Алгоритмы и анализ сложности, 3 семестр ПИ, Ответы на билеты

Собрано 27 декабря 2022 г. в 16:30

Содержание

Раздел #1: Вопрос 1

1.1. Машины Тьюринга и тезис Чёрча

Определение 1 (Машина Тьюринга). Да что за «Машина Тьюринга»?

- Абстрактная вычислительная машина.
- Формализация понятия алгоритма.
- Расширение конечного автомата.
- Лента (бесконечная).
- Головка записи-чтения (управляющее устройство), способная находиться в одном из множества состояний, которое конечно и точно задано.
- Это управляющее может перемещаться влево и вправо по ленте, читать и записывать в ячейки символы некоторого конечного алфавита.
- Существует ϵ-символ, который заполняет все пустые клетки ленты.
- Управляющее устройство работает согласно правилам перехода, которые представляют алгоритм, реализуемый данной МТ. Каждое правило перехода предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей ячейке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо (существует некий «синтаксический сахар» — остаться на месте). Некоторые состояния могут быть помечены как терминальные, и переход в любое из них означает конец работы, остановку алгоритма.

Это все очень интересно, но как насчет формализма?
Формально: $M = \{Q, G, \epsilon, \Sigma, \delta, q_0, F\}$ — запомните этот набор из семи элементов!

- Q — конечное, не являющееся пустым, множество состояний.
- G — конечный, не являющийся пустым, набор символов ленточного алфавита.
- ϵ — единственный пустой символ.
- $\Sigma = G \backslash \{\epsilon\}$ — набор входных символов.
- $\delta : (Q \backslash F) \times G \nrightarrow Q \times G \times \{L, R\}$ — частичная функция, называемая функцией перехода, где L — сдвиг влево, а R — сдвиг вправо. Если δ не определена для текущего состояния и символа ленты, то машина останавливается.
- q_0 — это начальное состояние.
- $F \subset Q$ — набор конечных состояний.

Замечание. А что если мы хотим такое состояние, которое будет являться term при одном символе на ленте и nonterm при другом символе?
Если немного подумать, то здесь все в порядке, поскольку мы можем сделать это состояние nonterm, перейти в другое term состояние, а при определенном символе сдвинуться, например, вправо, записав на ленту такой же символ, что был на ней.

Замечание. Хоть любой конечный алфавит и не ограничен одними цифрами 0 и 1, очевидно, что мы всегда можем его представить в виде двоичных чисел, введя на нем порядок.

Определение 2 (Частичная функция). частичная функция f из множества X в множество Y — это функция из подмножества S из X (возможно, самого X) в Y (обозначение: $\nrightarrow$).

Определение 3 (Детерминированная и недетерминированная машины Тьюринга). Машина Тьюринга называется детерминированнй, если каждой паре состояни и ленточного символа соответствует не более одного правила. В ином случае, машина является недетерминированной.

Определение 4 (Тезис Черча/Тьюринга/Черча-Тьюринга). Есть ли отличие?
На самом деле, все они говорят об одном, просто Черч в свое время придумал λ-исчисления, Тьюринг придумал Машину Тьюринга, а позже было показано, что эти формализмы эквивалентны.
Сам тезис сформулируем следующим образом: Класс алгоритмически вычислимых частичных функций совпадает с классом всех функций, вычислимых на машине Тьюринга.

Замечание. Стоит понимать, что это именно тезис, а не теорема, ведь понятие «алгоритмически вычислимая частичная функция» неформально.

Определение 5 (Вычислимая функция). Вычислимые функции — это множество функций вида, $f\colon N \to N$, которые могут быть реализованы на машине Тьюринга.
В качестве множества N обычно рассматривается множество B^* — множество слов в двоичном алфавите $B = \{0, 1\}$, с оговоркой, что результатом вычисления может быть не только слово, но и специальное значение «неопределённость», соответствующее случаю, когда алгоритм «зависает». Таким образом, можно дать следующее определение N:
$N = B^* \cup \{\text{undef}\}$, где $B = \{0, 1\}$, а undef — специальный элемент, означающий неопределённость.
Роль множества N может играть множество натуральных чисел, к которому добавлен элемент undef, и тогда вычислимые функции — это некоторое подмножество натуральнозначных функций натурального аргумента. Удобно считать, что в качестве N могут выступать различные счётные множества — множество натуральных чисел, множество рациональных чисел, множество слов в каком-либо конечном алфавите и др.

1.2. Классы RE, R и co-RE, доказательство R = RE ∩ co-RE

Определение 6 (Классы RE и co-RE). Класс RE (recursively enumerable) — класс decision problems (проблемы принятия решения), для которых ответ «да» может быть проверен машиной Тьюринга за конечное время.

- Если на проблему ответ «да», то существует некоторая процедура, которая требует конечного времени для определения этого.
- Ложь здесь отсутствует.
- Если на проблему ответ «нет», то машина Тьюринга может остановиться, а может и не остановиться.

Класс co-RE является дополнением к классу RE: ответ «нет» можно получить за конечное время абсолютно всегда, получение противоположного ответа может занять вечность.

Определение 7 (Формальный язык). Формальный язык (или просто язык) — это множество конечных слов над конечным алфавитом.

Определение 8 (Класс R). R — класс decision problems, решаемых на MT (набор всех рекурсивных языков).

Определение 9 (Рекурсивный язык). Формальный язык является рекурсивным, если существует полная машина Тьюринга (машина Тьюринга, которая останавливается для каждого заданного ввода), которая, когда на вход подается конечная последовательность символов, принимает ее, если она принадлежит языку, и отвергает ее в противном случае.

Теорема 1. R = RE ∩ co-RE.

Доказательство. Обозначим за X класс decision problems, содержащихся в классе RE, ответы «да» и «нет» на которые можно получить за конечное время. Очевидно, что $X \subset$ co-RE по определению, а также никакая другая задача, содержащаяся в RE не содержится в co-RE. То есть, X = RE ∩ co-RE. Однако X = R, так как это в точности класс decision problems, решаемых на машине Тьюринга, то есть: R = X = RE ∩ co-RE. □

1.3. Проблемы Acceptance, Halting, Emptiness

Определение 10 (Halting Problem). Имеется произвольная машина Тьюринга и произвольные входные данные для нее. Вопрос заключается в следующем: возможно ли разработать алгоритм, который точно скажет, останавливается ли эта машина Тьюринга или нет?

Теорема 2. Проблема останова является алгоритмически неразрешимой задачей.

Доказательство. Доказательство этой проблемы достаточно просто объяснил А.В. Спивак в следующем видео: ссылка. □

Определение 11 (Acceptance problem). Имеется произвольная машина Тьюринга и произвольные входные данные для нее. Вопрос заключается в следующем: возможно ли разработать алгоритм, который точно скажет, принимает ли эта машина Тьюринга входные данные, либо же отвергает их.

Теорема 3. Проблема принятия является алгоритмически неразрешимой задачей.

Доказательство. Пусть у нас есть $ACCEPT_{TM}$:

$$ACCEPT_{TM} = \{(< M >, s)| \text{ Машина Тьюринга } M \text{ принимает } s\}.$$

Поскольку, по предположению, $ACCEPT_{TM}$ вычислима, должна существовать некоторая машина Тьюринга A, которая ее решает. Теперь построим программу (машину Тьюринга) X, которая принимает в качестве своего единственного аргумента какую-либо МТ следующим образом:
$X(< M >)$ уходит в бесконечный цикл, только если A возвращает true на вход $(< M >, < M >)$ (иначе возвращаем true).
Если X принимает $< X >$, то A говорит, что она не принимает $< X >$. Если X не принимает $< X >$, то A опять говорит обратное. Противоречие. □

Определение 12 (Emptiness problem). Имеется произвольная машина Тьюринга M, где $L(M) \neq \varnothing$ (язык не пуст). Вопрос заключается в следующем: верно ли, что эта машина принимает хотя бы одну строку?

Теорема 4. Проблемы пустоты является алгоритмически неразрешимой задачей.

Доказательство. Оставляем доказательство данной проблемы в качестве упражнения читателям. □

Раздел #2: Вопрос 2

2.1. Совпадение классов языков, распознаваемых ДМТ, МДМТ, НМТ

TODO: Записать происходящее в теоремах лучше

Определение 13 (Детерминированная многоленточная машина Тьюринга). $M = \{Q, G, \epsilon, \Sigma, \delta, q_0, F\}$, где

- Q — конечное, не являющееся пустым, множество состояний.
- G — конечный, не являющийся пустым, набор символов ленточного алфавита.
- ϵ — единственный пустой символ.
- $\Sigma = G \backslash \{\epsilon\}$ — набор входных символов.
- $\delta : (Q \backslash F) \times G^k \nrightarrow Q \times (G \times \{L, R\})^k$, где k — количество лент, L — сдвиг влево, а R — сдвиг вправо. Если δ не определена для текущего состояния и символа ленты, то машина останавливается.
- q_0 — это начальное состояние.
- $F \subset Q$ — набор конечных состояний.

Замечание. Отличие ТМ с одной лентой и k-ТМ (ТМ с k лентами) заключается в том, что в функция перехода зависит от k значений на ленте, и переход осуществляет на k лентах сразу.

Теорема 5. Классы языков, распознаваемые детерминированными и многоленточными детерминированными машинами Тьюринга, совпадают.

Доказательство. Однолеточная ТМ — частный случай ТМ с множеством лент, поэтому нам достаточно свести ТМ с множеством лент к ТМ с одной лентой.
Шаги для сведения k-ТМ к 1-ТМ:

- Записать входные ленты k-ТМ на ленту 1-ТМ, разделив ленты $\#$.
- Пометить символы, на которые указывает каждая голова k-ТМ точной (или иным другим символом).
- Для исполнения необходимо:
  - Пройти всю ленту 1-ТМ, чтобы узнать головы.
  - Пройтись с начала ленты 1-ТМ, чтобы сделать необходимый переход.

– Если символы головки уходят вправо #, запишите пробел и сдвиг содержимого ленты.

□

Замечание. Автор полагает, что в данном способе есть недостаток, если ленты на k-ТМ бесконечны. В таком случае можно на ленту 1-ТМ записывать данные с лент k-ТМ с помощью кортежей длинны k, где каждый i кортеж будет описывать символы на i позиции на каждой ленте.

Теорема 6. Классы языков, распознаваемые детерминированными и недетерминированными машинами Тьюринга, совпадают.

Доказательство. Мы будем обходить дерево NDTM в ширину, т. к. при обходе в глубину мы бы могли попасть в ветку, в которой зависнем навсегда. Нам нужно сохранять, в какой месте дерева мы сейчас находимся. Для этого мы пронумеруем вершины по порядку на каждом уровне поиска в ширину, и будем записывать последовательность вершин, которую мы выбрали, например, 211 — выбрали вторую вершину, затем первую, затем снова первую. И в общем для перевода NDTM в DTM, нам понадобится три ленты и следующие шаги:

1. Найдём b — максимальное число недетерминированных выборов, которое мы можем сделать на любом узле (например 2).
2. Рассмотрим набор строк, содержащий все числа от 1 до b ($\sigma = \{1, 2\}$).
3. Будем проходить по набору, содержащему все возможные комбинации строк из σ $\sigma^+ = \{1, 12, 13, ..., 123, ..., 21, ..., 213, ..., b1, ...\}$ ($\sigma^+ = \{1, 2, 11, 12, 21, 22, 111, 112, ...\}$).
4. Создадим NDT, которая будет выполнять этот алгоритм, принимающая b в качестве входных данных и записывающая члены σ^+ на свою входную ленту.

Построим DTM, использующую верхний алгоритм и три ленты, где лены будут отвечать за следующее:

1. Входные данные NDTM.
2. Отвечает за симуляцию исполнения каждого выбора .
3. Отвечает за DTM, которая будет строить нам σ^+.

Как такая машина будет работать? Будем проходится по следующему алгоритму:

1. Копируем содержимое ленты 1 на ленту 3, затем берём путь, сгенерированный на ленте 3.
2. Очищаем ленты 2 и 3, а затем начинаем алгоритм сначала, пока не исследуем все возможные пути в дереве, если

(a) Закончились пути на 3 ленте.

(b) Достигли отклоняющего состояния на второй ленте.

(c) Если на 3 ленте был сгенерирован путь, которого не существует. Например, у i узла один ребёнок, а путь выглядит так $i2$.

3. Если мы встретили только отклоняющие состояние, то отклоняем входные данные, если встретили принимающее состояние, то принимаем входные данные.

□

Раздел #3: Вопрос 3

3.1. Построение универсальной машины Тьюринга

Определение 14 (Универсальная машиина Тьюринга). Универсальная машина Тьюринга — такая машина, которая может заменить собой любую машину Тьюринга. Получив на вход программу и входные данные, она вычисляет ответ, который вычислила бы по входным данным машина Тьюринга, чья программа была дана на вход.

Теорема 7. Универсальная машина Тьюринга существует.

Доказательство. Существование УМТ доказывается конструктивно, т.е. явно описывается способ ее построения. Каждая машина Тьюринга вычисляет некоторую фиксированную частично вычислимую функцию из входных строк по своему алфавиту. В этом смысле он ведет себя как компьютер с фиксированной программой. Однако мы можем закодировать таблицу действий любой машины Тьюринга в виде строки. Таким образом, мы можем построить машину Тьюринга, которая ожидает на своей ленте строку, описывающую таблицу действий, за которой следует строка, описывающая входную ленту, и вычисляет ленту, которую вычислила бы закодированная машина Тьюринга. Конкретику оставляем в качестве размышлений читателям. □

3.2. Формулировка теоремы о неразрешимости 10-й Проблемы Гильберта

Теорема 8 (10-я проблема Гильберта). Не существует алгоритма, который узнавал бы по произвольному диофантову уравнению, имеет ли оно решения в целых числах или нет.

Замечание. Больше о диофантовых уравнениях можно узнать, изучив восьмой вопрос.

3.3. Взаимное сведение 10-й проблемы Гильберта и задачи разрешимости для экзистенциальной арифметики натуральных чисел

Определение 15 (Экзистенциальная теория натуральных чисел). Экзистенциальная теория чисел из $\mathbb{N}$ — это множество всех верных утверждений вида

$$\exists X_1 ... \exists X_n : E(X_1, ..., X_n),$$

где $E(X_1, ..., X_n)$ — это формула без кванторов, в которую входят равенства и неравенства натуральных многочленов.
Предложение этой формы истинно, если можно найти значения всех переменных, которые при подстановке в формулу E делают ее истинной.

Определение 16 (Задача разрешимости для экзистенциальной теории натуральных чисел). Задача разрешимости заключается в ответе на вопрос о возможности истинности формулы при каких-либо значениях параметров в натуральных числах.

Теорема 9. 10-я проблема Гильберта и задача разрешимости для экзистенциальной теории натуральных чисел, в которой есть только проверка на равенство, являются взаимно сводимыми.

Доказательство. Доказательство в левую сторону: поскольку любой общий алгоритм, который может решить, имеет ли данное диофантово уравнение целочисленное решение, может быть модифицирован в алгоритм, который решает, имеет ли данное диофантово уравнение решение с натуральными числами, и наоборот. Это можно увидеть следующим образом: требование, чтобы решения были натуральными числами, может быть выражено с помощью теоремы Лагранжа о четырех квадратах: каждое натуральное число является суммой квадратов четырех целых чисел, поэтому мы просто заменяем каждый параметр суммой квадратов четырех дополнительных параметров. Представив уравнения в натуральных числах, оно легко интерпретируется в виде 10-й проблемы Гильберта.
Доказательство в правую сторону: запишем все левые части уравнения в экзистенциальной теории натуральных чисел в качестве их суммы квадратов (предварительно убедившись, что справа остались нули). Таким образом, диофантово уравнение будет верно тогда, и только тогда, когда все суммы обращаются в ноль. □

Раздел #4: Вопрос 4

4.1. Совпадение RE с классом языков, распознаваемых с помощью counter machines с 4 счётчиками

Теорема 10. RE совпадает с классом языков, распознаваемых с помощью counter machines с 4 счётчиками.

Доказательство. Шаг 1: Машина Тьюринга может быть смоделирована двумя стеками.
Машина Тьюринга состоит из конечного автомата и бесконечной ленты, изначально заполненной нулями, на которые машина может записывать единицы и нули. В любой момент головка чтения/записи машины указывает на одну ячейку на ленте. В этот момент эту ленту можно концептуально разрезать пополам. Каждую половину ленты можно рассматривать как стопку, где верхняя часть — это ячейка, ближайшая к головке чтения/записи, а нижняя — на некотором расстоянии от головки, при этом все нули на ленте находятся за нижней частью. Соответственно, машину Тьюринга можно смоделировать с помощью конечного автомата, а добавив к нему два стека. Перемещение головки влево или вправо эквивалентно извлечению бита из одного стека и перемещению его в другой. Запись эквивалентна изменению бита перед его отправкой.
Шаг 2: Стек может быть смоделирован двумя счетчиками.
Стек, содержащий нули и единицы, может быть смоделирован двумя счетчиками, когда биты в стеке рассматриваются как представляющие двоичное число (самый верхний бит в стеке является наименее значащим битом). Помещение нуля в стек эквивалентно удвоению числа. Помещение единицы эквивалентно удвоению и добавлению 1. Выталкивание эквивалентно делению на 2, где остаток — это бит, который выскочил. Два счетчика могут имитировать этот стек, в котором один из счетчиков содержит число, двоичное представление которого представляет биты в стеке, а другой счетчик используется в качестве блокнота. Чтобы удвоить число в первом счетчике, мы можем инициализировать второй счетчик до нуля, затем многократно уменьшать значение первого счетчика один раз и увеличивать второй счетчик дважды. Это будет продолжаться до тех пор, пока первый счетчик не достигнет нуля. В этот момент второй счетчик будет содержать удвоенное число. Уменьшение пополам выполняется путем двукратного уменьшения одного счетчика и однократного увеличения другого, и повторяется до тех пор, пока первый счетчик не достигнет нуля. Остаток можно определить по тому, достиг ли он нуля после четного или нечетного числа шагов, где четность количества шагов закодирована в состоянии автомата.
Шаг 3: Вывод.
Мы свели Машину Тьюринга к counter machines с 4 счётчиками. Исходя из определения RE, этот класс совпадает с требуемым классом. □

Раздел #5: Вопрос 5

5.1. Определение классов сложности P, NP, co-NP

Определение 17 (Класс сложности P). Классом P называются все проблемы принятия решений, которые могут быть решены детерминированной машиной Тьюринга с использованием полиномиального количества времени вычислений или полиномиального времени.

Пример (Задача из класса сложности P). Пусть у нас есть массив натуральных чисел, состоящий из n элементов. Вопрос: содержится ли число 5 в этом массиве?
Решение: Простой перебор элементов массива (если массив упорядочен, то можно воспользоваться бинарным поиском).

Определение 18 (Класс сложности NP). Классом NP называют множество задач принятия решения, решение с ответом «да» каждой из которых можно проверить на детерминированной машине Тьюринга за время, не превосходящее какой-либо полином.

Пример (Задача из класса сложности NP). Дано число n. Вопрос: раскладывается ли данное число на три простых?
Решение: заметим, что нельзя точно утверждать, содержится ли данная задача в классе P, поскольку единственное решение, которое пока что придумано — это простой перебор, занимающий более чем полиномиальное время работы. Однако если у нас на руках существует решение данной задачи с ответом «да», то мы легко можем проверить данное решение, перемножив три числа, содержащихся в решении. Если перемножение дает верный ответ, то решение верно, и наоборот. Доказательство того, что проверка занимает не более чем экспоненциальное время оставим в качестве упражнения читателям.

Замечание. На самом деле, решение прошлой задачи с ответом «нет» тоже можно проверить за экспоненциальное время. Само решение будет состоять в том, чтобы показать, что число раскладывается не на три простых, а на какое-либо другое количество. Проверка решения аналогична: перемножить и убедиться, либо же опровергнуть корректность решения. Это значит, что описанная нами задача также принадлежит и классу co-NP, о котором сказано ниже.

Замечание. Очевидно: P $\subset$ NP (если мы можем решить задачу за полиномиальное время, то мы можем проверить решение задачи, просто решив ее).

Определение 19 (Класс сложности co-NP). Классом co-NP называют множество задач принятия решения, дополнение к которому лежит в классе NP. Это означает, что каждая задача, решение которой с ответом «нет» можно проверить на детерминированной машине

Тьюринга за время, не превосходящее какой-либо полином, лежит в классе co-NP.

5.2. Определение полиномиальной сводимости и класса NP-полных языков

Определение 20 (Полиномиальная сводимость). Любой язык L_1 называется сводимым по Карпу к языку L_2, если существует функция $F: \Sigma^* \mapsto \Sigma^*$, вычисляемая за полиномиальное время, где $F(x)$ принадлежит L_2 в том случае, если x принадлежит L_1.

Определение 21 (Класс сложности $NP\text{-}complete$). Класс NP-complete — множество задач принятия решения из класса NP, к каждой из которых можно свести **любую** другую задачу из этого класса за полиномиальное время.

Замечание. Найдя алгоритм для решения любой задачи из класса NP-complete за полиномиальное время, возможно решать каждую задачу NP за полиномиальное время, а это решает проблему P = NP.

5.3. Взаимосвязи этих классов

Замечание. Взаимосвязи:
P $\subset$ NP;
P $\subset$ co-NP;
NP $\cap$ co-NP $\neq \varnothing$.

5.4. Доказательство того, что если NP-полный язык лежит в co-NP, то NP = co-NP

Теорема 11. Если NP-полный язык лежит в co-NP, то NP = co-NP.

Доказательство. Пусть L — это язык из класса co-NP. Заметим, что дополнение к этому языку лежит в классе NP. Сведем это дополнение к NP-полному языку (за полиномиальное время), который лежит в co-NP. Получается, что дополнение к языку L лежит в классе co-NP. Данное рассуждение мы можем проделать для любого co-NP языка. Получается, что дополнения к каждой задаче лежат в co-NP, но эти дополнения по определению лежат в NP, а значит NP = co-NP. □

Раздел #6: Вопрос 7

6.1. Теорема Кука-Левина об NP-полноте задачи CNF-SAT

Определение 22 (Булева формула). Булева формула — формула логики высказываний.

Замечание. Формула называется тождественно истинной (ложной), если она истинна (ложна) при любых значениях переменных. Две булевы формулы называются эквивалентными тогда и только тогда, когда они истинны на одном и том же подмножестве множества значений аргументов.

Определение 23 (Задача SAT). Задача SAT — это задача выполнимости булевых формул. Экземпляром задачи является булева формула, состоящая только из имён переменных, скобок и операций $\wedge$ (И), $\vee$ (ИЛИ) и $\neg$ (НЕ). Задача заключается в следующем: можно ли назначить всем переменным, встречающимся в формуле, значения ложь и истина так, чтобы формула стала истинной.

Определение 24 (Задача CNF-SAT). Определение почти аналогично, однако на булеву формулу накладывается ограничение: она должна быть записана в конъюнктивной нормальной форме (должна иметь вид конъюнкции дизъюнкций литералов).

Замечание. Любая булева формула может быть приведена к КНФ.

Пример. Формулы в КНФ:
$\neg A \wedge (B \vee C)$;
$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$;
$A \wedge B$.
Формулы не в КНФ:
$\neg(B \vee C)$;
$(A \wedge B) \vee C$;
$A \wedge (B \vee (D \wedge E))$.

Определение 25 (Задача 3-SAT). Задача 3-SAT — частный случай задачи SAT, где булева формула записана в 3-конъюнктивной нормальной форме.

Определение 26 (k-КНФ). k-конъюнктивной нормальной формой называют конъюнктивную нормальную форму, в которой каждая дизъюнкция содержит ровно k литералов.

Пример. Следующая формула записана в 2-КНФ: $(A \vee B) \wedge (\neg B \vee C) \wedge (B \vee \neg C)$.

Определение 27 (Задача 1-in-3-SAT). Задача 1-in-3-SAT — частный случай задачи 3-SAT, где каждая дизъюнкция содержит три литерала, только один из которых может быть правдив (TRUE).

Теорема 12 (Теорема Кука-Левина). Задача CNF-SAT выполнимости является NP-полной. То есть она находится в NP, и любая задача в NP может быть сведена за полиномиальное время детерминированной машиной Тьюринга к булевой задаче выполнимости.

Доказательство. Доказательство теоремы Кука-Левина для задачи SAT вполне понятно описано в Википедии: ссылка. □

Замечание. Из того, что CNF-SAT является NP-complete, следует то, что SAT является NP-complete (TODO: why?).

6.2. co-NP-полнота задачи TAUTOLOGY

Определение 28 (Тавтология). Тавтологией называется формула или утверждение, которое верно во всех возможных интерпретациях.

Пример. $x \neq y \vee x = y$.

Определение 29 (co-NP-complete). Язык L называется co-NP-complete, если любой co-NP язык можно свести к этому языку за время, не превосходящее какой-либо полином.

Определение 30 (Задача TAUTOLOGY). Задача TAUTOLOGY — задача определения тавтологии в булевой формуле. Если ответ на экземпляр задачи «да», то необходимо показать, что при любых значениях литералов булева формула верна. Если ответ «нет», то достаточно продемонстрировать один контрпример.

Теорема 13. Задача TAUTOLOGY является co-NP задачей.

Доказательство. Это, на самом деле, довольно очевидно, поскольку решение с ответом «нет», вместе с которым представлен контрпример, можно легко проверить на корректность, просто подставив в булеву формулу значения литералов и убедившись в правдивости решения (в его ложности). □

Теорема 14. Задача TAUTOLOGY является co-NP-полной задачей.

Доказательство. Проблема определения того, существует ли какая-либо оценка, которая делает формулу истинной — проблема булевой выполнимости. Проблема проверки тавтологии эквивалентна этой проблеме, потому что проверка того, что предложение S является тавтологией, эквивалентна проверке того, что не существует оценки, удовлетворяющей $\neg S$. Известно, что проблема булевой выполнимости является NP-полной. Следовательно, тавтология co-NP-полна. □

Раздел #7: Вопрос 8

7.1. Доказательство NP-полноты простых задач: 3-SAT, 1-in-3-SAT, SUBSET SUM, SYSTEM OF INCONGRUENCES

Теорема 15. 3-SAT является NP-полной задачей.

Доказательство. Мы можем преобразовать любую задачу L ∈ NP в задачу CNF-SAT за полиномиальное время (Теорема Кука-Левина). Значит, нам надо доказать, что задачу CNF-SAT можно свести до 3-SAT задачи за полиномиальное время.
Пусть представление CNF исходной задачи SAT равно:

$$\phi = C_1 \ \wedge C_2 \wedge ... \wedge C_n.$$

Представление ϕ выполнимо, если все C_i выполнимы.
Без потери общности предположим, что предложение C_r содержит больше, чем 3 литерала:

$$C_r = (a_1 \vee a_2 \vee ... \vee a_m), m > 3,$$

где каждый a_i выбирается из набора литералов:

$$x_1, x_2, ..., x_m, \neg x_1, \neg x_2, ..., \neg x_m.$$

Пусть

$$A = a_3 \vee a_4 \vee ... \vee a_m;$$
$$C_r = a_1 \vee a_2 \vee A.$$

Определим C'_r следующим образом:

$$C'_r = (a_1 \vee a_2 \vee y) \wedge (\neg y \vee A),$$

где y — новая переменная, которую мы вводим, чтобы составить предложение с 3 литералами.
Эта же процедура может быть применена повторно ко второму предложению в C'_r, пока не останется предложений с более чем тремя оставшимися литералами.
Теперь нам нужно доказать, что C'_r и C_r равновероятны.
> Покажем, что если C'_r выполнимо, то и C_r выполнимо:

$$(a_1 \vee a_2 \vee y) = 1, \ (\neg y \vee A) = 1$$

Если $y = 0$:

$$C'_r = (a_1 \vee a_2 \vee 0) \wedge (1 \vee A)$$
$$= (a_1 \vee a_2) \wedge (1)$$

$$(a_1 \vee a_2) = 1$$

Очевидно, что, если C_r выполнимо, то и $(a_1 \vee a_2) = 1$ выполнимо.
Если $y = 1$:

$$C'_r = (a_1 \vee a_2 \vee 1) \wedge (0 \vee A)$$

$$(A) = 1$$

Очевидно, что, если C_r выполнимо, то и $(A) = 1$ выполнимо.
$>$ Покажем, что если C_r выполнимо, то C'_r выполнимо.
Если $a_1 = 1$ или $a_2 = 1$, то $C'_r = 1$, потому что:

$$C'_r = (a_1 \vee a_2 \vee y) \wedge (1 \vee A)$$

Если $a_1 = 0$, $a_2 = 0$, тогда $A = 1$. Назначая $y = 1$, мы можем сделать C'_r выполнимым.
Используя описанную выше процедуру, $C_r \forall r$, где количество литералов больше 3 может быть преобразовано в предложения с не более 3 чем равновероятными переменными (если литералов меньше трех, можем добавить нули в предложения, дополнив количество до трех). Из приведенного выше доказательства мы видим, что для этого требуется полиномиальное время по количеству литералов в каждом предложении. □

Теорема 16. 1-in-3-SAT является NP-полной задачей.

Доказательство. Для доказательства достаточно свести 3-SAT задачу к задаче 1-in-3-SAT за полиномиальное время.
Пусть $(x \vee y \vee z)$ будет предложением в формуле 3-SAT. Добавим 4 новые булевые переменные $a_1, ..., a_4$, которые будут использоваться для моделирования этого предложения и никаких других. Тогда формула

$$R(\neg x, a_1, a_2) \wedge R(a_2, y, a_3) \wedge R(a_3, a_4, \neg z)$$

выполнима некоторой настройкой свежих переменных тогда и только тогда, когда по крайней мере одно из значений x, y или z является истинным (иными словами, существует такой набор из 4 новых булевым переменных, чтобы все три предложения в формуле выше содержали ровно(!) 1 истинное значение). Таким образом, любой экземпляр 3-SAT с m предложениями и n переменными может быть преобразован в равноудовлетворяемый экземпляр 1-in-3-SAT с $3m$ предложениями и $n + 4m$ переменными. □

Определение 31 (Задача SSP). Задача о сумме подмножеств (SUBSET SUM PROBLEM) заключается в нахождении (хотя бы одного) непустого подмножества некоторого набора чисел, чтобы сумма чисел этого подмножества равнялась нулю.

Пример. Пусть задано множество $\{-1, -3, -2, 5, 8\}$, тогда подмножество $\{-3, -2, 5\}$ даёт в сумме ноль.

Теорема 17. Задача SSP является NP-полной.

Доказательство. TODO: https://neerc.ifmo.ru/wiki/index.php?title=NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D1%82%D0%B0_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8_%D0%BE_%D1%81%D1%83%D0%BC%D0%BC%D0%B5_%D0%BF%D0%BE%D0%B4%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B0 □

Определение 32 (Задача SYSTEM OF INCONGRUENCES). TODO

Раздел #8: Вопрос 9

8.1. NP-полнота задачи о линейных диофантовых уравнениях в N, класс задач NP-hard

Определение 33 (Класс NP-hard). Задача X является NP-сложной, если существует NP-полная задача Y, такая, которая сводится к X полиномиальному времени.

Замечание. Поскольку любая NP-полная задача может быть сведена к любой другой NP-полной задаче за полиномиальное время, NP $\cap$ NP-hard $=$ NP-complete.

Определение 34 (Диофантовы уравнения). Диофантовыми уравнениями называются уравнения в целых числах.

Теорема 18. Задача о линейных диофантовых уравнениях в N является NP-полной.

Доказательство. TODO (доказываем, что задача лежит в NP + доказываем, что задача лежит в NP-hard). □

Раздел #9: Вопрос 10

9.1. Задачи VERTEX COVER, CLIQUE и INDEPENDENT SET

Определение 35 (Вершинное покрытие). Вершинное покрытие для неориентированного графа $G = (V, E)$ — это множество его вершин S, такое, что, у каждого ребра графа хотя бы один из концов входит в вершину из S.

Определение 36 (Задача VERTEX COVER). В общем случае, задача о вершинном покрытии состоит в поиске вершинного покрытия наименьшего размера для заданного графа (этот размер называется числом вершинного покрытия графа). Примером задачи вершинного покрытия является граф $G = (V, E)$ и натуральное число k, и задача состоит в том, чтобы проверить, существует ли вершинное покрытие размера не более k в G.

Теорема 19. Задача VERTEX COVER является NP-полной задачей.

Доказательство. Докажем, что задача VERTEX COVER является NP задачей, а потом, что она является NP-трудной.
> Если какая-либо проблема находится в NP, то, учитывая «сертификат» (решение) проблемы и экземпляр проблемы (в данном случае граф G и положительное целое число k), мы сможем проверить (проверить правильность данного решения) сертификат за полиномиальное время. Сертификатом задачи покрытия вершин является подмножество V' множества V, которое содержит вершины покрытия вершин. Мы можем проверить, является ли множество V' вершинным покрытием размера k, используя следующую стратегию (для графа $G(V, E)$):
* пусть *count* будет целым числом
* установить счетчик на 0
* для каждой вершины v в V'
* удалить все ребра, смежные с v, из множества E
* увеличить счетчик на 1
* если *count* = k и E пусто, тогда данное решение верно
* иначе данное решение неверно.

Легко видеть, что это можно сделать за полиномиальное время. Таким образом, задача покрытия вершин относится к классу NP.
> Чтобы доказать, что вершинное покрытие является NP-трудным, мы возьмем некоторую задачу, которая уже доказана как NP-сложная, и покажем, что эту задачу можно свести к задаче о вершинном покрытии. Для этого мы рассмотрим задачу CLIQUE (о ней можно прочитать ниже), которая является NP-полной (и, следовательно, NP-трудной).
Примером проблемы клики является граф $G(V, E)$ и целое неотрицательное число k, и нам нужно проверить существование клики размера k в графе G.
Теперь нам нужно показать, что любой экземпляр (G, k) задачи клики можно свести к экземпляру задачи вершинного покрытия. Рассмотрим граф G', который состоит из всех

вершин G,однако мы уберем все ребра из G, а каждую пару вершин, которая не была соединена в графе G соединим в нашем новом графе. Назовем этот граф дополнением к G. Теперь задача о том, существует ли в графе G клика размера k — это то же самое, что и задача о том, существует ли вершинное покрытие размера $|V|-k$ в G'. Нам нужно показать, что это действительно так.
В одну сторону: предположим, что в G есть клика размера k. Пусть множество вершин клики равно V'. Это означает $|V'| = k$. В дополнительном графе G' выберем любое ребро (u, v). Тогда хотя бы один из u или v должен принадлежать множеству $V - V'$. Это связано с тем, что если бы и u, и v принадлежали множеству V', то ребро (u, v) принадлежало бы V', что, в свою очередь, означало бы, что ребро (u, v) принадлежит G. Это невозможно, так как (u, v) не принадлежит G. Таким образом, все ребра в G' покрываются вершинами множества $V - V'$.
В другую сторону: теперь предположим, что существует вершинное покрытие V'' размера $|V| - k$. Это означает, что все ребра в G' соединены с некоторой вершиной в V''. В результате, если мы выберем любое ребро $(u, v)G'$, оба они не могут быть вне множества V''. Это означает, что все ребра (u, v), такие что и u, и v лежат вне множества V'', принадлежат G, т. е. эти ребра составляют клику размера k.
Таким образом, можно сказать, что клика размера k в графе G существует тогда и только тогда, когда существует вершинное покрытие размера $|V|^{\smile}k$ в G', и, следовательно, любой пример задачи о кликах может быть сведен к примеру задачи о вершинном покрытии. Таким образом, вершинное покрытие является NP-сложной задачей. Поскольку вершинное покрытие относится к классам NP и NP-hard, оно является NP-complete задачей. □

Определение 37 (Задача CLIQUE). Кликой в неориентированном графе называется подмножество вершин, каждые две из которых соединены ребром графа. Иными словами, это полный подграф первоначального графа. Размер клики определяется как число вершин в ней. Задача о клике существует в двух вариантах: в задаче распознавания требуется определить, существует ли в заданном графе G клика размера k, в то время как в вычислительном варианте требуется найти в заданном графе G клику максимального размера (нас, по большей части, интересует первый вариант).

Теорема 20. Задача CLIQUE является NP-полной.

Доказательство. Докажем, что задача является NP задачей, а потом, что она является NP-трудной.
> Сертификат (решение) представляет собой подмножество вершин V', состоящее из вершин, принадлежащих клике. Мы можем проверить это решение, проверив, что каждая пара вершин, принадлежащих решению, является смежной, просто проверив, что они имеют общее ребро друг с другом. Это можно сделать за полиномиальное время, то есть $O(V + E)$, используя следующую стратегию для графа $G(V, E)$:
* $flag = true$
* Для каждой пары $\{u, v\}$ в подмножестве V':

- Проверить, что эти вершины $\{u, v\}$ имеют общее ребро.

- Если ребра нет, то $flag = false$, затем $break$.

* Если $flag = true$: Решение верно
* Иначе: Решение не является верным.
> Чтобы доказать, что задача о кликах NP-сложна, воспользуемся помощью задачи, которая уже является NP-трудной, и покажем, что эту задачу можно свести к задаче о кликах.
Для этого рассмотрим задачу о независимом множестве (INDEPENDENT SET, о которой можно прочитать ниже), которая является NP-полной (и, следовательно, NP-сложной). Каждый экземпляр задачи о независимом множестве, состоящий из графа $G(V, E)$ и целого числа K, может быть преобразован в требуемый граф $G'(V', E')$ и K' задачи клики. Построим граф G' следующими модификациями:
$V' = V$, то есть все вершины графа G являются частью графа G', $E' = \overline{E}$ (дополнение то есть ребра, отсутствующие в исходном графе G).
Граф G' является дополнительным графом G. Время, необходимое для вычисления дополнительного графа G', требует обхода всех вершин и ребер. Временная сложность: $O(V + E)$.
Теперь мы докажем, что проблема вычисления клики действительно сводится к вычислению независимого множества. Редукция может быть доказана следующими двумя утверждениями:

- Предположим, что граф G содержит клику размера K. Наличие клики означает, что в G имеется K вершин, каждая из которых соединена ребром с остальными вершинами. Это также показывает, что, поскольку эти ребра содержатся в G, они не могут присутствовать в G'. В результате эти K вершин не смежны друг с другом в G' и, следовательно, образуют независимое множество размера K.

- Предположим, что дополнительный граф G' имеет независимое множество вершин размера K'. Ни одна из этих вершин не имеет общего ребра с другими вершинами. Когда мы дополняем граф, чтобы получить G, эти K вершин будут иметь общее ребро и, следовательно, станут смежными друг с другом. Следовательно, в графе G будет клика размера K.

Таким образом, мы можем сказать, что в графе G есть клика размера K, если в G' существует независимое множество размера K (дополнительный граф). Следовательно, любой случай проблемы клики может быть сведен к примеру проблемы независимого множества. Таким образом, проблема клики является NP-трудной. □

Замечание. Размышления в прошлом доказательстве выглядят довольно массивными, однако все это достаточно очевидно, если немного подумать.

Замечание. В доказательстве NP-полноты задачи INDEPENDENT SET мы используем задачу CLIQUE, поэтому предыдущее доказательство не является совсем корректным, поскольку доказательство циклично. Поэтому докажем, что задача о кликах NP-сложна, све-

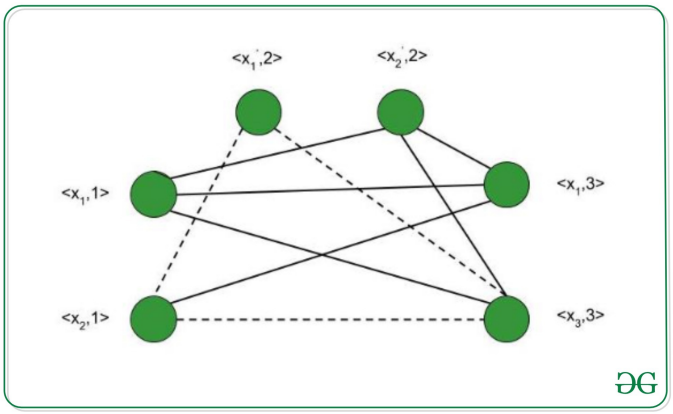

Рис. 1: Граф, построенный по указанным правилам

дя булеву проблему выполнимости к задаче решения клики.
Пусть логическое выражение будет следующим: $S = (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3)$, где x_1, x_2, x_3 — переменные. Обозначим предложения за C_1, C_2, C_3. Рассмотрим вершины как

$$< x_1, 1 >; < x_2, 1 >; < \overline{x_1}, 2 >; < \overline{x_2}, 2 >; < x_1, 3 >; < x_3, 3 >,$$

где второй член в каждой вершине обозначает номер предложения, к которому они принадлежат. Соединим эти вершины так, чтобы:

- никакие две вершины, принадлежащие одному предложению, не были связаны;
- никакая переменная (первый элемент вершины) не связана со своим дополнением (одинаковые переменные должны быть связаны).

Таким образом, граф $G(V, E)$ построен так, что:

$$V = \{< a, i > | a \in C_i\},$$

$$E = \{(< a, i >, < b, j >) | i \neq j; b \neq \overline{a}\}.$$

Рассмотрим подграф G с вершинами $< x_2, 1 >; < \overline{x_1}, 2 >; < x_3, 3 >$. Он образует клику размера 3 (рис. 1). Соответственно этому для присваивания $< x_1, x_2, x_3 >=< 0, 1, 1 >$ S принимает значение true. Следовательно, если у нас есть k предложений в нашем выражении выполнимости, мы получаем максимальную клику размера k и для соответствующего присвоения значений выражение выполнимости оценивается как истинное. Следовательно, для конкретного случая проблема выполнимости сводится к проблеме решения клики. Следовательно, проблема решения клики является NP-трудной. Стоит отметить, что мы рассмотрели частный случай, однако для любой булевой проблемы, сведенной к КНФ, это верно (это необходимо доказывать, но это напрямую вытекает из того, какие ограничения мы накладываем на ребра). Помимо этого, по-хорошему, необходимо доказать, что сведение происходит за полиномиальное время. Убедитесь в последнем самостоятельно.

Определение 38 (Задача INDEPENDENT SET). Задача INDEPENDENT SET заключается в нахождении множеств вершин в графе размера k, никакие две вершины из которых не являются смежными. Decision problem вопрос можно поставить следующим образом: существует ли множество определенного размера.

Теорема 21. Задача INDEPENDENT SET является NP-полной.

Доказательство. Доказательство принадлежности к классу NP оставляем в качестве упражнения читателям. Доказательство принадлежности к классу NP-hard легко проделать, изучив доказательство теоремы выше. Обратите внимание на замечание после предыдущей теоремы, поскольку оно важно. □

9.2. Определение FPT-алгоритмов. Примеры FPT-алгоритмов для VERTEX COVER и CLIQUE

Определение 39 (FPT-алгоритм). Параметризированная задача P считается разрешимой с фиксированным параметром, или FPT-разрешимой (Fixed-Parameter Tractable), если она может быть решена некоторым параметризированным алгоритмом за время

$$t(n,k) = O(n^{O(1)} * f(k))$$

для функции f, зависящей только от параметра k.
Порядок роста функции $f(k)$ не ограничивается. Так, возможно $f(k) = 2^{o(k)}$ или $f(k) = 2^{O(k)}$. Важно, что исключаются функции вида $f(n,k)$, например $f(n,k) = n^k$.

Замечание. Класс всех разрешимых с фиксированным параметром задач обозначается FPT. Соответствующие параметризированные алгоритмы, решающие такие задачи, называются FPT-алгоритмами.

Пример. Примером FPT-разрешимой задачи может служить задача о вершинном покрытии графа (VERTEX COVER), когда параметром выступает размер покрытия. В самом деле, вершинное покрытие размера k в графе G с n вершинами можно определить за время $O(n * 2^k)$. Если в данной задаче в качестве параметра взять древовидную ширину $tw(G)$ графа G, то метод динамического программирования способен отыскать наибольшее вершинное покрытие за время $O(n * 2^{tw(G)})$ (обход в ширину ??? TODO). Следовательно, параметризация данной задачи относительно $tw(G)$ приводит к FPT-разрешимости.

Пример. Пример с CLIQUE ??? TODO

Раздел #10: Вопрос 11

10.1. Определение классов функций FP и #P и класс языков PP

Определение 40 (Функциональная задача и класс функций FP). Функциональная задача — это вычислительная задача, в которой результат более сложный, чем у задачи принятия решения. Для функциональных проблем вывод не просто TRUE или FALSE. Разница между FP и P заключается в том, что задачи в P имеют одноразрядные ответы «да»/«нет», в то время как задачи в FP могут иметь любой результат, который может быть вычислен за полиномиальное время.

Пример. Сложение пяти чисел является проблемой FP, в то время как определение того, является ли их сумма нечетной, находится в P.

Определение 41 (Класс функций #P). #P может быть эквивалентно определен в терминах верификатора. Проблема решения находится в NP, если существует полиномиально проверяемый сертификат для данного экземпляра проблемы, то есть NP спрашивает, существует ли доказательство принадлежности для входных данных, правильность которых можно проверить за полиномиальное время. Класс #P спрашивает, сколько существует сертификатов для экземпляра задачи, корректность которых можно проверить за полиномиальное время.

Пример. Задача определения для данного графа G и числа k, содержит ли граф независимое множество размера k, находится в NP. Учитывая пару (G, k) в языке, сертификат представляет собой набор из k вершин, которые попарно несмежны (и, следовательно, являются независимым набором размера k). Проверка сертификата является решением NP задачи. Подсчет количества возможных сертификатов является задачей #P.

Определение 42 (Класс PP). PP — это класс проблем принятия решений, решаемых вероятностной машиной Тьюринга за полиномиальное время с вероятностью ошибки менее $1/2$ для всех случаев.

Определение 43 (Вероятностная машина Тьюринга). Вероятностная машина Тьюринга - это тип недетерминированной машины Тьюринга, в которой каждый недетерминированный шаг представляет собой «подбрасывание монеты», то есть на каждом шаге есть два возможных следующих хода, и машина Тьюринга вероятностно выбирает, какой ход предпринять.

10.2. Доказательство эквивалентности FP = #P и P = PP

Теорема 22. FP = #P тогда, и только тогда, когда P = PP.

Доказательство. Доказательство в правую сторону очевидно из приведенных ниже теорем.
Доказательство в левую сторону приведем далее (TODO). □

Теорема 23. Если FP = #P, то PP ⊆ P.

Доказательство. Данное доказательство содержит размышления «на пальцах». Если классы FP и #P совпадают, то мы можем находить не только решение задачи, но и находить их количество за полиномиальное время. Для последнего нам требуется проверка всех решений, поэтому мы можем найти правильное решение из PP, просто проверяя решения, пока не найдем верное. Мы можем сделать это за полиномиальное время из утверждения выше. Если мы можем найти правильное решение за полиномиальное время, то это задача также принадлежит классу P (я не уверен в том, что здесь написал: TODO). □

10.3. Доказательство NP ⊆ PP

Теорема 24. NP ⊆ PP.

Доказательство. □

Доказательство. Чтобы доказать это, покажем, что проблема NP-полной выполнимости принадлежит PP. Рассмотрим вероятностный алгоритм, который по формуле $F(x_1, x_2, ..., x_n)$ равномерно и случайным образом выбирает присваивание $x_1, x_2, ..., x_n$. Затем алгоритм проверяет, делает ли присвоение истинной формулу F. Если да, то выводит YES. В противном случае выводится YES с вероятностью

$$\frac{1}{2} - \frac{1}{2^{n+1}}$$

и НЕТ с вероятностью

$$\frac{1}{2} + \frac{1}{2^{n+1}}.$$

Если формула невыполнима, алгоритм всегда будет выводить ДА с вероятностью

$$\frac{1}{2} - \frac{1}{2^{n+1}} < \frac{1}{2}.$$

Если существует удовлетворяющее задание, оно выдаст YES с вероятностью не менее

$$(\frac{1}{2} - \frac{1}{2^{n+1}}) * (1 - \frac{1}{2^n}) = \frac{1}{2} + \frac{1}{2^{n+1}} > \frac{1}{2}$$

(ровно $1/2$, если он выбрал неудовлетворительное задание, и 1, если он выбрал удовлетворяющее задание, усреднение до некоторого числа, превышающего $1/2$). Таким образом, этот алгоритм ставит выполнимость в PP. □

Раздел #11: Вопрос 12

11.1. Замкнутость класса PP относительно дополнения

Теорема 25. PP = co-PP.

Доказательство. Пусть L — язык в PP, а L^c — его дополнение. По определению PP существует полиномиальный вероятностный алгоритм A со свойством, что

$$x \in L \Rightarrow \Pr[A \text{ accepts } x] > \frac{1}{2} \quad \text{и} \quad x \notin L \Rightarrow \Pr[A \text{ accepts } x] \leq \frac{1}{2}.$$

Будем утверждать, что без ограничения общности первое неравенство всегда строгое; из этого утверждения можно вывести теорему: пусть A^c является машиной, которая такая же, как A, за исключением того, что A^c принимает входные данные, когда A отклонила бы, и наоборот. Тогда:

$$x \in L^c \Rightarrow \Pr[A^c \text{ accepts } x] > \frac{1}{2} \quad \text{и} \quad x \notin L^c \Rightarrow \Pr[A^c \text{ accepts } x] < \frac{1}{2}.$$

что подразумевает, что L^c содержится в PP.
Утверждение, ссылающееся на «без ограничения общности», можно доказать (TODO). □

Раздел #12: Вопрос 13

12.1. Alternating Turing machine и класс AP

Определение 44 (Переменная машина Тьюринга). Переменная машина Тьюринга — это недетерминированная машина Тьюринга, состояния которой делятся на два набора: экзистенциальные состояния и универсальные состояния. Экзистенциальное состояние является принимающим, если какой-то переход приводит к принимающему состоянию; универсальное состояние является принимающим, если каждый переход приводит к принимающему состоянию. (Таким образом, универсальное состояние без переходов принимает безоговорочно; экзистенциальное состояние без переходов безоговорочно отвергает). Машина в целом принимает слово, если начальное состояние принимает слово.
Формально:
Переменная машина Тьюринга представляет собой кортеж из 5 элементов.
$M = (Q, \Gamma, \delta, q_0, g)$, где:
Q — это конечное множество состояний;
Γ — это конечный ленточный алфавит;
$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ называется функцией перехода (L сдвигает голову влево, а R сдвигает голову вправо);
$q_0 \in Q$ — это начальное состояние;
$g : Q \to \{\wedge, \vee, \text{accept}, \text{reject}\}$ определяет тип каждого состояния.
Если M находится в состоянии $q \in Q$ с $g(q) = \text{accept}$, то эта конфигурация считается принимающей, а если $g(q) = \text{reject}$, то говорят, что конфигурация отклоняется. Конфигурация $g(q) = \wedge$ считается принимающей, если все конфигурации, достижимые за один шаг, принимаются, и отклоняющей, если некоторая конфигурация, достижимая за один шаг, отклоняется. Конфигурация $g(q) = \vee$ считается принимающей, когда существует некоторая конфигурация, достижимая за один шаг, которая принимается, а отклоняющей, когда все конфигурации, достижимые за один шаг, отклоняются. Говорят, что M принимает входную строку w, если начальная конфигурация M принимается (отклоняет входную строку, если исходная конфигурация отклоняется).

Определение 45 (Класс AP). Класс AP является набором задач, решаемых с помощью переменной машины Тьюринга за полиномиальное время.

12.2. Совпадение AP и PSPACE

Определение 46 (Класс PSPACE). Класс PSPACE представляет собой набор всех задач принятия решений, которые могут быть решены машиной Тьюринга с использованием полиномиального объема памяти.

Замечание. Детерминированная машина Тьюринга может имитировать недетерминированную машину Тьюринга, не требуя гораздо большего пространства (даже если это может занять гораздо больше времени). Это значит, что рассуждать о классе PSPACE можно, используя как ДМТ, так и НМТ.

Теорема 26. AP = PSPACE.

Доказательство. Вложение AP $\subset$ PSPACE доказывается обычным образом: рекурсивный алгоритм будет работать на полиномиальной памяти. Обратное следует из теоремы о PSPACE-полноте языка TQBF (о нем написано ниже): машина сначала вычислит формулу, которая получается в результате сводимости, а затем будет выбирать набор её аргументов, чередуя $\exists$- и $\forall$-состояния. (Это все??? Не совсем очевидна первая часть доказательства. TODO http://ru.discrete-mathematics.org/fall2017/3/complexity/compl-book.pdf) $\square$

Раздел #13: Вопрос 14

13.1. AP-полнота задачи TRUE QUANTIFIED BOOLEAN FORMULAS

Определение 47 (AP-полная задача). Задача из AP называется AP-полной, если любая задача из класса AP сводится к этой задаче за полиномиальное время.

Определение 48 (TQBF задача). TQBF задача заключается в поиске переменных, которые будут удовлетворять истинной булевой формуле с кванторами, где отсутствуют свободные переменные: $TQBF = \{\phi | \phi$— булева формула с кванторами, $Free(\phi) = \varnothing, val(\phi) = 1\}$.

Пример. $\forall x(y + x = 5)$. Здесь y — свободная переменная (поэтому формула не является истинной булевой формулой с кванторами).

Пример. $\exists y : \forall x(x \vee y)$ — истинная булева формула с кванторами.

Теорема 27. TQBF задача является PSPACE-полной.

Доказательство. Докажем, что TQBF является PSPACE задачей, а затем, что она является PSPACE-полной задачей.
> Существует простой рекурсивный алгоритм для определения, находится ли QBF в TQBF (т. е. является ли формула истинной). Рассмотрим некоторую QBF: $Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$. Если формула не содержит кванторов, мы можем просто вернуть формулу. В противном случае мы удаляем первый квантор и проверяем оба возможных значения для первой переменной:

$$A = Q_2 x_2 \cdots Q_n x_n \phi(0, x_2, \ldots, x_n),$$

$$B = Q_2 x_2 \cdots Q_n x_n \phi(1, x_2, \ldots, x_n).$$

Если $Q_1 = \exists$, то возвращаем $A \vee B$.
Если $Q_1 = \forall$, то возвращаем $A \wedge B$.
Как быстро работает этот алгоритм? Для каждого квантора в исходном QBF алгоритм выполняет два рекурсивных вызова только для линейно меньшей подзадачи. Это дает алгоритму экспоненциальное время выполнения $O(2^n)$.
Сколько места занимает этот алгоритм? В каждом вызове алгоритма он должен сохранять промежуточные результаты вычисления A и B. Каждый рекурсивный вызов удаляет один квантор, поэтому общая глубина рекурсии линейна по количеству кванторов. Формулы, в которых отсутствуют кванторы, могут быть вычислены в логарифмическом пространстве по числу переменных. Начальный QBF был полностью количественно определен, так что

кванторов по крайней мере столько же, сколько переменных. Таким образом, этот алгоритм использует $O(n + logn) = O(n)$ пространства. Это делает язык TQBF частью класса сложности PSPACE.
> Чтобы показать, что TQBF является сложным для PSPACE, требуется показать, что любой язык в классе сложности PSPACE может быть сведен к TQBF за полиномиальное время. Т.е.,

$$\forall L \in \mathsf{PSPACE}, L \leq_p \text{TQBF}.$$

TODO: https://translated.turbopages.org/proxy_u/en-ru.ru.24e867e7-63a5f7a8-fc536b28-74722d776562/https/en.wikipedia.org/wiki/True_quantified_Boolean_formula □

Замечание. Из того, что TQBF задача является PSPACE-полной, следует AP = PSPACE (теорема выше). Поскольку эти классы равны, то AP-complete = PSPACE-complete (вполне очевидный факт).

Раздел #14: Вопрос 15

14.1. Полиномиальная иерархия PH

14.2. Полные языки (о булевых формулах с фиксированным числом перемен кванторов) для каждого из уровней иерархии

Раздел #15: Вопрос 16

15.1. Арифметика Пресбургера

Определение 49 (Арифметика Пресбургера). Арифметика Пресбургера — это теория первого порядка (формальное исчисление, допускающее высказывания относительно переменных, фиксированных функций и предикатов [утверждений]), описывающая натуральные числа со сложением, но в отличие от арифметики Пеано, исключающая высказывания относительно умножения.

Определение 50 (Аксиомы арифметики Пресбургера). Язык арифметики Пресбургера включает константы 0, 1, одну операцию + и предикат равенства =. Аксиомы имеют вид:

1. $\neg(0 = x + 1)$
2. $x + 1 = y + 1 \to x = y$
3. $x + 0 = x$
4. $(x + y) + 1 = x + (y + 1)$
5. $(P(0) \wedge (P(x) \to P(x + 1))) \to P(y)$, где P — формула первого порядка включающая 0, 1, +, = и одну свободную переменную x.

Следует заметить, что (5) на самом деле не одна аксиома, а схема аксиом, представляющая бесконечное множество аксиом, по одной, для каждой формулы P. (5) является формализацией принципа математической индукции. Она не может быть эквивалентно заменена никакой конечной системой аксиом. Таким образом арифметика Пресбургера не является конечно аксиоматизируемой.

Замечание. В отличие от арифметики Пеано, арифметика Пресбургера является разрешимой теорией. Это означает, что для любого предложения на языке арифметики Пресбургера можно алгоритмически определить, является ли это предложение доказуемым из аксиом арифметики Пресбургера.

Пример. Следующий пример выражения в арифметике Пресбургера (необходима элиминация кванторов) показывает, что что каждое число либо четное, либо нечетное:

$$\forall x \exists y : (y + y = x) \vee (y + y + 1 = x).$$

Замечание. Любое выражение в целых числах в арифметике Пресбургера может быть представлено в натуральных числах, поэтому можем использовать следующий кортеж: $(\mathbb{Z}; =; <; +; 0; 1)$. Такое расширение не меняет класса выразимых предикатов. (Автор не уве-

рен в том, что здесь написал: TODO).

15.2. Необходимость расширения сигнатуры

Замечание. Отметим сразу же, что с сигнатурой, указанной выше, элиминация кванторов невозможна. В самом деле, формула $\exists y(x = y + y)$; истинная для чётных x, не эквивалентна никакой бескванторной формуле. Поэтому нам нужно, прежде чем проводить элиминацию кванторов, расширить сигнатуру.

15.3. Формулировка теоремы об элиминации кванторов

Теорема 28. Приведённый пример формулы выше подсказывает, какое расширение нам необходимо — делимость по модулю (на константы): В $(\mathbb{Z}; =; <; +; 0; 1; \equiv_2; \equiv_3; \ldots)$ выполнима элиминация кванторов.

Раздел #16: Вопрос 17

16.1. Алгоритм элиминации кванторов в арифметике Пресбургера

Раздел #17: Вопрос 18

17.1. Монадические теории натуральных чисел второго порядка

Определение 51 (Логика первого порядка). Логика первого порядка — формальное исчисление, допускающее высказывания относительно переменных, фиксированных функций и предикатов.

Замечание. Кванторы в логике первого порядка могут применяться только к переменным.

Определение 52 (Логика второго порядка). Логика второго порядка является расширением логики первого порядка. Кванторы в ней могут применяться не только к переменным, но и к предикатам и функциям (также к множествам).

Замечание. Логика второго порядка несводима к логике первого порядка.

Определение 53 (Монадическая логика второго порядка). Монадическая логика второго порядка (MSO) — это ограничение логики второго порядка, в котором разрешена только количественная оценка унарных отношений (т. е. множеств). То есть, кванторы могут появляться только(!) перед множествами. Таким образом, количественная оценка функций из-за эквивалентности отношений (функция может быть определена как особый вид бинарного отношения) как описано выше, также не допускается.

17.2. Неразрешимость MSOTh $< \mathbb{N}; 0, 1, +, =, \in >$

Определение 54. В математической логике и теории алгоритмов под разрешимостью подразумевают свойство формальной теории обладать алгоритмом, определяющим по данной формуле, выводима она из множества аксиом данной теории или нет.

Теорема 29. Проблема выполнимости для монадической логики второго порядка неразрешима вообще, потому что эта логика включает логику первого порядка.

Доказательство. Получается, что необходимо доказать, что логика первого порядка неразрешима. TODO: https://studbooks.net/2305902/matematika_himiya_fizika/dokazatelstvo_nerazreshimosti_problemy_ostanovki. □

Раздел #18: Вопрос 19

18.1. Конечные автоматы

Определение 55 (Конечный автомат). Формально КА определяется в виде упорядоченной пятёрки элементов некоторых множеств:

$$A = (S, X, Y, \delta, \lambda),$$

где:
S — конечное множество состояний автомата;
X, Y — конечные входной и выходной алфавиты соответственно, из которых формируются строки, считываемые и выдаваемые автоматом;
$\delta : S \times X \to S$ — функция переходов;
$\lambda : S \times X \to Y$ — функция выходов.

Определение 56 (Автоматный язык). Автоматный язык — язык, распознаваемый некоторым конечным автоматом.

18.2. Замкнутость регулярных языков относительно различных операций

Определение 57 (Регулярный язык). Регулярный язык REG над алфавитом $\Sigma = \{c_1, c_2, ..., c_k\}$ — множество, которое может быть получено из языков, каждый из которых содержит единственное слово — c_i или ϵ, и пустого языка при помощи последовательных применений операций объединения, конкатенации или замыкания Клини и никаких других, то есть:

- Определим регулярные языки нулевого уровня как $R_0 = \{\varnothing, \{\epsilon\}, \{c_1\}, \{c_2\}, ..., \{c_k\}\}$.
- Регулярные языки ненулевого уровня определим рекуррентным соотношением: $R_{i+1} = R_i \cup \{L_1 \cup L_2, L_1 L_2, L_1^* | L_1, L_2 \in R_i\}$.
- Тогда $REG = \bigcup_{i=0}^{\infty} R_i$.

Пример. Звезда Клини для множества строк:
(“Go”, “R”)* = (ϵ, “Go”, “R”, “GoGo”, “GoR”, “RGo”, “RR”, “GoGoGo”, “GoGoR”, “GoRGo”, ...).

Теорема 30 (Теорема Клини). Классы автоматных и регулярных языков совпадают.

Доказательство. Нужно ли ее доказывать вообще? TODO: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A2%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%9A%D0%BB%

D0%B8%D0%BD%D0%B8_(%D1%81%D0%BE%D0%B2%D0%BF%D0%B0%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%BD%D1%8B%D1%85_%D0%B8_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D1%85_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2) □

Теорема 31. Пусть L_1, L_2 — регулярные языки над одним алфавитом Σ. Тогда следующие языки также являются регулярными:

1. $L_1 \cup L_2$;
2. $\overline{L_1}$.
3. $L_1 \cap L_2$;

Доказательство. Пройдемся по пунктам:

1. $L_1 \cap L_2$ является регулярным по определению регулярных языков.
2. Рассмотрим автомат $A' =< \Sigma, Q_1, s_1, Q_1 \backslash T_1, \delta_1 >$, то есть автомат, в котором терминальные и нетерминальные состояния инвертированы. Очевидно, он допускает те и только те слова, которые не допускает обычный автомат, а значит, задаёт язык $\overline{L_1}$. Таким образом, $\overline{L_1}$ — регулярный.
3. $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ — регулярный.

□

18.3. Совпадение классов языков, распознаваемых ДКА и НКА

Теорема 32. Для всякого недетерминированного конечного автомата можно построить детерминированный конечный автомат, задающий точно такой же язык.

Доказательство. Вершины ДКА — это подмножества множества вершин исходного НКА. Чтобы получить вершину ДКА (т.е. множество вершин исходного НКА), в которую осуществляется переход по данному символу из данной вершины ДКА (т.е. из множества M вершин НКА), надо объединить все вершины НКА, которые получаются всевозможными переходами по данному символу из всех вершин множества M, а затем добавить также вершины, достижимые с помощью переходов по пустой цепочке. Начальная вершина ДКА – это объединение всех начальных вершин НКА плюс вершины, достижимые из начальных с помощью переходов по пустым цепочкам. Конечные вершины ДКА – это подмножества вершин исходного НКА, содержащие хотя бы одну конечную вершину.
Число вершин ДКА, вообще говоря, экспоненциально зависит от числа вершин исходного НКА.
Таким образом, классы языков, задаваемых детерминированными и недетерминированными конечными автоматами, совпадают, и можно говорить просто об автоматных языках, т.е. языках, задаваемых конечными автоматами. □

18.4. Разрешимость проблем Emptiness и Equality

Определение 58 (Проблема Emptiness для конечного автомата). В общем случае проблему распознавания того, что язык конечного автомата пуст, называют проблемой пустоты для конечного автомата.

Теорема 33. Проблема Emptiness для конечного автомата является разрешимой.

Доказательство. Чтобы решить эту проблему, достаточно найти множество заключительных состояний автомата, достижимых из начального состояния. Так как конечный автомат — это ориентированный граф, то решить такую проблему можно, например, с помощью, поиска в ширину. Язык, допускаемый конечным автоматом, пуст тогда и только тогда, когда множество заключительных состояний, достижимых из начального состояния, пусто. □

Определение 59 (Проблема Equality для конечных автоматов). Проблему распознавания того, является ли пара автоматов эквивалентной, называется проблема Equality.

Замечание. Автоматы называются эквивалентными, если ...

Теорема 34. Проблема Equality для конечных автоматов является разрешимой.

Доказательство. TODO □

Раздел #19: Вопрос 20

19.1. k-регулярные подмножества $\mathbb{N}^n$

Замечание. Вообще говоря, автоматы принимают слова в произвольном алфавите. Однако множество слов всегда можно записать в какой-либо позиционной системе исчисления ($\mathbb{N}$). Каждое слово будет подаваться автомату по цифре в этой системе исчисления.

Определение 60 (Подмножество $\mathbb{N}$). Подмножество $\mathbb{N}$, принимаемое автоматом — это закодированный в какой-либо системе исчисления язык, алфавитом которого являются цифры (автомат принимает по одной цифре).

Замечание. Вообще говоря, особо не важно, принимать ли цифры слева направо, либо же справа налево (можно аккуратно доказать данный факт).

Замечание. Легко обобщить определение, которые мы дали выше на подмножество $\mathbb{N}^m$: просто будем считать, например, что кортеж подается последовательно. Однако стоит отметить, что если числа разного размера, то меньшие необходимо доопределять нулями слева.

Определение 61 (k-автоматные подмножества $\mathbb{N}$). Множество $K \subseteq \mathbb{N}$ называется k-автоматным (k-регулярным), если в алфавите $\{0, 1, ..., k-1\}$ множество k-ичных записей элементов K принимается некоторым автоматом.

Определение 62 (k-автоматные подмножества $\mathbb{N}^m$). Множество $K \subseteq \mathbb{N}^m (m > 1)$ называется k-автоматным (k-регулярным), если в алфавите $\{0, 1, ..., k-1\}^m$ множество k-ичных записей элементов K принимается некоторым автоматом.

19.2. Пример: 2-регулярность отношения делимости на 3

19.3. Формулировка теоремы Кобхема-Семёнова и её применение для доказательства невозможности построения конечного автомата

Определение 63 (Мультипликативная независимость). Пара положительных целых чисел a и b называются мультипликативно независимой, если для целых чисел n и m $a^n = b^m$ подразумевает $n = m = 0$.

Замечание. Рассмотрим следующую задачу: дано подмножество $\mathbb{N} = \{0; 1; 2; 3; ...\}$. Обозначим его E. Можем ли мы найти элементарный алгоритм, который принимает элементы

этого подмножества и отвергает те элементы, которые не принадлежат E (под «элементарным алгоритмом» мы подразумеваем конечный автомат)? Кобхем дал пару ответов на этот вопрос. В 1969 году он доказал, что существование такого алгоритма глубого зависит от системы исчисления (numeration base).

Теорема 35 (Первая теорема Кобхема). Пусть p и q являются двумя мультипликативно независимыми целыми числами, которые больше или равны двум. Тогда подмножество $E \subset \mathbb{N}$ является как p-recognizable (p-распознаваемым), так и q-recognizable, тогда, и только тогда, когда E — это конечное объединение арифметических прогрессий.

Замечание. Здесь "p-recognizable" означает что существует автомат, который принимает язык, который в точности состоит из разложения в системе исчисления p элементов E.

Пример. Например, множество $\{2^n; n \in \mathbb{N}\}$ является 2-recognizable. Так как не существует конечного объединения арифметических прогрессий, он не может быть 3-recognizable. Первая теорема Кобхема подразумевает что множество $\{2n; n \in \mathbb{N}\}$ является p-recognizable для любого $p \in \mathbb{N}$. Но это не говорит нам ничего о структуре recognizable множеств целых чисел. Рассмотрим следующую теорему, которая дает полное описание их структуры.

Теорема 36 (Вторая теорема Кобхема). A set $E \subset \mathbb{N}$ is p-recognizable if and only if its characteristic sequence $x \in \{0,1\}^{\mathbb{N}}$ ($x_i = 1$ if and only if $i \in E$) is the image by a letter to letter morphism of a fixed point of a substitution of constant length p.

Замечание. Пусть $d \geq 1$. Будем обозначать $(x_1, ..., x_d) \in \mathbb{N}^d$ в системе исчисления p как кортеж слов над алфавитом $\{0, 1, ..., p-1\}^d$ (с добавлением нулей в начале, если это необходимо). Например, в системе исчисления с основанием два мы запишем пару $(1, 5)$ как $(0,1), (0,0), (1,1)$. Будем говорить, что $E \subset \mathbb{N}^d$ является p-recognizable если E, записанное в системе исчисления с основанием p распознается некоторым конечным автоматом.

Теорема 37 (Теорема Кобхема-Семенова). Пусть p и q — пара мультипликативно независимых целых чисел, которые больше или равны двум. Множество $E \in \mathbb{N}^d$ является одновременно p-recognizable и q-recognizable тогда, и только тогда, когда E является полулинейным множеством (semilinear).
В другой формулировке, можно считать, что теорема звучит следующим образом:
пусть множество $E \subseteq \mathbb{N}^m$ определимо по отдельности в $< \mathbb{N}; 0; 1; +; V_k; =>$ и $< \mathbb{N}; 0; 1; +; V_l; =>$ (смотрите об этом ниже), где k и l мультипликативно независимы. Тогда E определимо в $< \mathbb{N}; 0; 1; +; =>$ (в арифметике Пресбургера).

19.4. Доказательство того факта, что всякое отношение, выразимое в $< \mathbb{N}; 0, 1, +, V_k, =>$, является k-регулярным

Определение 64 (Функция V_k). Определим функцию V_k следующим образом:
$>$ $V_k(x) = t$, где t — максимальное значение числа k в какой-либо натуральной степени, на которое делится x, если $x > 0$;
$>$ $V_k(x) = 1$, если $x = 0$.

Пример. Рассмотрим V_2:

- $x = 0$: $V_2(0) = 1$;
- $x = 1$: $V_2(1) = 2^0$, поскольку максимальная степень 2, на которую делится 1 является нулем;
- $x = 2$: $V_2(2) = 2^1$;
- $x = 3$: $V_2(3) = 2^0$;
- $x = 4$: $V_2(4) = 2^2$;
- $x = 5$: $V_2(5) = 2^0$;
- $x = 6$: $V_2(6) = 2^1$, поскольку 6 делится на 2^1;
- $x = 7$: $V_2(7) = 2^0$;
- $x = 8$: $V_2(8) = 2^3$;
- и так далее...

Замечание. V_k можно трактовать как двухместный предикат, однако в виде функции это понимается проще.

Определение 65 (Арифметика Бюхи). $< \mathbb{N}; 0; 1; +; V_k; => $ — элементарная теория структуры, называемая арифметикой Бюхи.

Замечание. Для разных k получаются разные арифметики.

Теорема 38. Пусть $k \geq 2$ — натуральное число, $E \subseteq \mathbb{N}^m$ — подмножество кортежей натуральных чисел. Тогда следующие утверждения эквивалентны:

- E k-распознается некоторым конечным автоматом A (является k-регулярным подмножеством);
- E определимо в арифметике Бюхи с параметром k.

Доказательство. Доказательство, по сути, конструктивно: глядя на формулу, можно алгоритмически построить автомат, а глядя на автомат, можно написать формулу.

Доказательство слева направо оставляем в качестве упражнения читателям (оно не требуется в вопросах);
Доказательство справа налево: TODO https://www.youtube.com/watch?v=MCgSH4lkHTA&ab_channel=%D0%9B%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0%D0%B2%D0%9C%D0%BE%D1%81%D0%BA%D0%B2%D0%B5 □

Раздел #20: Вопрос 25

20.1. Автоматы Бюхи и ω-регулярные языки

Определение 66 (ω-язык). Бесконечное слово представляет собой последовательность бесконечной длины (в частности, последовательность ω-длины) символов, а ω-язык представляет собой набор бесконечных слов. Здесь ω относится к множеству натуральных чисел.
Формально:
Пусть Σ — набор символов (не обязательно конечный). Следуя стандартному определению теории формального языка , Σ^* — это множество всех конечных(?) слов над Σ. Каждое конечное слово имеет длину, которая является натуральным числом. Бесконечные слова, или ω-слова, можно рассматривать как функции $\mathbb{N} \to \Sigma$. Множество всех бесконечных слов над Σ обозначается Σ^ω. Множество всех конечных и бесконечных слов над Σ иногда записывается как Σ^∞.

Определение 67 (ω-регулярный язык).

Замечание. ω-язык распознается автоматом Бюхи тогда и только тогда, когда он является ω-регулярным языком (можно считать эквивалентным определением).

20.2. Автомат Бюхи для сложения двух вещественных чисел

Пример. Конечный автомат для сложения натуральных чисел: https://studylib.ru/doc/207373/konechnye-avtomaty.