Лекции по математической логике и теории алгоритмов
А. Л. Семенов, 2023
Лекция 12

1 Исчисления и алгоритмы

Определение 1. *Двоичная цепочка* — цепочка в двоичном алфавите 0, 1.

При работе с исчислениями и алгоритмами объекты рассмотрения «конструктивны», к такому виду объектов относятся, например, конечные графы. Мы не будем пытаться дать общее определение конструктивного объекта и даже рассматривать что-то более общее, чем цепочки в конечном алфавите. Более того, и цепочки мы будем обычно рассматривать двоичные. Если нам понадобятся дополнительные символы, мы будем явно это указывать.

1.1 Исчисления и породимые множества

В математике, математической логике и в нашем курсе важную роль играют индуктивные определения, мы с ними уже встречались. Индуктивные определения могут использоваться при работе с различными классами объектов, в том числе и с произвольными множествами. Для нас, однако, важен случай, когда эти объекты «конструктивны», о чем мы уже говорили выше.

Определение 2. *Исчисление* — это однозначно понятный человеку текст, включающий:

- *алфавит исчисления*, включающий исходный алфавит $\{0, 1\}$;
- конечное множество *имен классов*; значением имени класса является класс объектов *относящихся* к этому *классу*; классы могут пересекаться; по объекту и имени класса можно за конечное время выяснить, относится ли объект к этому классу;
- *правило порождения* — отношение G между конечными множествами объектов, с указанием их классов, и объектами, с указанием класса объекта: если выполнено $G(A, b)$, то мы говорим, что объект b *порождается* множеством A (является *порождением*) множества A по правилу G.

  Выяснение того, что данное правило порождения выполнено, занимает обязательно конечное время; можно эту реализацию поручить машине.

  Заметим, что одно конечное множество G может порождать бесконечно много объектов b.

Множество, *порождаемое данным правилом* — это наименьшее множество S со следующим свойством: Для всякого конечного подмножества в S любое его порождение правилом (их может не быть) лежит в S (замкнутость).

Определение 3. Множество объектов данного класса,*порождаемое* данным *исчислением* — множество объектов этого класса, порождаемых правилом исчисления.

Определение 4. *Породимое* множество – множество объектов какого-то одного класса, порождаемое каким-то исчислением.

Нетрудно сообразить, что разные данные ранее в курсе индуктивные определения описывают конкретные исчисления.

Задача 1. Зафиксируем какую-то конечную сигнатуру, например, состоящую из одного имени объекта a и одного имени двухместного отношения B Опишите исчисление, порождающее все формулы логики отношений для этой сигнатуры.

Подсказка. Обратитесь к определению формулы из Лекции 3 Логические языки.

Решение. Для порождения всех формул мы использовали классы:

- имена объектов
- переменные
- имена n-местных отношений
- формулы

и правило порождения

- Все имена объектов и отношений, переменные порождаются пустым множеством.
- Если каждое из $t_1, t_2, \ldots, t_n$ — это имя объекта или переменная, A — имя n-местного отношения, то цепочка
$$A(t_1, t_2, \ldots, t_n),$$
порождается множеством $t_1, t_2, \ldots, t_n, A$ и является формулой.
- Если A — имя 0-местного отношения, то A является формулой и порождается пустым множеством.
- Если A — формула, x — переменная, то:
$\neg(A)$ — формула
$\exists x(A)$ — формула
$\forall x(A)$ — формула

- Если A и B формулы, то:
  $(A) \wedge (B)$,
  $(A) \vee (B)$,
  $(A) \to (B)$,
  $(A) \equiv (B)$
  — формулы.

В нашем случае нужно конкретизировать классы имен объектов и имен отношений. В частности, мы получим, что цепочки $B(a, x_2)$ $\exists x, (B(x_1, x_1)$относятся к классу формул и т.д.

Определение 5. Порождения пустого множества (всегда ли они есть?) можно называть *аксиомами*, другие элементы правила порождения — *правилами вывода*

Введя новый класс объектов: множеств, функций и т.д., математики часто задаются вопросом о замкнутости этого класса относительно естественных операций. Вот пример такого вопроса:

Задача 2. Будут ли объединение, пересечение и дополнение породимых множеств — породимы?

Подсказка. Пытаясь построить исчисление для объединения и пересечения можно пытаться просто «смешать» два исчисления для двух породимых множеств и «потом разбираться». Однако при таком подходе возникает трудность: порождая новые объекты, мы можем иногда использовать правило первого исчисления, иногда правило второго. Нужно как-то не смешивать, а «разделить» порождение в двух исчислениях.
В случае дополнения ситуация, кажется, еще хуже. Как можно породить то, что породить нельзя?

Решение. Естественно воспользоваться тем, что у нас объекты могут быть отнесены к тому или иному классу.
Итак, пусть у нас есть два исчисления со своими алфавитами. Мы хотим построить исчисления: для объединения порождаемых множеств и для пересечения этих множеств. Первые шаги в этих двух случаях будут одинаковыми.

Алфавиты двух данных исчислений просто объединим. Классы, напротив, разделим, будем считать имена классов непересекающимися. Этого можно достичь, например, к именам классов первого исчисления добавив индекс 1, к именам классов второго исчисления добавив индекс 2. Заведем еще один — *итоговый* класс, который нам понадобится в конце построения.

Также поступим и с двумя правилами, в каждом элементе правил двух исчислений заменим имена классов на имена с индексами.
Отнесем к правилу строящегося исчисления все полученные таким образом элементы.
Нетрудно доказать, что полученное объединенное правило будет порождать в точности те объекты, которые порождаются исходными правилами, у классов этих объектов будут проставлены индексы.

Пусть первое исчисление порождало объекты некоторого класса, скажем, первого, второе исчисление – второго. Добавим в правило для объединения пары, позволяющие у элемента каждого одноэлементного множества заменять первый и второй классы на итоговый. В правило для пересечения добавим пары, где первый элемент состоит из двух одинаковых объектов первого и второго класса, а второй элемент — это тот же объект, но уже итогового класса.
Очевидно, мы построили исчисления для объединения и пересечения породимых множеств.

■

Пусть фиксировано некоторое исчисление.

Определение 6. *Вывод* (в данном исчислении) — цепочка объектов, такая, что для каждого объекта x в ней существует (конечное) множество объектов, предшествующих x, такое, что x получается из него по правилу порождения. Последний элемент цепочки называется *результатом вывода.*

Задача 3. Доказать, что объект порождаем исчислением $\Leftrightarrow$ у него есть вывод.

Подсказка. Воспользуйтесь определением порождаемости, где есть требование замкнутости и минимальности. Попробуйте также использовать большую идею объединения возрастающей цепочки множеств.

Решение. Определим бесконечную возрастающую последовательность множеств. Каждый ее член получается добавлением к предыдущему всех порождений его конечных подмножеств.
Обозначим через U объединение этой цепочки.
Всякий элемент U появился в нем на каком-то конечном шаге.
Индукцией по номеру этого шага получаем, что все элементы U имеют вывод. Действительно, к выводу какого-то объекта надо отнести, например выписав их последовательно в любом порядке, выводы тех объектов, из конечного множества которых этот объект порожден. С другой стороны, индукцией по длине вывода можно доказать, что если элемент имеет вывод, то он лежит в U.

В то же время U замкнуто, действительно, если в нем есть конечное подмножество, то оно содержится уже в каком-то из членов последовательности, значит следующий член содержит порожденный этим конечным подмножеством элемент. Наконец, U минимально среди замкнутых. Именно, индукцией по номеру множества получаем, что всякое множество из последовательности лежит в любом замкнутом.

Таким образом, построенное объединение, с одной стороны, совпадает с множеством объектов, у которого есть вывод, с другой — с множеством порожденных объектов. Ясно, что это верно для каждого из классов.

1.2 Алгоритмы и вычислимые функции

Определение 7. *Алгоритм* — это однозначно понятный человеку текст. Этот текст:

- должен включать *алфавит* алгоритма, содержащий двоичный; *объектами алгоритма* будут цепочки в этом алфавите
- должен задавать (описывать) понятные человеку, обязательно завершающиеся:
  - *Проверку остановки*, дающую в применение к объекту И или Л;
  - *Действие переработки*, дающее в применении к объекту другой объект.

Вычисление алгоритма — последовательность объектов алгоритма:

- первый объект последовательности — произвольная двоичная цепочка — *исходное данное*,
- для всякого элемента x последовательности, кроме последнего (если последовательность конечна):
  - если Проверка остановки для x дает значение И, то следующий элемент последовательности является последним в последовательности и называется *результатом вычисления*;
  - следующий элемент последовательности существует и получается из x Действием переработки.

Переход от объекта последовательности к следующему мы называем *шагом (работы) алгоритма.*

Таким образом, прежде чем действовать, мы выясняем, не нужно ли будет, сделав очередной шаг, завершить деятельность. На самом первом шаге, и на последующих, мы можем добавить к аргументу дополнительные символы, заготовить несколько его

копий, как частей объекта, запустить какие-то вспомогательные алгоритмы, тоже описанные в Действии переработки и т.д. Зная, что очередной шаг будет последним, мы убираем все, что не войдет в результат.

Определение 8. *Функция, вычисляемая алгоритмом* — это отображение, ставящее в соответствие исходному данному результат вычисления алгоритма, начинающегося с этого исходного данного.
Если последовательность для какого-то исходного данного бесконечна, то результата нет и значение функции на таком аргументе не определено.

Определение 9. *Вычислимая функция* — функция, вычисляемая каким-либо алгоритмом.

Задача 4. Существует алгоритм со следующим свойством:
В последовательности, начинающейся с пустой цепочки, где каждый следующий элемент получается применением алгоритма к предыдущему, встречаются все двоичные цепочки.

Подсказка. В случае натуральных чисел алгоритм мог бы просто добавлять единицу.

Решение. Результатом для пустой цепочки является 0. Результатом для цепочкой длины n из одних единиц будет цепочка длины $n+1$ из одних нулей; результатом для цепочки не из единиц будет следующее число в двоичной системе той же длины (то есть, с нулями в начале).

Определение 10. Построенный в предыдущей задаче алгоритм назовем *счислительным алгоритмом.*

Задача 5. Постройте взаимнооднозначное соответствие между парами цепочек и цепочками. Цепочку, соответствующую паре будем называть *кодом пары.* Одновременно постройте два алгоритма: один (первый) дает из кода пары первый элемент пары, другой (второй) – второй элемент пары.

Определение 11. Зафиксировав какое-то соответствие из предыдущей задачи, код пары цепочек a, b будем обозначать $[a, b]$.

Подсказка. Можно обходить бесконечную двухмерную таблицу.

Решение. Действительно, воспользуемся бесконечной таблицей, где имена столбцов и имена строк – это все двоичные цепочки; на пересечении строки и столбца стоит пара соответствующих имен. Хорошо известно, как такую таблицу можно «обходить», то есть выстраивать все клетки в линейную последовательность. Используя при этом счислительный алгоритм, будем получать, доходя до пары, ее код и соответственно, из коды пары – два ее элемента.

В дальнейшем мы рассмотрим и другие алгоритмы кодировки, обладающие дополнительными свойствами.

Определение 12. *Перечислимое множество*: пустое, или являющееся множеством всех значений всюду определенной вычислимой функции.

Задача 6. Множество перечислимо $\Leftrightarrow$ оно породимо.

Подсказка. Попробуем установить соответствие между правилом порождения и действием переработки.

Решение. Пусть множество перечислимо. Построим соответствующее исчисление. В качестве алфавита исчисления возьмем алфавит алгоритма. Именами классов у нас будут «исходные данные», «промежуточные данные» и «результаты вычисления». Правило задает как аксиомы все двоичные цепочки, им присваивается класс исходных данных. Правила вывода $< \{a\}, b >$ соответствуют Действию переработки: a относится к классу исходных, или промежуточных данных, b получается из a за один шаг Действия переработки; b приобретает класс результата вычисления, если значения Проверки остановки на a — И. Если значение этой Проверки — Л, то b получает класс промежуточных данных.

Пусть множество породимо. Как и в других случаях, если оно пусто, то его перечислимость тривиальна. Если оно не пусто, будем пытаться каждую цепочку рассматривать как код вывода в исчислении, порождающем наше множество, пользуясь определением вывода и Правилом исчисления. Если она кодом вывода не является, будет выдавать в качестве результата всегда один и тот же фиксированный порождаемый объект. Если цепочка оказалась кодом вывода, то результат вывода считаем результатом работы нужного алгоритма.

Задача 7. Объединение и пересечение перечислимых множеств перечислимы.

Подсказка. Строящийся алгоритм, конечно, каким-то образом должен запускать алгоритмы для каждого из множеств.

Решение. Если одно из множеств пусто, то утверждение очевидно верно. Будем считать, что оба — не пусты, а для пересечения, что и пересечение непусто. Фиксируем какой-то элемент одного из множеств, если пересечение непусто, то лежащий и в другом тоже.
Будем строить нашу функцию для объединения или пересечения. Ее значением на пустой цепочке будем считать выделенный элемент.
Пусть исходное данное — это непустая двоичная цепочка x.
Если последний символ в x — 0, отбросим его и применим к остатку первый алгоритм, если последний символ в x — 1 отбросим его и применим к остатку второй алгоритм.

Если результатами считать все получаемые результаты, включая и результат для пустой цепочки, то получаем алгоритм для объединения.

Для пересечения можно поступить несколько хитрее.
Закодируем цепочками пары цепочек. Теперь, получив на вход цепочку, требуемый алгоритм выделит из нее две цепочки и даст, как исходные данные, двум алгоритмам, например, чередуя их шаги. Если два дадут один и тот же результат, то мы нашли элемент пересечения, он будет результатом построенной функции. Если же результаты различны, выдадим в качестве результата заранее заготовленный выделенный элемент.

Конечно, утверждение задачи вытекает из предшествующих результатов о породимых множества и из эквивалентности породимости и перечислимости.

Определение 13. Множество называется *разрешимым*, если его характеристическая функция вычислима.

Задача 8. Множество разрешимо $\Leftrightarrow$ оно и его дополнение перечислимы.

Подсказка. Имея алгоритм для «разрешения», то есть вычисляющий характеристическую функцию, надо построить два алгоритма для перечисления.
Имея два алгоритма для перечисления, надо построить алгоритм для характеристической функции.

Решение. Как обычно, если множество, или его дополнение — пусто, задача тривиальна.
Алгоритм для перечисления разрешимого множества, получив на вход объект, выясняет, лежит ли он в множестве, если лежит, то он же выдается как результат, если не лежит, то в качестве результата выдается один и тот же заранее подготовленный объект из множества. Перечисление дополнения — аналогично.

Пусть теперь есть алгоритм для перечисления множества и алгоритм для перечисления дополнения.
Требуемый алгоритм, получив исходное данное, запускает последовательный алгоритм, который мы построили выше. Для каждого результата последовательного алгоритма мы запускаем алгоритм для перечисления множества и алгоритм для перечисления дополнения. Если один из них дал в качестве результата наше исходное данное, то мы получили ответ. Если это не так, переходим к следующему результату последовательного алгоритма. ■

Задача 9. Функция вычислима $\Leftrightarrow$ ее график перечислим.
Область определения любой вычислимой функции перечислима.

Подсказка. Мы можем дождаться окончания любого процесса, лишь бы он закончился.

Решение. Если функция, скажем F, нигде не определена, то утверждение — верно. Если функция не пустая, зафиксируем некоторый ее элемент (точку на графике).

Исходя из алгоритма вычисления функции, построим алгоритм A перечисления ее графика. Этот алгоритм получает на вход цепочку. Если эта цепочка — код тройки $[[x, y], n]$ то строящийся алгоритм дает возможность алгоритму вычисления функции F сделать n шагов для исходного данного x, если за n шагов или раньше, вычисление закончилось, то результатом работы A будем считать $[x, y]$, если не закончилось, или не началось (когда на вход поступил не вход тройки) будем результатом считать зафиксированный заранее элемент. Если результатом работы взять не пару, а ее первый компонент, то описанный алгоритм будет перечислять область определения вычислимой функции.

Пусть график перечислим. Тогда, получив исходное данное, скажем x, мы должны запустить алгоритм перечисления графика и подождать, пока «приплывет» пара, где первым элементом является x. Если такой пары нет, то функция F не определена, а наш алгоритм не закончит работу.

■

Алгоритмы — это тексты в некотором алфавите. К кодированию текстов мы уже прибегали, обсуждая теорему Гёделя о неполноте. Фиксируем некоторый способ кодирования всех текстов в этом алфавите цепочками в двоичном алфавите.

Определение 14. Функция U называется *универсальной*, если она в применении к каждому коду $[p, x]$ пары $< p, x >$, где p — код алгоритма, а x — цепочка, находит результат применения алгоритма A с кодом p к исходному данному x.

$$U([p, x]) = A(x)$$

Задача 10. Универсальная функция вычислима.

Подсказка. Если вы поняли предшествующий текст, в частности, определение универсальной функции, то вы можете ее вычислять.

Решение. Действительно, если мы понимаем описание на нашем языке любого конкретного алгоритма, то мы понимаем приведенное выше описание алгоритма вычисления универсальной функции.

■

Определение 15. Определим “диагональную” функцию D равенством:

$$D(x) = U([x, x])0.$$

Тогда D — функция, вычисляемая некоторым алгоритмом. Обозначим код этого алгоритма (двоичную цепочку) через d.

Наша диагональная функция напоминает о Канторовой диагонали и Гёделевой диагонали. Дадим строкам и столбцам бесконечной таблицы имена — двоичные цепочки. В клетки поместим значение функции с кодом в строке на аргументе — имени столбца клетки. Тогда D получается «порчей» — дописыванием нуля к значениям, стоящим на диагонали таблицы.

Задача 11. Может ли функция D быть всюду определенной?

Подсказка. Чему равно $D(d)$?

Решение. Ясно, что $D(d)$ не определено (в этой клетки таблицы ничего не стоит).

Задача 12. Можно ли продолжить функцию D до всюду определенной?

Подсказка. А почему бы и нет?

Решение. Конечно можно, скажем, считать, что значение этой определенной функции равно 1 всюду, где D не определена.

Задача 13. Можно ли продолжить функцию D до вычислимой всюду определенной?

Подсказка. Решение для диагональной функции с пустой клеткой здесь не проходит.

Решение. Обозначим через D' какое-то продолжение функции D. Пусть D' — вычислима, d' — код вычисляющего ее алгоритма. Тогда

$$D(d') = U([d', d'])0 = D'(d')0.$$

Это равенство невозможно: если $D(d')$ определено, то оно должно быть равно $D'(d')$, если $D(d')$ не определено, то оно также не может быть равным всюду определенному выражению $D'(d')0$.

Задача 14. Может ли область определения функции D быть разрешимой?

Подсказка. Если бы область определения была разрешимой, можно было бы доопределить D до всюду определенной.

Решение. Если бы эта область определения была разрешимой, мы могли бы написать алгоритм, который сначала бы проверял, лежит ли цепочка в области определения, и если — нет, то выдавал 1, а если — да, то значение функции D.

Напомним, что область определения D перечислима, как у всякой вычислимой функции.

Таким образом мы построили не разрешимое перечислимое множество.

Способ построения такого множества напоминает способ доказательства несчетности множества последовательностей нулей и единиц (другими словами — характеристических функций подмножеств натурального ряда). Он также напоминает способ доказательства не определимости истины. Можно сказать, что это все — примеры большой идеи **Диагонали**.