два типа отношений: операционное и информационное.

Операционное отношение: две вершины A и B соединяются направленной дугой $A \Rightarrow B$ тогда и только тогда, когда вершина B может быть выполнена сразу после вершины A .

Информационное отношение: две вершины A и B соединяются направленной дугой $A \Rightarrow B$ тогда и только тогда, когда вершина B используется в качестве аргумента некоторое значение, полученное в вершине A .

Операционный граф: вершины — операции, дуги — операционные отношения.

Граф операционной истории программы: вершины — срабатывания операторов, дуги — операционные отношения.

Информационный график: вершины — операции, дуги — информационные отношения.

Граф информационной истории программы: вершины — срабатывания операторов, дуги — информационные отношения.

Независимо от способа построения ориентированного графа, те его вершины, которые не имеют ни одной входящей или выходящей дуги, будем называть соответственно **входными** или **выходными** вершинами

Информационная история программы.

Вершины: срабатывания операторов

Дуги: информационное отношение

```
for (i = 0; i < n; ++i) {
    A[i] = A[i - 1] + 2;
    B[i] = B[i] + A[i];
}
```

Критический путь графа алгоритма — это длина самого длинного пути от <входа> графа к его <выходу> (от первого оператора к последнему).

Ярусно-параллельная форма графа алгоритма позволяет описать ресурс параллелизма программ и алгоритмов.

• Начальная вершина каждой дуги расположена на ярусе с номером m , чем номер яруса конечной вершины.

• Между вершинами, расположенными на одном ярусе, не может быть дуг.

Высота ЯПФ — это число ярусов

Ширина яруса — число вершин, расположенных на ярусе

Ширина ЯПФ — это максимальная ширина ярусов в ЯПФ.

Высота ярусно-параллельной формы — это сложность параллельной реализации алгоритма/программы. ЯПФ определяется неоднозначно.

Ярусно-параллельная форма называется **канонической**, если у любой вершины, кроме вершины первого яруса, есть входная дуга, идущая с предыдущего яруса. Высота канонической ЯПФ = длине критического пути + 1.

Этапы решения задач на параллельных вычислительных системах **Это точно то о чём спрашивается!** Решение задачи на параллельной вычислительной системе будет эффективным только в том случае, если на всем пути от Задачи до Компьютера не возникнет ни одного <узкого места>. Центральная проблема: отображение программ и алгоритмов на архитектуру параллельных вычислительных систем (Co-Design).

Весь процесс решения некоторой задачи на параллельной ВС можно разбить на следующие этапы:

1. Формулировка задачи.

2. Составление модели исследуемого в задаче объекта.

3. Определение метода решения задачи для получения необходимой результирующей информации на основании используемой модели.

4. Разработка алгоритма решения задачи на основе используемого метода и модели исследуемого в задаче объекта.

5. Выбор технологии программирования.

6. Разработка программы для соответствующей параллельной ВС на основе имеющегося алгоритма и получение результирующей информации после выполнения программы.

[Пушкин-Залупкин, *Нашки источники!*, page 69-96]

0.0.42 Классификация языков, определяемых конечными автоматами, регулярными выражениями и праволинейными грамматиками. Эквивалентность и минимизация конечных автоматов.

Грамматики и регулярные множества и выражения

• Грамматика — это четверка $G = (N, T, P, S)$, где N — алфавит нетерминальных символов, T — алфавит терминальных символов, $N \cap T = \emptyset$, P — конечное множество правил вида $\alpha \rightarrow \beta$, где $\alpha \in (N \cup T)^*$ (хотя бы 1), $\beta \in (N \cup T)^*$, $S \in N$ — начальный знак или аксиома грамматики.

• Отношение выводимости \Rightarrow — если $\gamma \rightarrow \delta \in P$, то $\alpha\gamma\beta \Rightarrow \alpha\delta\beta$, $\forall\alpha, \beta \in (N \cup T)^*$.

• Сентенциальная форма грамматики G — цепочка, выводимая из начального символа.

• Языком, порождаемым грамматикой G ($L(G)$) называется множество всех её терминальных сентенциальных форм $L(G) = \{w | w \in T^*\}, S \Rightarrow_G^* w$.

• Грамматики **эквивалентны**, если они порождают один и тот же язык.

• Классификация грамматик по Хомскому:

Тип 0 Любая порождающая грамматика.

Тип 1 (Неукорачивающая или контекстно-зависимая грамматика) Если каждое правило кроме $S \rightarrow e$ имеет вид $\alpha \rightarrow \beta$, где $|\alpha| \leq |\beta|$, и в том случае, когда $S \rightarrow e \in P$, S не встречается в правых частях правил.

Тип 2 (Контекстно-свободная грамматика.) Если каждое правило имеет вид $A \rightarrow \beta$, где $A \in N$, $\beta \in (N \cup T)^*$.

Тип 3 (Регулярная праволинейная (или леволинейная) грамматика.) Каждое правило имеет вид либо $A \rightarrow xB$, либо $A \rightarrow x$, где $A, B \in N$, $x \in T^*. (для леволинейных поменять x на Bx .)$

• Типом языка называется максимальный тип грамматики, порождающей этот язык.

• Регулярное множество определяется рекурсивно:

1. \emptyset — регулярное множество в алфавите T .

2. $\{e\}$ — регулярное множество в T , где e — пустая цепочка.

3. $\{a\}$ — регулярное множество в T для каждого $a \in T$.

4. Если P и Q — регулярные множества в T для каждого $a \in T$, то регулярными являются $P \cup Q$ (объединение), PQ (конкатенация), P^* (итерация).

5. Ничто другое не является регулярным множеством.

• Регулярное выражение определяется рекурсивно:

1. \emptyset — регулярное выражение, обозначающее рег. множество \emptyset .

2. e — регулярное выражение, определяющее рег. множество $\{e\}$.

3. a — регулярное выражение, определяющее рег. множество $\{a\}$.

4. Если p и q — регулярные выражения, то регулярными являются $p|q$ (обозначает множество $P \cup Q$), pq (обозначает множество PQ), p^* (обозначает множество P^*).

5. Ничто другое не является регулярным выражением.

Конечные автоматы

Для распознавания регулярных множеств служат конечные автоматы.

• Недетерминированный КА — пятёрка $M = (Q, T, D, q_0, F)$ где Q — конечное множество состояний, T — входной алфавит, D — функция переходов, отображающая множество $Q \times (T \cup \{\epsilon\})$ во множество подмножеств Q , $q_0 \in Q$ — начальное состояние, $F \subseteq Q$ — множество заключительных состояний.

• Детерминированный КА — НКА, в котором $D(q, e) = \emptyset$ для любого $q \in Q$ и $D(q, a)$ содержит не более 1 элемента для любых $q \in Q$ и $a \in T$.

• Теорема 1. Язык, допускаемый ДКА, — множество всех его допустимых цепочек — является регулярным множеством.

• Теорема 2. Пусть $G = (N, T, P, S)$ — праволинейная. Тогда существует НКА $M = (Q, T, D, q_0, F)$, для которого $L(M) = L(G)$.

• Теорема 3. Для каждого НКА M существует праволинейная грамматика G такая, что $L(M) = L(G)$. Но НКА можно построить эквивалентный ему ДКА. По ДКА можно построить эквивалентный ДКА с минимальным числом состояний.

Состояние s конечного автомата M **эквивалентно** состоянию t конечного автомата N тогда и только тогда, когда автомат M , начав работу в состоянии s будет допускать в точности те же цепочки, что и автомат N , начавший работу в состоянии t .

Два автомата M_1 и M_2 называются эквивалентными, если их входные алфавиты совпадают и для каждого состояния M_1 существует эквивалентное ему состояние M_2 .

Алгоритм минимизации:

1. Удаляем все недостаточные состояния и те, из которых нет пути в конечное состояние.

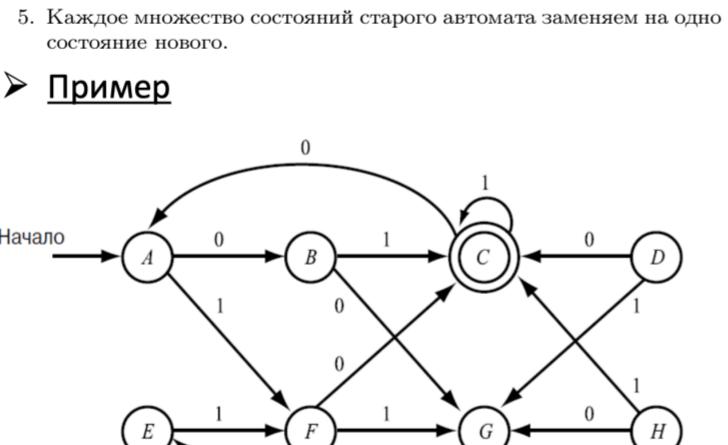
2. Разбиваем исходное множество состояний на два, первое из которых состоит из всех конечных состояний, а второе — из всех остальных: $\{F\}, \{Q \setminus F\}$.

3. Если элементы одного множества при переходе по одному символу попадают в разные множества, то разбиваем исходное множество так, чтобы в одном множестве оказались только те состояния, которые переходят в состояния одного множества. Например пусть есть множество $\{s_1, s_2, s_3, s_4\}, \{s_5, s_6\}$ и переходы $s_1 \xrightarrow{a} s_1, s_2 \xrightarrow{a} s_1, s_3 \xrightarrow{a} s_5, s_4 \xrightarrow{a} s_6$. Тогда новые множества будут иметь следующий вид: $\{s_1, s_2\}, \{s_3, s_4\}, \{s_5, s_6\}$.

4. Продолжаем разбиение до тех пор, пока это возможно.

5. Каждое множество состояний старого автомата заменяем на одно состояния нового.

Пример



Там где крестик — состояния неэквивалентны. Переирем все возможные переходы из каждого состояния. В данном случае из каждого состояния по 0 и 1. Если два состояния переходят по одному символу в разные состояния, то ставится крестик (они уже неэквивалентны)

[Игнатьев, *Лекция по формальным языкам*, page 25-37]

0.0.43 Жизненный цикл программного обеспечения (ПО). Основные виды деятельности при разработке ПО. Каскадная и итерационная модели жизненного цикла.

Весь период существования ПО, связанный с подготовкой к его разработке, разработкой, использованием и модификациями, до того момента, когда полностью прекращается всяко ее использование, называют **жизненным циклом ПО**.

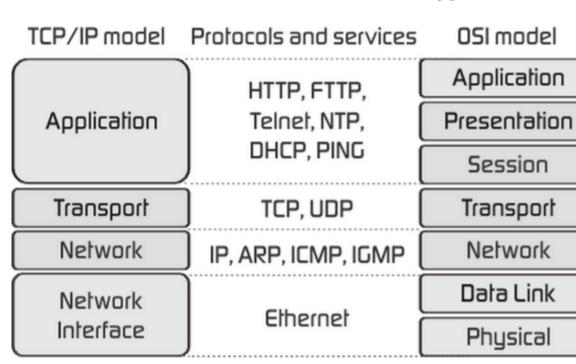
Процессы жизненного цикла ПО строятся из отдельных видов деятельности (activities). Стандартом ISO/IEC 12207 Standard for Information Technology определены 74 вида деятельности, связанных с разработкой и поддержкой ПО. Основные из них (думаю, из этого знать нужно только выделенное, как основные типы деятельности):

</

0.0.45 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

- 1. **Децентрализация управления** — нет единого центра управления для сети Интернет.
- 2. **Выход из строя одного компьютера или участка сети не приводит к неработоспособности всей сети.**
- 3. **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 (длятограммы) либо последовательно друг за другом по виртуальным соединениям. Узел-принимик из пакетов собирает сообщение. В таких сетях по одной физической линии связи могут обмениваться данными много узлов.
- 4. **Инкапсуляция** — последовательное вложение протокольной единицы (PDU) вышестоящего уровня в протокольную единицу нижележащего уровня. PDU вышестоящего уровня не доступны на нижележащем уровне (нижележащий уровень не понимает структуры данных вышеуказанных уровней).
- 5. Каждый уровень имеет строго определенный интерфейс с нижележащим и вышеуказанным уровнями и набор сервисов, которые может использовать вышеуказанный уровень.



ISO/OSI:

Физический — передача последовательности битов через физическую линию.

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

Сетевой — построение маршрута между отправителем и получателем. **Транспортный** — получение данных из вышеперасположенного уровня, 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 решений проблем, связанных с представлением данных, в основном — проблемами синтаксиса и семантики передаваемой информации.

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

TCP/IP:

Прикладной уровень. На прикладном уровне (Application layer) работают 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 правильность 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 пакетов на физическом уровне (то есть специальные последовательности бит, определяющие 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 определённое 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 $C_s = (C_1, C_2, \dots, C_r)$ — 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
- Длину вектора C устанавливает администратор

Алгоритм Дейкстры (пример алгоритма по состоянию канала):

1. Определение топологии сети: каждый маршрутизатор передаёт лавиной всем своим соседям состояния своих линий и строит 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.



[Смелянский, Компьютерные сети, том 1] [Смелянский, Компьютерные сети, том 2]

0.0.46 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.

- **Валидация** — проверка того, что сам продукт правилен, то есть подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

- **Методы обеспечения качества** представляют собой техники, гарантирующие 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 возвращаемыми ею 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

0.0.49 Теоретические основы передачи данных, физический уровень стека протоколов. Системы передачи данных Ethernet и Wi-Fi: алгоритмы работы, управление множественным доступом к каналу

Теоретические основы

Данные — описание фактов, явлений.

Сигнал — представление данных при передаче.

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).

Полосы пропускания канала — спектр частот, которые канал пропускает без существенного понижения мощности сигнала.

Скорость передачи зависит от способа кодирования данных на физическом уровне и **сигнальной скорости** — скорости изменения значения сигнала.

Пропускная способность канала — максимальная скорость, с которой канал способен передавать данные.

Теорема Найквиста-Котельникова: $R_{max_data_rate} = 2 * D * \log_2 L$ (bps per second)

- $R_{max_data_rate}$ — максимальная пропускная способность канала

- D — ширина полосы пропускания канала (максимальная частота сигнала в спектре).

- L — количество уровней (значений) сигнала.

Теорема Котельникова — Аналоговый сигнал $u(t)$, не содержащий частот выше F_{max} Гц, полностью определяется последовательностью своих значений в моменты времени, отстоящие друг от друга на $\frac{1}{2F_{max}}$

Шум в канале измеряется, как соотношение мощности полезного сигнала к мощности шума: S/N (измеряется в децибелах $1dB = 10 \cdot \log_{10} S/N$)

Теорема Шеннона: $R_{max} = D * \log_2(1 + \frac{S}{N})$ bps (bits per second)

- R_{max} — максимальная скорость передачи данных по каналу с шумом

- D — ширина полосы пропускания канала (максимальная частота сигнала в спектре).

- S/N — соотношение сигнал-шум в канале, S — мощность сигнала, N — мощность шума

Характеристики физической среды:

- Полоса пропускания

- Пропускная способность

- Задержка

- Загущение

- Помехоустойчивость

- Достоверность передачи

- Стоимость

- Простота прокладки

- Сложность обслуживания

Ethernet

- Узлы в сети Ethernet адресуются при помощи 6-байтового двоичного числа, называемого MAC-адресом (Media Access Control — управление доступом к носителю).

- Распределением MAC-адресов между производителями оборудования занимается международная некоммерческая организация IEEE (Institute of Electrical and Electronics Engineers — Институт инженеров электротехники и электроники).

- Любой участник может послать в сеть сообщение, но только тогда, когда в ней <тихо> — нет другой передачи.

- Ethernet использует протокол разрешения адресов (ARP) для определения MAC-адресов и их сопоставления с известными адресами сетевого уровня.

- У каждого узла в IP-сети есть MAC и IP-адреса.

- Узел должен использовать собственные MAC- и IP-адреса в полях источника, а также предоставить MAC и IP-адреса для назначения.

- Несмотря на то, что IP-адрес назначения будет предоставлен более высоким уровнем OSI, отправляющему узлу необходимо найти MAC-адрес назначения для данного канала Ethernet. В этом заключается назначение протокола ARP.

Wi-Fi

- Обычно схема сети Wi-Fi содержит не менее одной точки доступа и не менее одного клиента.

- Также возможно подключение двух клиентов в режиме точка-точка (Ad-hoc), когда точка доступа не используется, а клиенты соединяются посредством сетевых адаптеров <напрямую>.

- Точка доступа передаёт свой идентификатор сети (SSID — Service Set Identifier) с помощью специальных сигнальных пакетов.

- Зная SSID сети, клиент может выяснить, возможно ли подключение к данной точке доступа.

- При попадании в зону действия двух точек доступа с идентичными SSID приёмник может выбирать между ними на основании данных об уровне сигнала.

- Стандарт Wi-Fi даёт клиенту полную свободу при выборе критерии для соединения.

Управление множественным доступом

- Проблема управления множественным доступом встает в тот момент, когда несколько отправителей хотят отправить свои данные через сеть.

- Протокол множественного доступа может определять и предотвращать коллизию пакетов (кадров) данных при условии, что в качестве режима конкурирующего доступа используется метод доступа к каналу, или зарезервированы ресурсы для установления логического канала.

- Механизм множественного доступа основан на схеме мультиплексирования (передача нескольких потоков данных с меньшей скоростью по одному каналу связи) физического уровня.

Протоколы:

- CSMA/CD (Carrier Sense Multiple Access with Collision Detection) Если во время передачи кадра рабочая станция обнаруживает другой сигнал, занимающий передающую среду, она останавливает передачу, посыпает сигнал преднамеренной помехи и ждёт в течение случайного промежутка времени (backoff delay), перед тем как снова отправить кадр. В Ethernet коллизии могут быть обнаружены сравнением передаваемой и получаемой информации. Если она различается, то другая передача накладывается на текущую (возникает коллизия) и передача прерывается немедленно. Поясняется сигнал преднамеренной помехи, чтобы вызвать задержку передачи всех передатчиков на произвольный интервал времени, снижая вероятность коллизии во время повторной попытки. Ethernet является классическим примером протокола CSMA/CD.

- Aloha (слотированная) Модификация alohi. Все время работы канала разделяется на слоты. Размер слота при этом должен быть равен максимальному времени кадра. Такая организация работы канала требует синхронизации, одна из станций испускает сигнал начала очередного слота, поскольку передачу теперь можно начинать не в любой момент, а только по специальному сигналу, то время на обнаружение коллизии сокращается вдвое. Размер слота при этом должен быть равен максимальному времени кадра.

[Смелянский, *Компьютерные сети, том 2*, page 69-96]

0.0.50 Понятие о парадигме программирования. Основные парадигмы программирования. Языки и парадигмы программирования.

Парадигма программирования — совокупность идей и понятий, определяющих стиль программирования.

Императивная парадигма (процедурная) основана на фон-неймановской модели компьютера. Основные понятия императивных языков программирования представляют собой абстракции основных понятий фон-неймановской модели.

Любой императивный язык программирования включает в себя:

- Понятие переменной — абстрагирует понятие ячеек памяти.
- Понятие операции — обобщает арифметико-логические команды.
- Понятие оператора — абстрагирует общее понятие команды.

Операторы: присваивания, управления (циклы, операторы выбора, перехода и т.д.).

Далее во всех парадигмах рассмотрена задача перевернуть последовательность символов.

C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_ELEMENTS 1024
char * pInput = NULL;
char Buffer[MAX_ELEMENTS];
int main() {
    int current, count = 0;
    pInput = (char*)calloc(1, sizeof(char));
    while (fgets(Buffer, sizeof(Buffer), stdin)) {
        int oldCount = count;
        current = strlen(Buffer);
        count += current;
        pInput = (char*)realloc(pInput, count+1);
        if (*pInput) {
            fprintf(stderr, "No memory");
            return 1;
        } else strncat(pInput + oldCount, Buffer);
    }
    for (char * pBegin = pInput, * pEnd = pInput + count - 1;
         pBegin < pEnd; ++pBegin, ++pEnd) {
        char t = *pBegin;
        *pBegin = *pEnd;
        *pEnd = t;
        puts(pInput);
        free(pInput);
        return 0;
    }
}
```

Объектная парадигма основана на понятии объекта. Объект обладает состоянием и поведением.

Поведение — взаимодействие с другими объектами посредством сообщений. Для каждого вида сообщения существуют обработчики, которые могут модифицировать состояние и посыпать сообщения. Объекты с одинаковым поведением и набором состояний объединяются в классы. Объектная парадигма сочетается с императивной.

Состояние описывается набором переменных, обработчики сообщений представляют собой процедуры или функции, имеющие доступ к состоянию. Посылка сообщения сводится к вызову соответствующего обработчика.

Объектная парадигма основана на императивной.

C++

```
#include <iostream>
#include <string>

class Circle {
    double radius;
    std::string color;
public:
    Circle(double r = 1.0, std::string c = "red") :
        radius(r), color(c){}
    double getRadius() { return radius; }
    std::string getColor() { return color; }
    double getArea() { return radius*radius*3.1416; }
};

int main() {
    Circle c1(1.2, "blue");
    std::cout << "Radius" << c1.getRadius()
        << " Area=" << c1.getArea()
        << " Color=" << c1.getColor() << std::endl;

    Circle c2(3.4);
    std::cout << "Radius=" << c2.getRadius()
        << " Area=" << c2.getArea()
        << " Color=" << c2.getColor() << std::endl;
    return 0;
}
```

Функциональная парадигма основана на понятиях функции и выражение.

Основная операция — вызов функции.

Выражение — это комбинация вызовов функций.

Функции — объекты первого порядка, то есть могут быть значениями переменных, возвращаемыми и передаваемыми значениями, могут быть созданы динамически.

При вызове функции вначале вычисляются выражения — фактические параметры — затем их значения подставляются вместо аргументов в выражение-тело функции. Наконец, вычисляется значение тела, которое и будет значением вызова.

Основные понятия:

- Лисп-выражение — это атом, либо список.
- Атом — это либо символ (идентификатор), либо число.
- Список — это последовательность членов списка, разделенных пробелами, заключенная в круглые скобки.

Член списка — это либо атом, либо список.

- Есть специальный атом — nil, который представляет собой пустой список. Это единственный атом, который одновременно является и списком.

Лисп

```
(defun shift (l r)
  (if (null l)
      r
      (shift (cdr l) (cons (car l) r)))
)
(defun reverse (s) (shift s nil))
```

Логическая парадигма: парадигма программирования, основанная на математической логике — программы в ней задаются в форме логических утверждений и правил вывода.

Идея — описать семантику задачи в терминах формул исчисления предикатов.

Программа на Прологе = описание предикатов.

Пример: append(L1, L2, L3) — истина, если L3 — конкатенация списков L1 и L2.

Предикаты описываются двояко: факты и правила.

Факт: имяПредиката(список_константных_аргументов).

Правило: P(X1,X2,X3,...) :- P1(X1,...), P2(X1,...,Y,...), ...

— означает импликацию, запятая в правой части — логическое "и".

Правила соединяются в единую формулу путем логической операции "или". Фактически получаем единую формулу — дизъюнкцию импликаций (каждое правило — хорновский дизъюнкт).

Prolog

```
reverse([], []).
reverse([X|Q], Z) :- reverse(Q, Y), append(Y, [X], Z).
```

[Головин, *Материалы по языкам программирования к госэкзамену*]

0.0.51 Синхронизация в распределенных системах. Синхронизация времени. Логические часы. Выборы координатора. Взаимное исключение. Координация процессов.

Основное

Распределенная (компьютерная) система (PC) — совокупность связанных сетью независимых компьютеров, которая представляется пользователю единим компьютером.

Свойства децентрализованных алгоритмов:

1. используемая информация распределена среди множества ЭВМ;
2. процессы действуют на основе только локальной информации;
3. отсутствие критического узла, выход из строя которого приводит к краху алгоритма;
4. отсутствие общего источника глобального времени.

пункты 1-3 от недопустимости хранения всей информации необходимой для принятия решения в одном месте.

Синхронизация времени

о **Аппаратные часы** — счетчик временных сигналов и регистр с начальным значением счетчика [из слайдов].

(**Аппаратные часы** — счетчик времени, система содержащая автономный источник питания и регистр. [из вики]).

о Отношение “произошло до” (\rightarrow): $a \rightarrow b$ означает, что все процессы согласны, что сначала произошло событие a , а затем b . Оно очевидно в 2-х случаях:

- оба события $(a \text{ и } b)$ произошли в одном процессе;
- событие a — отправка сообщения m , событие b — прием m .

Отношение \rightarrow транзитивно ($a \rightarrow b \text{ и } b \rightarrow c \Rightarrow a \rightarrow c$).

Если события x и y произошли в разных процессах, не обменявшись сообщениями, то отношение $x \rightarrow y$ и $y \rightarrow z$ неверны. Такие события x и y называются одновременными.

о **Логические часы** — согласованное (между ЭВМ РС) время (потенциально) не имеющее ничего общего с астрономическим временем.

Введем логическое время C следующим образом: если $a \rightarrow b$, то $C(a) < C(b$

0.0.53 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
 - присваивание: $x \leftarrow op\ y; x \leftarrow op\ y; x \leftarrow y; x[i] \leftarrow y; x \leftarrow y[1];$
 - переходы: $goto\ L; ifTrue\ x\ goto\ L; ifFalse\ x\ goto\ L;$
 - дополнительное: $param\ x; call\ x\ n; return\ y;$
2. таблицы символов,
- переменные, их имена в программе и атрибуты, такие как область видимости, тип, для имен функций - число параметров, и т. п.

- LIR (нижний 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).

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.

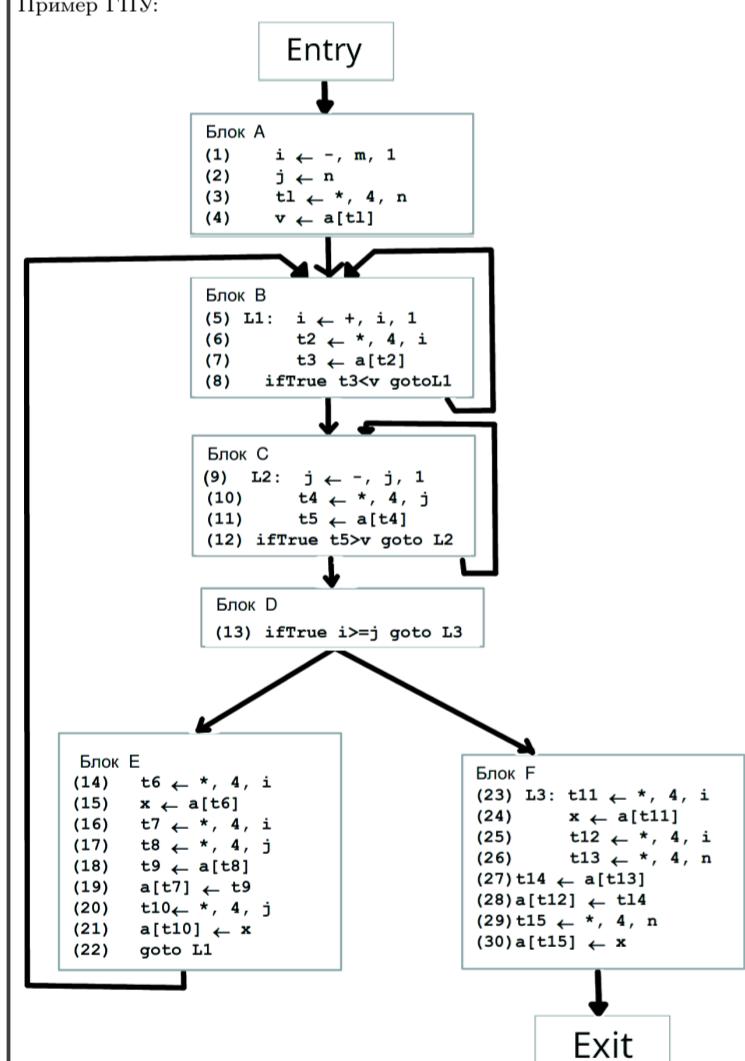
Если первая инструкция базового блока имеет метку, то в этот блок будет входить дуги из всех базовых блоков, у которых последняя инструкция — переход на эту метку.

Чтобы построить ГПУ, надо

1. выделить базовые блоки;
2. прорешести дуги 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 Гайсаряна, slide 7-28]

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

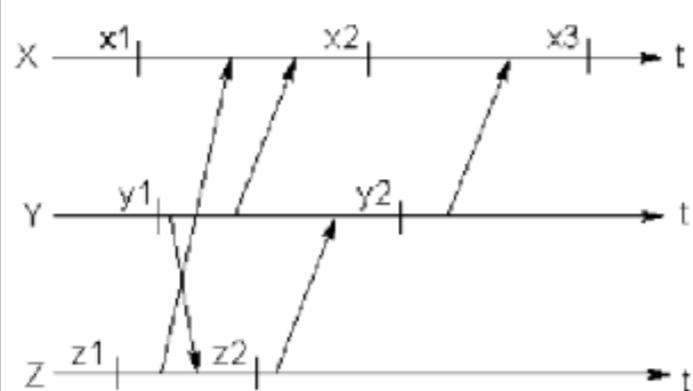
Основные определения

- 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: <визитные> (система активна, но некорректно работает), 2) <пропажа признаков жизни> (частичная или полная).

Два подхода - восстановление решения после отказа системы (или ее компонента) и преобразование отказа системы (отказоустойчивость). 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 отказу процесса Y в у1, а затем и процессов X и Z в начальных состояниях x1 и y1. Этот эффект известен как эффект домино.

- Множество контрольных 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 контрольных точек называется консистентным, если для любой зафиксированной операции приема сообщения, соответствующая операция посылки также зафиксирована (нет сообщений-сирот).

Методы фиксации контрольных точек

1. Простой метод — фиксация локальной контрольной точки после каждой операции посылки сообщения. При этом посыпка сообщения и фиксация должны быть единой неделимой операцией (транзакцией). Множество последних локальных контрольных точек является консистентным (но не строго консистентным). Чтобы избежать потерь сообщений при восстановлении с использованием консистентного множества контрольных точек необходимо повторить отправку тех сообщений, квантации о получении которых стали недействительными в результате отказа. Используя временные метки сообщений можно распознавать сообщения-призраки.
2. Синхронная фиксация. Два вида контрольных точек - постоянные и пробные.

Постоянная контрольная точка — это локальная контрольная точка, являющаяся частью консистентной глобальной контрольной точки. **Пробная контрольная точка** — это временная контрольная точка, которая становится постоянной только в случае успешного завершения алгоритма.

Фиксация: 1 фаза. Инициатор фиксации (процесс P_i) создает пробную КТ и просит все остальные процессы сделать то же самое. При этом процессу запрещается посыпать неслужебные сообщения после того, как он сделает пробную КТ. Каждый процесс извещает P_i о том, сделал ли он пробную КТ. Если все процессы сделали пробные контрольные точки, то P_i превращает пробные точки в постоянные. Если какой-либо процесс не смог сделать пробную точку, все точки отменяются.

2 фаза. P_i информирует все процессы о своем решении.

Восстановление: 1 фаза. Инициатор отказа спрашивает остальных, готовы ли они откатываться. Когда все будут готовы к откату, откат.

2 фаза. P_i сообщает всем о принятом решении. Получив это сообщение, каждый процесс поступает указанным образом. С момента ответа на опрос готовности и до получения принятого решения процессы не должны посыпать сообщения.

3. Асинхронная фиксация. Множество контрольных точек может быть неконсистентным. При отказе происходит поиск подходящего консистентного множества путем поочередного отката каждого процесса в ту точку, в которой зафиксированы все посланные им и полученные другими сообщения.

Репликация - механизм синхронизации содержимого нескольких копий объекта

Рассмотрим возможные протоколы работоспособности коммуникаций процессоров:

• Протокол принятия единных решений.

Все исполнители являются исправными и должны либо все принять, либо все не принять заранее предусмотренное решение.

• Протокол принятия согласованных решений.

Наработка принятия решения при надежных коммуникациях, но ненадежной работе процессоров. В системе с n ненадежно работающими процессорами можно достичь согласия только при наличии $2m + 1$ верно работающих процессоров (более 2/3) (задача Византийских генералов). Предположим, что коммуникации надежны, а процессоры нет.

Алгоритм надежных неделимых широковещательных рассылок сообщений:

Фаза 1 Процесс-отправитель посыпает сообщение группе процессов (список их идентификаторов содержится в сообщении). Процессы приписываются сообщению приоритет и помещают в очередь (как недоставленное), информируют отправителя.

Фаза 2 Отправитель получил все ответы => выбирает максимальный полученный приоритет и присваивает его сообщению, рассыпает всем процессам. Получатели принимают, помещают сообщение как доставленное, упорядочивают сообщения в своих очередях.

Если получатель обнаружит, что он имеет сообщение с пометкой <недоставленное>, отправитель которого сломался, то он для завершения выполнения протокола осуществляет следующие два шага в качестве координатора: опранивает всех о статусе сообщения; получив все ответы, реагирует на них. Если сообщение у какого-то получателя помечено как <доставленное>, то его окончательный приоритет рассыпается всем. Получив это сообщение каждый процесс выполняет шаги фазы 2. Иначе координатор заново начинает весь протокол с фазы 1.

[Курс распределенных систем]

0.0.55 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 процедуральными 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.

Java

```
(Integer x, Integer y) -> x + y
```

или

```
(Integer x, Integer y) -> return x + y;
```

— лямбда-выражения. Типами параметров лямбда-выражений могут быть только объективные типы, тип возвращаемого значения выводится из возвращаемых выражений.

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 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

Целые 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 возникнуть 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:

- $(-1)^S \cdot M \cdot p^P$, 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.
- Главный проблемой символьного типа является выбор кодировки. Современное решение — Unicode.

Логические типы: в C отсутствует, в C++ можно преобразовывать к целому, в C#, Java — нет.

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

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

Базовый блоком (ББ) или линейным участком называется последовательность следующих одна из другой инструкций МИР, 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.

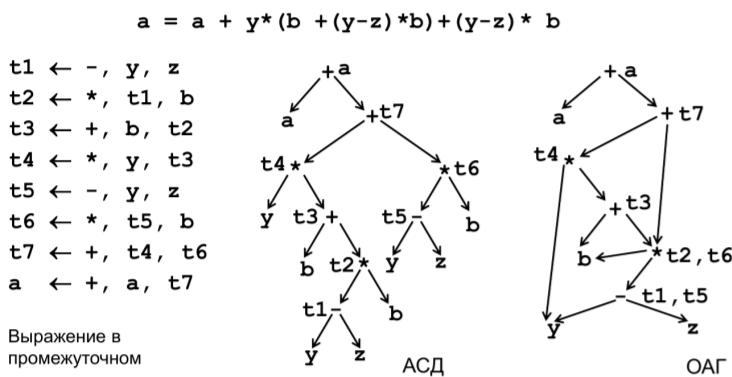
Локальная оптимизация — это оптимизация, которая выполняется в пределах одного базового блока (ББ). Возможные локальные оптимизации:

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/констант (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
```

	(a) Блок Е до оптимизации	(б) Блок Е после нумерации значений	(в) Блок Е после оптимизации
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 КОП # операнда # операнда Присоединенные переменные			
	Определение значения (сигнатура)		

При нумерации значений (и восстановлении базового блока из ОАГ) автоматически получаем **удаление общих подвыражений**. Для значений, которым соответствуют 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.

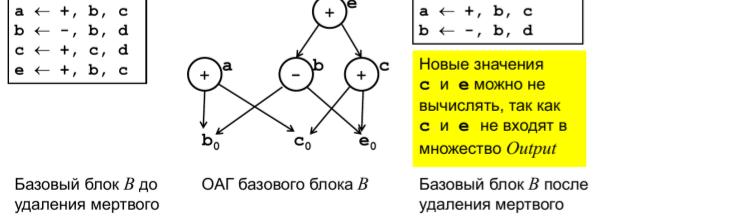
Удаление мертвого кода — более сложная оптимизация, которая требует знаний о других блоках. Для описания алгоритма расширим понятие базового блока

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

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 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 перем. 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) метод разбиения задачи на подзадачи
- 2) 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 допустимых значений переменной x . В основе МВГ лежит следующая идея: если нижняя граница значений функции на подобласти A дерева поиска больше, чем верхняя граница на подобласти B , то A может быть исключена из дальнейшего рассмотрения (закрыт). Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

Основная задача линейного программирования (озЛП) (c_1, c_2, \dots, c_n), найти

$$\max_{x \in R, Ax \leq b} (c, x).$$

Каноническая задача ЛП:

$$\max_{x \in Z, Ax \leq b} (c, x).$$

Формально данные задачи не являются дискретными, но они могут быть сведены к перебору конечного числа угловых точек (вершин) полиздара, задающего ограничения) на основании принципа граничных решений:

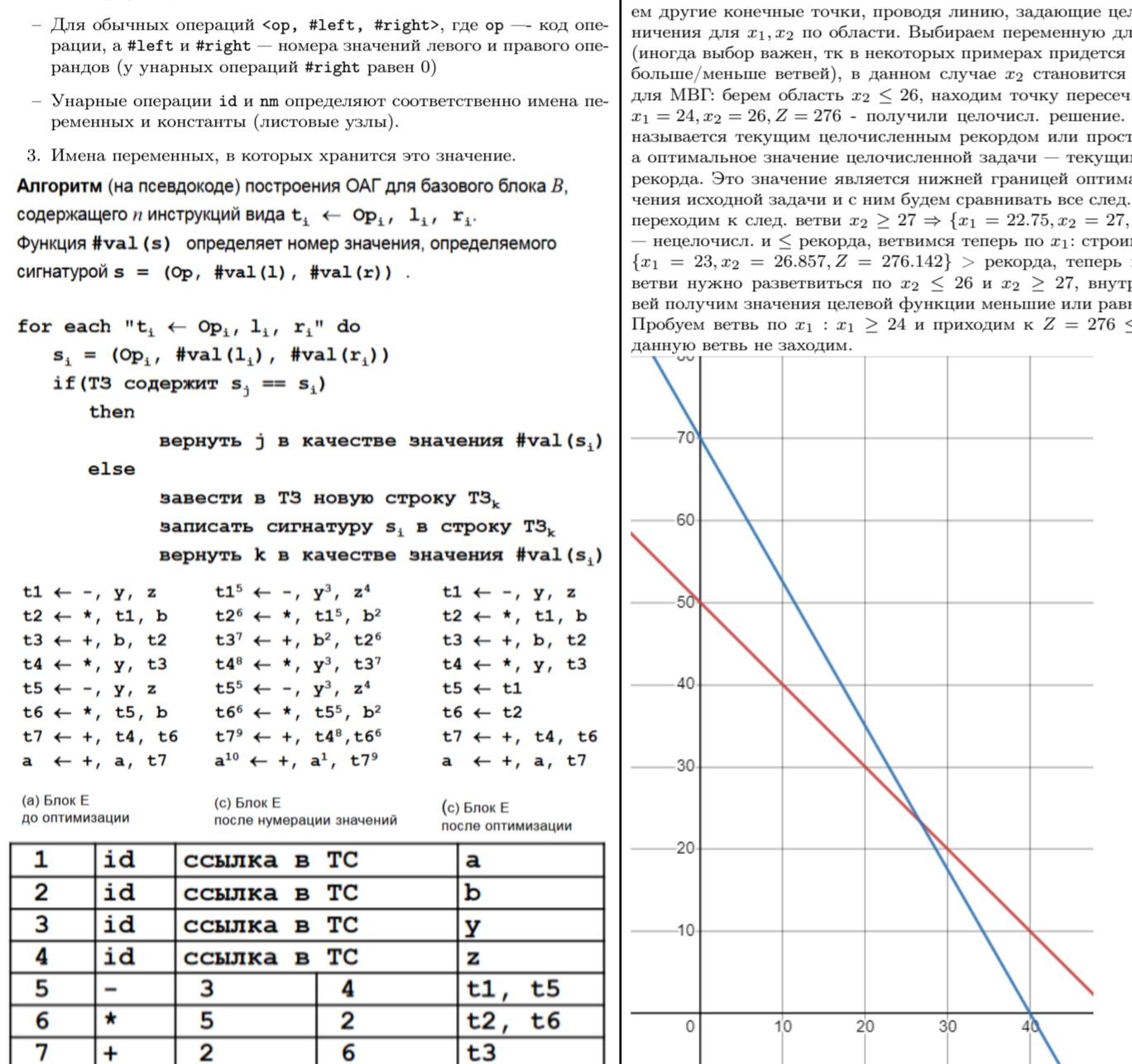
Если задача имеет решение, то найдется такая подматрица A_I матрицы A , что любое решение системы уравнений $A_I x = b_I$ реализует максимум.

Для задачи целочисленного ЛП на переменные накладывается требование их принадлежности мн-ву целых чисел:

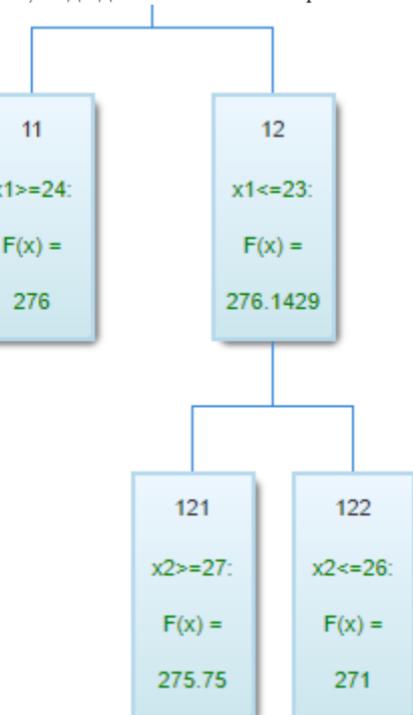
$$\max_{x \in Z, Ax \leq b} (c, x).$$

Пример: Найти $\max Z = 5x_1 + 6x_2$ при ограничениях $x_1 + x_2 \leq 50$, $4x_1 + 7x_2 \leq 280$, $x_1, x_2 \geq 0$ — целые

Общая схема МВГ для примера: строим области ограничения $x_1 \leq -x_2 + 50$ и $x_1 \leq -\frac{7}{4}x_2 + 70$, точка пересечения $x_1 = \frac{70}{3} = 23.33$, $x_2 = \frac{80}{3} = 26.67$, $Z = 5 * \frac{70}{3} + 6 * \frac{80}{3} = 276.667$ — не явл.целочисл. Решение лежит внутри области, заданной ограничениями. Мы проверяем другие конечные точки, проводя линию, задающие целочисленные ограничения для x_1, x_2 по области. Выбираем переменную для ветвления (иногда выбор важен, тк в некоторых примерах придется рассматривать больше/меньше ветвей), в данном случае x_2 становится параметром для МВГ: берем область $x_2 \leq 26$, находим точку пересеч. с границей $x_1 = 24$, $x_2 = 26$, $Z = 276$ — получили целочисл. решение. Сама точка называется текущим целочисленным рекордом или просто рекордом, а оптимальное значение целочисленной задачи — текущим значением рекорда. Это значение является нижней границей оптимального значения исходной задачи и с ним будем сравнивать все след. значения Z, переходом к след. ветви $x_2 \geq 27 \Rightarrow \{x_1 = 22.75, x_2 = 27, Z = 275.75\}$ — нецелочисл. и \leq рекорда, ветвимся теперь по x_1 : строим $x_1 \leq 23 \Rightarrow \{x_1 = 23, x_2 = 26.857, Z = 276.1429\} >$ рекорда, теперь внутри этой ветви нужно развернуться по $x_2 \leq 26$ и $x_2 \geq 27$, внутри этих ветвей получим значения целевой функции меньше или равные рекорду. Пробуем ветвь по $x_1 : x_1 \geq 24$ и приходим к $Z = 276 \leq$ рекорда, в данную ветвь не заходим.



Ограничения, скда добавляем также ограничение текущей ветви



Ветвление по x_1

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

0.0.59 Потоки в сетях. Алгоритм построения максимального потока. Оценка сложности алгоритма.

- **Потоковая сеть** — набор $G = (V, E, c, s, t)$, где (V, E) — орграф без множественных дуг (в частности, $(u, v) \in E \implies (v, u) \notin E$), c — вектор пропускных способностей дуг, $|E| = |V|$, вершины s, t — исток и сток соответственно. Для каждой вершины u обозначим:

$$u_+ = \{v \in V \mid (u, v) \in E\}$$

$$u_- = \{v \in V \mid (v, u) \in E\}$$

- f — поток в G , т.е. вектор, $|f| = |E|$, для которого выполнены условия:

$$1. f(e) \in [0, c(e)], e \in E$$

$$2. \forall u \in V \setminus \{s, t\}, f(u) = \sum_{v \in u_+} f(u, v) = \sum_{v \in u_-} f(v, u) = f_-(u).$$

- $\|f\| = f_+(s)$ — величина потока в сети.

- **Задача поиска максимального потока:** Найти поток f максимальной величины при условиях 1., 2.

Алгоритм Форда-Фалкерсона

Инициализация. Для каждой дуги $(u, v) \in E$ добавим обратную дугу (v, u) , сделав на ней соответствующую пометку. Положим изначально $f(e) = 0 \forall e \in E$.

Шаг алгоритма. Пусть имеется некоторый вычисленный поток $f(e) \forall e \in V$. Определим "остаток пропускной способности" следующим образом: для прямых дуг $r(u, v) = c(u, v) - f(u, v)$, для обратных $r(v, u) = f(v, u)$. Попытаемся найти какой-либо путь p из s в t , в котором у всех дуг остаток пропускной способности положителен. Если такого пути нет, то максимальный поток построен, и алгоритм завершается. Если он нашелся то вычислим остаток пропускной способности: $r(p) = \min\{r(e) \mid e \in p\}$, и обновим поток для каждой дуги в p — если дуга (u, v) примыкает, то $f(u, v) = f(u, v) + r(p)$, если дуга (v, u) обратная, то $f(u, v) = f(u, v) - r(p)$.

Алгоритм гарантировано сходится при условии целочисленности пропускной способности для каждой дуги. Для каждого найденного пути величина потока увеличивается хотя бы на 1, алгоритм поиска пути может быть любым, но, например, можно найти его при поиском в ширину со сложностью $O(|E|)$, и таким образом сложность алгоритма будет составлять $O(\|f\| + |E|)$.

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

Метод разделения переменных для решения первой краевой задачи.

$$\begin{cases} u_t(x, t) = a^2 u_{xx}(x, t) + f(x, t), &$$

0.0.61 Комбинаторные методы нахождения оптимального пути в графе.

Определения

- Взвешенный орграф $G = (V, E, c)$ называется *сетью* (c — функция, определяющая вес ребра).
- Ориентированный 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 Беллмана-Форда. Сложность $\mathcal{O}(|V| + |E|)$.

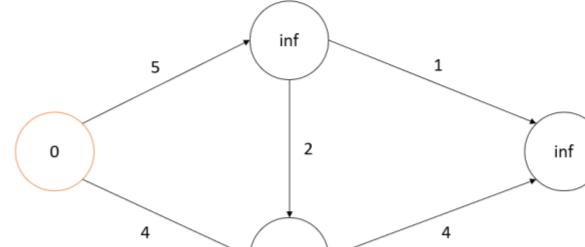
Пусть мы хотим найти длину кратчайшего пути из s в остальные вершины. Обозначим через a_{kp} длину кратчайшего пути из s в p , состоящего из k дуг, или ∞ , если такого пути не существует.

Инициализация. $a_{0s} = 0$, $a_{0p} = \infty$ для всех вершин $p \neq s$.

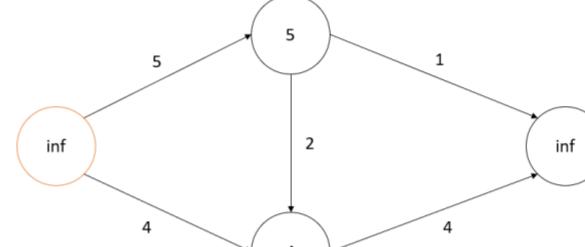
Шаг алгоритма. Зная все минимальные длины путей из $k - 1$ дуг, посчитать минимальную длину пути из k дуг можно, перебрав все вершины, из которых имеются дуги, идущие в данную. Для всех вершин p : $a_{kp} = \min\{a_{k-1,p} + c(p', p)\} \mid p' \in V, (p', p) \in E\}$.

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: $a_{ns} = \min\{a_{ns}, a_{kp}\}$.

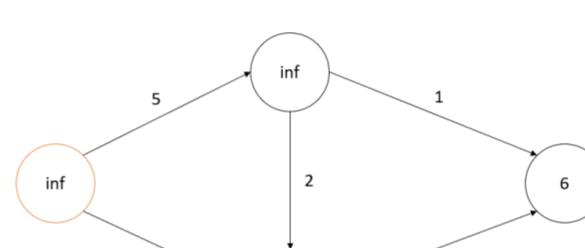
K = 0



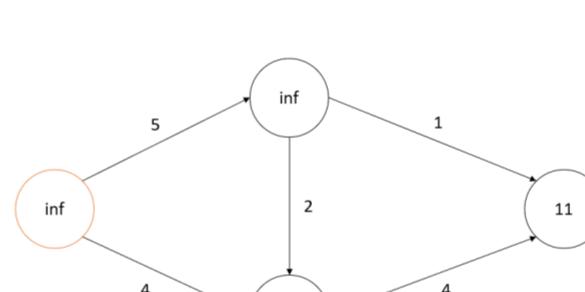
K = 1



K = 2



K = 3



Алгоритм Дейкстры. Сложность варьируется (см. ниже).

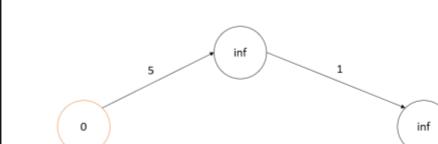
Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до s . Алгоритм работает пошагово — на каждом шаге он посещает одну вершину и пытается уменьшить метки.

Работа алгоритма. Алгоритм завершается, когда все вершины посещены. Инициализация. Метка самой вершины s полагается равной 0, метки остальных вершин — ∞ . Это отражает то, что расстояния от s до других вершин пока неизвестны. Все вершины графа помечаются как непосещенные.

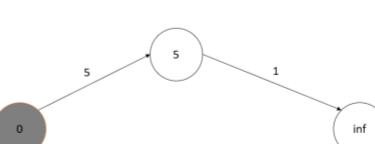
Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещенных вершин выбирается вершина u , имеющая минимальную метку. Обновим метки соседних с u вершин: $\forall s : (u, s) \in E : \text{label}_s = \min\{\text{label}_s, \text{label}_u + c(u, s)\}$. Пометим вершину u посещенной.

Сложность алгоритма варьируется в зависимости от того как хранить множество непосещенных вершин для удобства получения вершины с минимальной веткой. При хранении множества как списка, упорядоченного по убыванию меток, сложность алгоритма составляет $\mathcal{O}(|V|^2)$. При использовании двойичной кучи сложность уменьшается до $\mathcal{O}(|E| + |V| \log |V|)$.

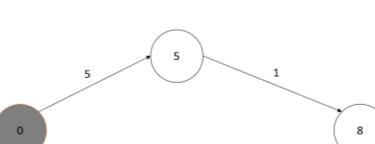
Начальное состояние



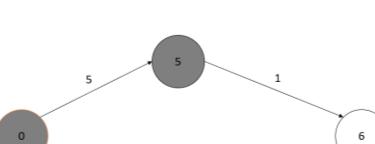
Первый шаг



Второй шаг



Третий шаг



Четвертый шаг:
+ серенький последний круг

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

0.0.62 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.

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: состояния в точке программы p_j перед инструкцией I_j (входным состоянием, $In[I_j]$) и состоянием в точке программы p_{j+1} после инструкции (выходным состоянием, $Out[I_j]$).

• 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.: $Out[I_j] = f_{I_j}(In[I_j])$ при прямом обходе и $In[I_j] = f_{I_j}^b(Out[I_j])$ при обратном.

• Для ББ B из инструкций I_1, \dots, I_n по определению $In[B] = In[I_1], Out[B] = Out[I_n]$. **Передаточная функция** f_B блока B по определению равна композиции передаточных функций его инструкций: $f_B(x) = f_{I_n}(f_{I_{n-1}}(\dots f_{I_1}(x) \dots)) = (f_{I_1} \circ f_{I_2} \circ \dots \circ f_{I_n})(x)$, 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 $Out[B] = f_B(In[B])$; при обратном обходе: $In[B] = f_B^b(Out[B])$.

(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 убивает все предыдущие определения u , а потом порождает d — новое определение u . Следовательно, $f_I(x) = gen_I \cup (x - kill_I)$, где x — состояние во входной точке инструкции I .

Пусть базовый блок B содержит n инструкций, каждая из которых имеет передаточную функцию $f_i(x) = gen_i \cup (x - kill_i)$, $i = 1, 2, \dots, n$. Тогда передаточная функция для базового блока B может быть записана как $f_B(x) = gen_B \cup (x - kill_B)$, где $kill_B = kill_1 \cup kill_2 \cup \dots \cup kill_n$, и $gen_B = gen_1 \cup (gen_{n-1} - kill_n) \cup \dots \cup (gen_1 - kill_2 - kill_3 - \dots - kill_n)$. Таким образом, для каждого базового блока B_i можно выписать уравнение: $Out[B_i] = f_{B_i}(In[B_i])$.

В случае анализа достигающих определений: $Out[B_i] = gen_{B_i} \cup (In[B_i] - kill_{B_i})$.

$Pred(B)$ — множество всех вершин ГПУ, которые непосредственно предшествуют вершине B . Следовательно, $In[B_i] = \bigcup_{P \in Pred(B_i)} Out[P]$.

Произведя подстановку, получим систему уравнений:

$$In[B_i] = \bigcup_{P \in Pred(B_i)} (gen_P \cup (In[P] - kill_P)), i = 1, 2, \dots, n$$

Монотонные и дистрибутивные передаточные функции

• **Полурешетка** это множество L , на котором определена бинарная операция «сбор» \wedge , такая, что $\forall x, y, z \in L$:

$$- x \wedge x = x \quad (\text{идемпотентность})$$

$$- x \wedge y = y \wedge x \quad (\text{коммутативность})$$

$$- x \wedge (y \wedge z) = (x \wedge y) \wedge z \quad (\text{ассоциативность})$$

• Для всех пар $x, y \in L$ определим отношение \leqslant : $x \leqslant y$ 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**: $\langle Forward, GK, Def, \cup \rangle$, где GK — семейство передаточных функций вида $gen-kill$, Def — множество определений переменных.

• Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **монотонной**, если $\forall x, y \in L, \forall f \in F (x \leqslant y) \Rightarrow f(x) \leqslant f(y)$.

• Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **монотонной** (определение эквивалентно предыдущему), если $\forall x, y \in L, \forall f \in F (x \leqslant y) \leqslant f(x) \leqslant f(y)$.

• Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **дистрибутивной**, если $\forall x, y \in L, \forall f \in F f(x \wedge y) = f(x) \wedge f(y)$.

Метод неподвижной точки и его применение для поиска достигающих определений

(Вобщем, никто не называет это методом неподвижной точки, но видимо имелось ввиду это.)

Общий итеративный алгоритм решения задачи анализа потока данных: $\langle D, F, L, \wedge \rangle$, передаточная функция $f_B \in F$, константа $v \in L$ для граничного условия.

Выход: значения из L для $In[B]$ и $Out[B]$ для каждого блока B в графике потока.

1) $OUT[\text{ВХОД}] = v_{\text{ВХОД}}$;

2) **for** (каждый базовый блок B , отличный от входного) $OUT[B] = \top$;

3) **while** (внесены изменения в OUT)

4) **for** (каждый базовый блок B , отличный от входного) {

5) $IN[B] = \Lambda_P - \text{предшественник } B \text{ OUT}[P]$;

6) $OUT[B] = f_B(IN[B])$;

}

а) Итеративный алгоритм для прямой задачи потока данных

1) $IN[\text{ВЫХОД}] = v_{\text{ВЫХОД}}$;

2) **for** (каждый базовый блок B , отличный от выходного) $IN[B] = \top$;

3) **while** (внесены изменения в IN)

