

Билеты по теории автоматов

(на всякий случай, уточняем, что теория (а это первый и второй вопросы) сдается без подготовки: сел, вытянул вопрос, начал его отвечать. Да, это пиздец)

Первый вопрос

1. Машина Тьюринга. Проблема останова. Тезис Черча-Тьюринга.

Машина Тьюринга (МТ) – это абстрактная вычислительная машина, состоящая из управляющего устройства (УУ), бесконечной ленты, разбитой на сегменты, и головки считывания-записи (ГСЗ).

МТ в каждый момент времени находится в состоянии q и обозревает символ z . И в зависимости от этого МТ может изменить состояние, заменить символ на ленте или сместиться по ней вправо/влево на одну ячейку.

МТ задается пятеркой $MT = (Q, q_0, z, F, \delta)$, где:

- $Q = \{q_0, q_1, \dots, q_k\}$ – конечное множество состояний;
- q_0 – начальное состояние;
- F – множество финальных состояний (из него нет переходов, зачастую достаточно и одного);
- z – внешний алфавит;
- δ – функция перехода. Она имеет вид $\delta(q, z) = (q', z', d)$, где q и z – текущие состояние и символ на ленте, q' и z' – новые состояние и символ на ленте, d – направление сдвига (вправо/влево/на месте).

Тезис Черча-Тьюринга гласит, что если значение функции можно вычислить при помощи алгоритма, то для этого можно создать МТ.

Проблема останова – пример алгоритма нерешаемой задачи. Не существует такого алгоритма, что принимает на вход слово и другой алгоритм, который бы за конечное время определил бы, зациклится ли алгоритм или нет.

Доказательство от противного:

Если бы такой алгоритм существовал, можно было бы сконструировать хитрую "программу-парадокс". Эта программа-парадокс брала бы любую программу, скормливала ее (вместе с ее же собственным кодом в качестве входных данных) нашему алгоритму.

Если алгоритм говорит: "Эта программа остановится", то наша программа-парадокс специально уходит в бесконечный цикл.

Если алгоритм говорит: "Эта программа зациклится", то наша программа-парадокс немедленно останавливается.

Теперь спросим: что сделает наша программа-парадокс, если ей на вход дать ее собственный код? Если она остановится, то (по ее логике) она должна была зациклиться. Если она зациклится, то (по ее логике) она должна была

НАМ ПИЗДА, ПО РАЗМЕРАМ ПРЕВОСХОДЯЩАЯ САС =)

остановиться. Это противоречие! Значит, наше первоначальное предположение о существовании "супер-отладчика" было неверным.

2. Автомат-распознаватель. Лемма о разрастании.

Автомат-распознаватель – это самый простой автомат, который проверяет слово, подходит ли оно нам или нет.

Для работы ей нужны следующие детали (ну или умным языком: «Автомат-распознаватель задается пятеркой»):

- $A = \{a_0, a_1, \dots, a_k\}$ – конечное множество состояний;
- a_0 – начальное состояние;
- F – множество финальных состояний (из них есть переходы);
- Z – алфавит;
- δ – функция перехода. Она имеет вид $\delta(a, z) = (a', z', d)$, где a и z – текущие состояние и символ на ленте, a' и z' – новое состояние.

Лемма о разрастании – это правило проверки языка, что для его распознавания можно построить конечный автомат (КА). Она гласит:

Для любого языка (L), который может распознать КА, существует такое число n (состояний в КА), что если слово (w) принадлежит языку и его длина больше n , то такое слово можно разбить на три части: $w = xyz$, причем $0 \leq |y| \leq n$ и $xy^kz \in L$ для всех $k \geq 0$.

Если простыми словами:

Лемма о разрастании – это хитрое свойство, которое есть у всех "простых" языков (которые распознаются конечными автоматами, их называют регулярными).

Она говорит: если язык простой (регулярный), то любое достаточно длинное слово из этого языка можно "накачать" или "сдуть" в середине, и оно всё равно останется "правильным" (останется в этом языке).

3. Формальная грамматика. Типы формальных грамматик и языков.

Формальная грамматика (ага, просто грамматика) – это способ точного описания формального языка, то есть множества строк, построенных из некоторого алфавита. Грамматика задает правила, по которым могут быть порождены (сгенерированы) все строки, принадлежащие языку, и только они.

Она задается четверкой $G = (\Sigma, N, P, S)$, где:

- Σ – терминальный алфавит. Это все возможные выходные символы, их принято писать маленькими буквами;

- N – нетерминальный алфавит. По сути, это вспомогательные символы, которые не входят в выходной алфавит, они нужны чисто для построения слов. Принято обозначать их большими буквами;
- P – множество правил. Каждое правило выглядит как $\alpha \rightarrow \beta$ и означает, что цепочка α может быть заменена на цепочку β . В левой части, α , должен быть хотя бы один нетерминальный символ;
- S – стартовый символ. С него начинается порождение слов языка, и он обязан быть нетерминальным.

Есть несколько типов таких грамматик, поэтому чел с фамилией **Хомский** придумал для них иерархию. Но для начала вот вам несколько обозначений, чтоб правила читались понятнее:

- α, β, γ – строки из алфавита $N \cup \Sigma \cup \{\lambda\}$ (АБСОЛЮТНО любая строка);
- δ – строка из $N \cup \Sigma \cup \{\lambda\}$ (непустая строка из любого из алфавитов);
- μ, R – строки из N (из нетерминального алфавита);
- a – строка из Σ (из терминального алфавита).

А теперь к иерархии Хомского:

1. Неограниченная грамматика. В ней все правила имеют вид « $\alpha\mu\beta \rightarrow \gamma$ », т.е. слева должна быть строка с хотя бы 1 нетерминалом, а справа – из терминалов, нетерминалов или вообще пустая. Больше правил нет.

По сути, такая грамматика эквивалентна МТ.

2. Контекстно-зависимая. В ней все правила имеют вид « $\alpha\mu\beta \rightarrow \alpha\delta\beta$ », т.е. нетерминал слева может быть заменен на непустой символ алфавита ТОЛЬКО в окружении α и β (в контексте, так сказать).

Такая грамматика уже эквивалентна МТ, где длина ленты равна длине входного слова.

3. Контекстно-свободная. В ней все правила имеют вид « $\mu \rightarrow \gamma$ », т.е. нетерминал слева может быть заменен на что угодно, не зависимо от окружения (контекста).

Такая грамматика эквивалентна КА с магазинной памятью (со стеком, если так понятнее).

4. Регулярная грамматика. В ней уже есть подразделы на правую и левую:

- « $\mu \rightarrow Ra$ » - правило левой грамматики, т.е. новый нетерминал может появиться только слева от терминала;
- « $\mu \rightarrow aR$ » - правило правой грамматики, т.е. новый нетерминал может появиться только справа от терминала.

Такая грамматика эквивалентна конечным автоматам.

ПРИМЕЧАНИЕ: иерархия Хомского является вложенной, т.е. каждая следующая грамматика – это частный случай предыдущего.

4. Регулярная грамматика. Левые и правые регулярные грамматики.

Регулярная грамматика – это самая простая из формальных грамматик по иерархии Хомского. Она порождает **регулярные языки**, которые могут быть распознаны КА. У нее строгие ограничения, но от этого ее анализ и использование становятся солидно проще.

Регулярная грамматика подразделяется на два вида:

1. Левая грамматика. В ней все правила имеют вид « $\mu \rightarrow Ra$ », т.е. новый нетерминал может появиться только слева от терминала. Левую грамматику удобнее использовать, когда нас интересует проверить, например, конец слова;

2. Правая грамматика. В ней все правила имеют вид « $\mu \rightarrow aR$ », т.е. новый нетерминал может появиться только справа от терминала. Правую грамматику удобнее использовать, чем левую в большинстве случаев, т.к. слово разрастается вправо (имхо).

Левая и правая грамматики, по сути, эквивалентны, т.е. если нужно, то можно перевести грамматику из правой в левую, и наоборот. Но такой перевод осуществляется через построение автомата (практичнее и понятнее) или реверсирование правил языка.

ВАЖНО: регулярная грамматика не может быть одновременно и левой, и правой. Иначе грамматика перестанет быть регулярной.

Регулярная грамматика эквивалентна КА, и регулярным выражениям. Т.е. по регулярной грамматике можно создать КА или регулярное выражение, и наоборот.

5. Регулярное выражение. Теорема Клини. Лемма Ардена.

Регулярное выражение (РВ) – это последовательность символов, которая определяет шаблон слов в языке. Иными словами, РВ применяются для описания множества слов (чем и является язык) в компактной и удобной форме.

В регулярных выражениях можно выделить следующие случаи:

1. \emptyset – РВ не содержит ни одной строки и задает пустой язык $L=\{\emptyset\}$;
2. λ – РВ содержит одну пустую строку и задает язык $L=\{\lambda\}$;
3. a – РВ содержит одну строку, состоящую из одного символа, и задает язык $L=\{a\}$;
4. Если R и S – РВ для языков L_R и L_S соответственно:
 - 4.1. $R+S$ или $R|S$ – РВ для объединения языков L_R и L_S ;
 - 4.2. RS – РВ для конкатенации языков L_R и L_S ;

4.3. R^* – PB для языка L_R^* . Обозначает, что выражение R можно выкинуть или объединить с самим собой сколько угодно раз. Эта операция называется операцией Клини – в честь Стивена Клини, изобретателя PB.

Приоритет операций:

1. $*$ (операция Клини)
2. Конкатенация
3. Объединение

PB – это частный случай алгебры Клини. Поэтому у PB есть следующие аксиомы операций (R, S, T – это PB):

1. Ассоциативность объединения и конкатенации: $(R+S)+T=R+(S+T)$ и $(RS)T=R(ST)$;
2. Коммутативность: $R+S=S+R$, но $RS \neq SR$;
3. Дистрибутивность: $R(S+T)=RS+RT$ и $(S+T)R=SR+TR$
4. Идемпотентность (безразмерность): $R+R=R$ и $(R^*)^*=R^*$

Теорема Клини гласит, что множества регулярных и автоматных языков совпадают.

А это уже означает, что конечные автоматы, регулярные грамматики (которые также эквивалентны КА) и регулярные выражения обладают одинаковой выразительной мощностью – все они описывают один и тот же класс языков, а именно класс регулярных языков. Это позволяет переходить от одного представления к другому, выбирая наиболее удобное для конкретной задачи.

Лемма Ардена – это удобный инструмент, помогающий в решении системы линейных уравнений. Линейные уравнения встречаются в алгебре Клини, когда нужно перевести КА в PB и закрывают один важный недостаток алгебры Клини – отсутствие операции вычитания.

Лемма гласит:

« $x = A^*B$ – наименьшее решение для уравнения $x = Ax + B$, причем, если $\lambda \notin A$, то такое решение будет единственным. A и B – это известные PB.

Аналогично $x = BA^*$ – решение для уравнения $x = xA + B$ »

6. Автомат-преобразователь. Автоматы Мили и Мура.

Конечный автомат-преобразователь – автомат, который преобразовывает поданное на вход слово и возвращает на выходе результат обработки этого слова (выходное слово). Он задается шестеркой $S = (A, Z, W, \delta, \lambda, a_0)$, где:

- $A = \{a_0, a_1, \dots, a_m, \dots, a_m\}$ – конечное множество состояний (алфавит состояний);

- $Z = \{z_1, \dots, z_f, \dots, z_F\}$ – конечное множество входных символов (входной алфавит);
- $W = \{w_1, \dots, w_g, \dots, w_G\}$ – конечное множество выходных символов (выходной алфавит);
- $\delta: A \times Z \rightarrow A$ – функция переходов; каждой паре (a_m, z_f) она ставит в соответствие состояние перехода a_s , т. е. $a_s = \delta(a_m, z_f)$;
- $\lambda: A \times Z \rightarrow W$ – функция выходов; каждой паре (a_m, z_f) она ставит в соответствие выходной символ w_g , т. е. $w_g = \lambda(a_m, z_f)$;
- $a_0 \in A$ – начальное состояние автомата.

Наибольшее распространение получили конечные автоматы-преобразователи модели Мили и Мура. Они отличаются тем, что выходной символ в автомате Мили зависит как от исходного состояния, так и от входного символа в рассматриваемый момент времени, а в автомате Мура выходной символ зависит только от состояния перехода.

Из-за этого различия способы их определения так же отличаются:

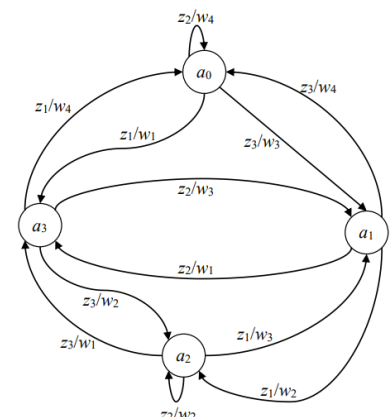
1. Автомат Мили задается в виде таблицы как совмещенная таблица переходов-выходов (СТПВ). В ячейку таблицы переходов, помимо состояния (в которое осуществляется переход) заносится еще и символ выходного алфавита:

	a_0	a_1	a_2	a_3
z_1	a_3/w_1	a_2/w_2	a_1/w_3	a_0/w_4
z_2	a_0/w_4	a_3/w_1	a_2/w_2	a_1/w_3
z_3	a_1/w_3	a_0/w_4	a_3/w_1	a_2/w_2

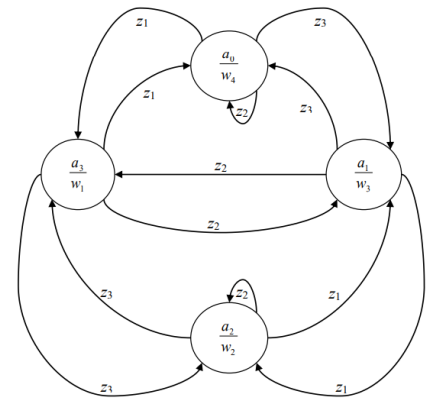
У автомата Мура таблица задается как отмеченная таблица переходов (ОТП), в которой символ выходного алфавита указывается над состоянием, к которому он «привязан»:

	w_4	w_3	w_2	w_1
	a_0	a_1	a_2	a_3
z_1	a_3	a_2	a_1	a_0
z_2	a_0	a_3	a_2	a_1
z_3	a_1	a_0	a_3	a_2

2. В графическом представлении автомат Мили задается в виде упорядоченного графа, где вершины графа – состояния, а ребра – переходы, причем на переходах указывается как символ входного алфавита (по которому осуществляется переход), так и символ выходного.



Автомат Мура так же задается в виде упорядоченного графа, только в вершинах графа, помимо состояний, указываются и привязанные к ним символы выходного алфавита, а на ребрах – только символы входного алфавита.



7. Элементарный автомат. Триггеры D, T, RS и JK. Теорема Глушкова о структурной полноте. Обобщенная структурная схема автомата.

Теорема Глушкова о структурной полноте гласит, что ЛЮБОЙ конечный автомат можно реализовать, используя набор элементарных автоматов и функционально полную систему логических элементов.

Система логических элементов считается функционально полной, если на ней можно реализовать любую логическую функцию:

- ❖ «И», «ИЛИ», «НЕ»;
- ❖ «И-НЕ» (стрелка Шеффера);
- ❖ «ИЛИ-НЕ» (стрелка Пирса).

Каждый пункт – это отдельная, функционально полная система лог. элементов

Автомат является элементарным, если:

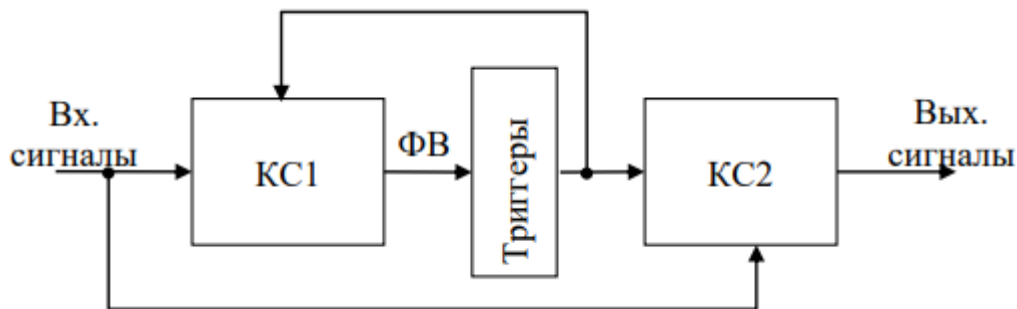
1. Из каждого состояния есть переход в каждое состояние (т.е. он обладает полнотой переходов);
2. В каждом состоянии автомат выдает уникальный выходной символ (он обладает полнотой выходов).

Примерами элементарных автоматов являются триггеры:

1. D-триггер. Является элементом задержки, и в то же время элементом памяти, т.к. сохраняет то логическое значение, которое получил на входе. У триггера только один вход.
2. T-триггер. Триггер переключения: при подаче на него сигнала «1» переключается из одного состояния в другое. Если подавать на него 0, то он останется в том же состоянии. У триггера только один вход.
3. RS-триггер. Так же является триггером переключения, но уже с двумя входами: S (Set) и R (Reset). Режим работы у него следующий: При подаче 0 на оба входа он сохранит свое состояние. При подаче 0 на R и 1 на S триггер переходит в состояние 1. При подаче 1 на R и 0 на S триггер сбрасывается в состояние ноль. Подать 1 на оба входа нельзя, иначе он уйдет в неопределенное состояние.

3. JK-триггер. Усовершенствованная версия RS-триггера. Для простоты запоминания можно запомнить входы как J=Jump и K=Kill. При подаче 0 на оба входа триггер сохраняет свое состояние. При подаче 1 на J и 0 К триггер переходит в состояние 1. При подаче 0 на J и 1 на К триггер переходит в состояние 0. При подаче 1 на оба входа триггер меняет свое состояние на противоположное.

Обобщенная схема структурного автомата для автомата Мили:



В случае с автоматом Мура из схемы уйдет стрелка от входных сигналов к КС2. Это обусловлено их разницей ([см. предыдущий вопрос](#)).

КС – комбинационные схемы, состоящие из логических элементов. КС1 нужна для формирования сигналов функции возбуждения триггеров, а КС2 – для формирования выходных сигналов.

8. Понятия детерминированного и недетерминированного конечного автомата, полностью и частично-определенных автоматов, эквивалентных и минимальных автоматов, эквивалентного продолжения.

Детерминированный конечный автомат можно представить как некоторое устройство, имеющее один вход и один выход и в каждый момент времени находящееся в одном состоянии из конечного множества состояний.

Недетерминированный автомат может находиться в множестве состояний одновременно. Для одного состояния и входного символа может быть ноль, один или несколько возможных переходов. Переходы позволяют менять состояние без чтения входного символа.

Частично-определенный автомат – это частный случай детерминированного автомата, где могут отсутствовать некоторые переходы. Для некоторых пар (состояние, символ) перехода нет. Такой автомат может "зависнуть".

Полностью определенный автомат – это частный случай детерминированного автомата, где нет пропущенных переходов. Для любой пары (состояние, символ) есть точно один переход. Такой автомат никогда не "зависает". Преобразование частично-определенного автомата в эквивалентный ему полностью определённый производится добавлением

тупикового состояния и заменой прочерков в таблице переходов на переход в это состояние.

Эквивалентные автоматы – это автоматы, которые распознают абсолютно один и тот же язык (одинаковые слова). Могут быть разного типа (детерминированный/недетерминированный) и иметь разное число состояний, но их выходное поведение идентично.

Минимальный автомат – это автомат, который имеет наименьшее возможное количество состояний среди всех эквивалентных ему детерминированных автоматов.

Эквивалентное продолжение – это превращение частичного детерминированного автомата в полный детерминированный автомат, не меняя его язык. Автомат B , выполняющий то же преобразование, что и автомат A , называется эквивалентным продолжением автомата A . Обратим внимание, что при этом автомат B может выполнять преобразования слов, не допустимых автоматом A . Добавляется состояние (не конечное), куда направляются все отсутствовавшие переходы. Язык автомата остается прежним, но он больше не "зависает".

Второй вопрос

1. Синтез конечного автомата по регулярной грамматике (правой и левой).

Регулярная грамматика – это самый простой подвид формальной грамматики, который является эквивалентным конечным автоматом. Регулярная грамматика подразделяется на правую и левую. Отличаются они только подстановкой символа нетерминального алфавита (в левой он подставляется слева, в правой – очевидно, справа). Более подробно про это расписано [в первом блоке вопросов](#).

Алгоритм построения автомата, несмотря на «сторону» грамматики, один и тот же, и осуществляется с помощью установления соответствия между грамматикой и КА:

Грамматика	Конечный автомат
Σ (алфавит языка)	z (алфавит автомата)
N (нетерминальный алфавит) $\cup \{a_z\}$ (множество финальных состояний)	A (множество состояний)
P (множество правил)	δ (функция перехода)
S (стартовый нетерминальный символ)	a_0 (начальное состояние)
$\{a_z\}$ (множество финальных состояний) [см. последний абзац]	F (множество финальных состояний)

Дальше по такой подстановке все становится проще. Состояния называем как символы нетерминального алфавита, а переходы берем из множества правил: слева стоит состояние, откуда идет переход, а справа – символ, по которому мы делаем переход, и состояние, куда ведет переход.

Так как в грамматике нет отдельного символа для финального состояния, его мы будем называть «F». Переход будет вести в состояние F, если правило грамматики, по которому этот переход создается, не порождает новые НЕТЕРМИНАЛЬНЫЕ символы.

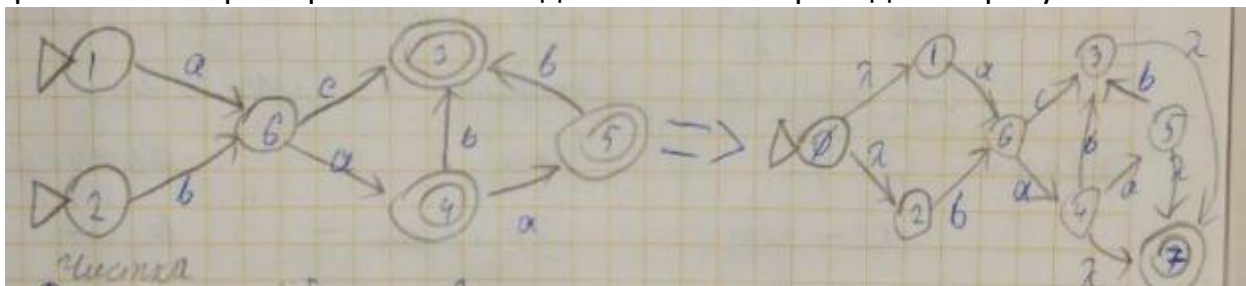
2. Синтез регулярного выражения по конечному автомату. Метод Бжозовского и Маккласки. Метод линейных уравнений.

Из КА-распознавателя можно получить регулярное выражение двумя способами.

Первый способ – метод Бжозовского и Маккласки. Он состоит из двух этапов:

1. Преобразование КА в КА с одним начальным и финальным состояниями. Данный этап выполняется один раз, причем для этого

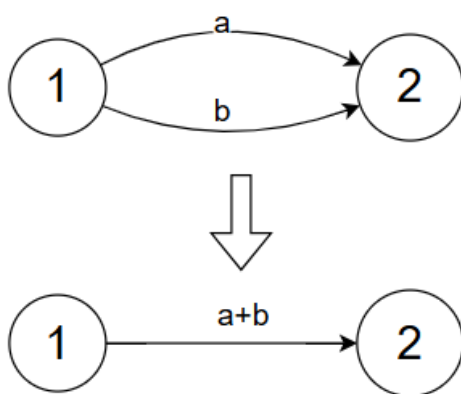
используются пустые переходы (λ -переходы) из нового начального и в новое финальное. Пример выполнения данного этапа приведен на рисунке ниже:



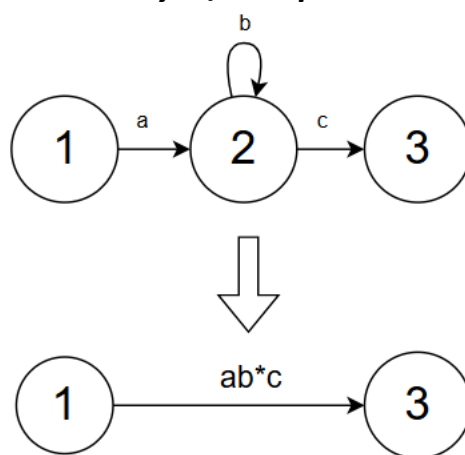
Как видно, мы создали два состояния (новые начальное и финальное). От нового начального строим пустые переходы в старые начальные, а в новое финальное – из старых финальных.

2. Редукция (чистка) ребер и вершин. Данный этап уже выполняется рекурсивно, пока у нас не останется только начальное и финальное состояния. Заключается данный этап в том, что мы меняем соответствующий элемент на эквивалентное РВ:

Редукция ребер



Редукция вершин



Второй способ – метод линейных уравнений. Он так же состоит из двух этапов:

1. Для каждого состояния пишем регулярное выражение так, чтобы оно соответствовало языку, распознающемуся КА при условии, что это состояние – начальное. РВ в данном случае и будет нашим неизвестным, которое обозначается как R_i , i – номер состояния.

Стоит уточнить, что РВ для финального состояния будет всегда равно λ (пустому символу)

2. Решаем систему. Наша цель – найти R_0 .

Проблема в том, что в алгебре Клини (которой подчиняются все операции с РВ) нет операций с минусом, поэтому используется лемма Ардена:

« $x = A^*B$ – наименьшее решение для уравнения $x = Ax + B$, причем, если $\lambda \notin A$, то такое решение будет единственным. A и B – это известные РВ.

Аналогично $x = BA^*$ – решение для уравнения $x = xA + B$ »

Пример решения (для понимания) приведен на рисунке ниже:

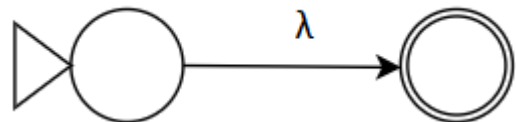
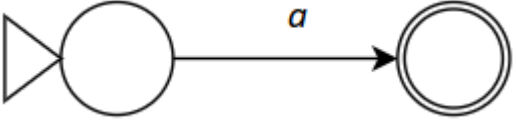
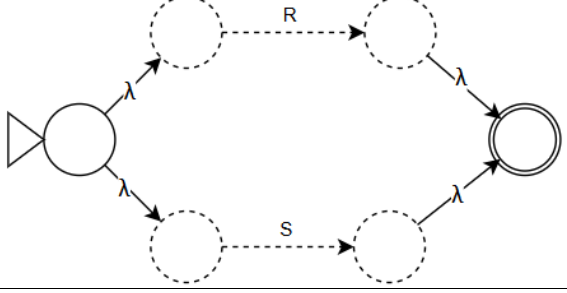

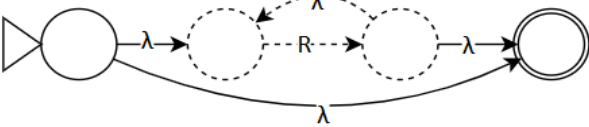
$$\begin{cases} R_0 = aR_1 + cR_3 \\ R_1 = (b+c)R_2 \\ R_2 = aR_1 + cR_3 \\ R_3 = \varepsilon \end{cases} ; \begin{cases} R_0 = aR_1 + c \\ R_1 = (b+c)R_2 \\ R_2 = aR_1 + c \end{cases} ; \begin{cases} R_0 = aR_1 + c \\ R_1 = (b+c)(aR_1 + c) = \underbrace{(b+c)aR_1}_{A} + \underbrace{(b+c)c}_{B} \end{cases}$$

$$\begin{cases} R_0 = a((b+c)a)^*(b+c)c = [(a(b+c))^*a(b+c) + \varepsilon]c = [(a(b+c))^*c] \\ R_1 = (b+c)a^*(b+c)c \end{cases}$$

3. Синтез конечного автомата по регулярному выражению. Алгоритм Глушкова. Алгоритм Мак-Нотона-Ямады-Томпсона.

Из регулярного выражения можно получить КА-распознаватель двумя способами.

Первый способ – алгоритм Мак-Нотона-Ямады-Томпсона. Для простоты его можно называть просто алгоритмом Томпсона. Суть алгоритма заключается в простой замене части РВ на соответствующий граф:

Конструкция РВ	Граф
λ (пустая строка)	
a (любой один символ)	
$R+S$	
RS	
R^*	

КА, созданный по алгоритму Томпсона, обладает следующими свойствами:

1. Всегда одно начальное и финальное состояния;
2. Нет переходов в начальное и из финального состояния;
3. Из каждого состояния есть 1 переход по символу или до 2 переходов по λ ;
4. Число состояний в КА считается по формуле $2S - C$, где S – длина строки РВ без скобок, C – число конкатенаций (соединений строк).

Второй способ – алгоритм Глушкова. Он состоит из следующих этапов:

1. Линеаризация РВ. Каждый символ должен встречаться в РВ не более одного раза, поэтому выполняется их глобальная индексация. В результате получим PV_1 .

2. Вычисление следующих множеств, допустимых PV_1 :

S – множество первых символов (отражает кол-во переходов из начального состояния)

F – множество последних символов (и, по совместительству, финальных состояний)

P – множество всех пар символов (оно же – множество всех переходов, исключая начальное состояние)

Λ (заглавная лямбда) = $\{\lambda\}$, если РВ (не PV_1) допускает λ , иначе $\Lambda = \emptyset$ (если $\Lambda = \{\lambda\}$, то начальное состояние является финальным)

3. Построение КА, распознающего PV_1 . Тут есть важное правило: все дуги, отмеченные одинаковым символом (по PV_1) ведут в одну вершину (для удобства, ее подписывают так же, как и символ). В результате получится локальный КА.

4. Удаление линеаризации. На этом этапе мы удаляем все индексы у символов и приводим название состояний в привычный вид.

В результате получится НКА со следующими свойствами:

1. Имеет только одно начальное состояние, но может иметь несколько финальных с переходами из них.

2. Количество переходов из каждого состояния неограниченно, нет переходов по λ

3. Число состояний всегда больше числа символов в РВ без учета знаков на 1.

4. Построение детерминированного конечного автомата по недетерминированному. Удаление пустых переходов. Концепция множеств.

В результате работы по синтезу КА, он может получиться недетерминированным, что плохо ([вопрос 8 из блока «Первый вопрос»](#)). Поэтому его нужно детерминировать.

Хоть и λ -переходы допустимы для ДКА, на практике они, скорее, избыточны, поэтому от них можно избавиться.

Если есть λ -переход из состояния p в состояние q , то нужно продублировать все переходы из q в p , а λ -переход удалить.

ВАЖНО: если q является финальным, то и p должно стать финальным.

Для построения ДКА из НКА можно прибегнуть к концепции множеств. Будут рассматриваться все возможные множества состояний исходного НКА (2^n множеств для n состояний НКА). Введем следующие правила:

1. Каждое множество – это состояние в искомом ДКА.
2. Начальное состояние ДКА – это множество, состоящее из всех начальных состояний НКА.
3. Финальные состояния ДКА – все множества, в которых есть хотя бы одно финальное состояние НКА.

Детерминизация будет происходить следующим образом:

1. Сначала будет сформировано начальное множество, после определим переходы из него. Состояния, в которые осуществляется переход по одному символу, будут формировать множество.
2. Если подобное множество не фигурировало ранее, то оно образует новое состояние ДКА. После этого будут рассматриваться переходы из этого множества.
3. Это будет повторяться до тех пор, пока не перестанут появляться новые множества.

5. Синтез автоматов Мили и Мура по оператору соответствия.

Оператор соответствия (ОС) – это один из способов задания абстрактного автомата-преобразователя, представляющий собой набор входных и соответствующих им выходных слов данного автомата. ОС оформляется в виде таблицы, где левая часть содержит перечень входных слов, а правая – соответствующие им выходные слова, которые автомат формирует при их обработке.

Для корректной работы автомата, построенного с помощью ОС, должны соблюдаться два условия:

1. Начальное состояние: Перед поступлением на вход первого символа любого входного слова, автомат должен находиться в своем начальном состоянии a_0 .
2. Возврат в начальное состояние: После поступления на вход последнего символа любого входного слова, автомат должен вернуться в начальное состояние a_0 .

Для построения автомата по ОС второй должен быть приведен к **автоматному виду**. Для этого ОС должен соответствовать двум условиям:

1. Однозначность отображения: Если два входных слова имеют совпадающие начальные отрезки длиной n символов, то их соответствующие выходные слова также должны иметь совпадающие начальные отрезки длиной n символов.

2. Равенство длин: Длины входного слова и соответствующего ему выходного слова должны быть равны.

Если ОС не соответствует автоматному виду, то его нужно преобразовать путем добавления "пустых" символов α и β : α приписывается справа к входному слову, а β приписывается слева к выходному слову.

А теперь к самому синтезу автоматов. Он состоит из трех этапов:

1. Приведение ОС к автоматному виду.

2. Построение графов автоматов Мили и Мура на основе преобразованного ОС.

3. Составление соответствующих таблиц переходов (СТПВ для Мили, ОТП для Мура) по графам.

Если первый этап у них одинаковый, то остальные, хоть и немного, но различаются. Примеры и процессы построения изложены в [вопросе 3 из блока задач](#).

Синтез автомата Мили.

В автомате Мили выходной символ зависит как от текущего (исходного) состояния, так и от входного символа в рассматриваемый момент времени.

Построение графа начинается с начального состояния a_0 . Из a_0 выходят дуги для каждого возможного первого входного символа. Последующие состояния и переходы выводятся непосредственно из ОС. Каждая дуга перехода маркируется «входной_символ/выходной_символ».

Таблица переходов строится на основе графа. В случае с автоматом Мили она называется Совмещенная Таблица Переходов-Выходов (СТПВ). Её ячейки на пересечении строки (входной символ) и столбца (текущее состояние) содержат следующее состояние и выходной символ, в формате «следующее_состояние/выходной_символ».

Синтез автомата Мура.

В автомате Мура выходной символ зависит только от состояния, в которое автомат переходит (следующего состояния).

Принцип построения графа схож с автоматом Мили, но с двумя ключевыми отличиями:

1. Выходные символы записываются внутри узлов состояний в графе, вместе с именем состояния.

2. В отличие от автомата Мили, который всегда имеет одно начальное состояние, автомат Мура может иметь несколько начальных состояний. Число начальных состояний соответствует числу различных последних выходных символов в выходных словах оператора соответствия.

Таблица переходов так же строится на основе графа. Только теперь она называется Отмеченной Таблицей Переходов (ОТП). Первые две строки в таблице – это символ внутри узла и имя состояния(узла). Ниже под каждой такой парой указывается при каком символе в какое состояние переходит автомат.

6. Минимизация автомата. Понятие эквивалентных состояний. Метод расщепления классов эквивалентных состояний. Понятие обратного автомата. Алгоритм Бжозовского.

Минимизация конечного автомата — это процесс построения эквивалентного автомата, который имеет минимально возможное число состояний.

Процесс минимизации включает в себя три основных действия:

1. **Удаление недостижимых состояний.** Недостижимым состоянием называется такое состояние, в которое автомат не может перейти из начального состояния ни при какой последовательности входных символов.

Алгоритм поиска недостижимых состояний включает шаги:

Шаг 0: создать множество R, содержащее начальное состояние автомата.

Шаг 1: добавить в R состояния, в которые переходит автомат из состояний, уже находящихся в R.

Шаг 2: если на шаге 1 множество изменилось, то перейти к шагу 1; иначе R – множество достижимых состояний; остальные состояния недостижимы.

2. **Удаление тупиковых состояний.** Тупиковым состоянием называется такое состояние, из которого автомат не может перейти в финальное состояние ни при какой последовательности входных символов.

Алгоритм поиска тупиковых состояний включает шаги (с конца в начало):

Шаг 0: создать множество N, содержащее все финальные состояния автомата.

Шаг 1: добавить в N состояния, из которых переходит автомат в состояния, уже находящихся в N.

Шаг 2: если на шаге 1 множество изменилось, то перейти к шагу 1; иначе N – множество нетупиковых состояний; остальные состояния являются тупиковыми.

3. Объединение неразличимых (эквивалентных) состояний. Это ключевой шаг в минимизации. Эквивалентные состояния — это состояния, которые ведут себя одинаково при любых входных воздействиях.

Формально, состояние a_i автомата A_1 эквивалентно состоянию a_j автомата A_2 , если автомат A_1 в состоянии a_i и автомат A_2 в состоянии a_j под воздействием любой последовательности входных символов вырабатывают одинаковые выходные реакции. При минимизации автомата группа эквивалентных состояний, принадлежащих одному автомату, заменяется **одним состоянием**.

ВАЖНО: для поиска эквивалентных состояний наш ДКА должен быть полностью определен.

Для определения эквивалентных состояний есть два способа:

1. Метод расщепления классов эквивалентных состояний.
2. Метод, основанный на использовании треугольной таблицы.

Более детально второй способ описан в [вопросе 6 из блока задач](#).

Рассмотрим первый способ:

Два состояния автомата называются k -эквивалентными, если не существует слова длиной k или менее, на которое автомат в этих состояниях реагировал бы по-разному.

В частности, два состояния автомата-распознавателя называются нуль-эквивалентными (0-эквивалентными, то есть E_0 из примера), если они оба являются финальными или оба не являются финальными.

Шаг 0: Установить $k = 0$. Выполнить разбиение множества состояний автомата на классы нуль-эквивалентных состояний.

Шаг 1: Увеличить k на 1. Выполнить разбиение множества состояний автомата на классы k -эквивалентных состояний. Это делается путем анализа переходов из состояний текущих классов в классы $(k-1)$ -эквивалентных состояний. Если состояния в одном классе переходят в разные классы $(k-1)$ -эквивалентности по одному и тому же входному символу, то текущий класс расщепляется.

Шаг 2: Сравнить полученное разбиение множества состояний на классы k -эквивалентных состояний с разбиением на классы $(k-1)$ -эквивалентных состояний:

- Если разбиение изменилось, перейти к Шагу 1.
- В противном случае (если разбиение не изменилось), получено окончательное разбиение множества состояний автомата на классы эквивалентных состояний.

Обратный автомат — это автомат, который распознаёт "обратные слова".

Построение обратного автомата из исходного автомата (как детерминированного, так и недетерминированного) включает в себя следующие шаги:

Начальные состояния исходного автомата становятся финальными состояниями обратного автомата.

Финальные состояния исходного автомата становятся начальными состояниями обратного автомата.

Направления всех стрелок (переходов) в графе автомата поворачиваются на противоположные.

Обратный автомат играет ключевую роль в **алгоритме Бжозовского**.

Алгоритм Бжозовского — это метод минимизации детерминированного конечного автомата (ДКА), который использует понятие обратного автомата и детерминизации. Он позволяет построить минимальный ДКА из недетерминированного конечного автомата (НКА).

Если взять недетерминированный конечный автомат (НКА), сделать из него обратный автомат, а затем детерминизировать (используя стандартный алгоритм построения подмножеств), то полученный ДКА будет минимальным для обращённого языка.

Алгоритм Бжозовского просто применяет эту идею дважды. Выполнение операции один раз даёт минимальный автомат для обратного языка. Выполнение её второй раз возвращает вас к исходному языку, и в результате вы получаете минимальный автомат для него.

Алгоритм Бжозовского выглядит так:

1. Создаём обратный автомат
2. Детерминизируем его
3. Создаём снова обратный автомат
4. Детерминизируем его

7. Сокращение числа состояний частично-определённых преобразователей. Понятие совместимых состояний. Алгоритм Полла и Ангера.

КА-преобразователи минимизируются, как и КА-распознаватели. Только в случае с преобразователями нам не нужно их доопределять, а вместо эквивалентности вводится понятие совместимости:

Пара состояний совместима, если нет входных слов, допустимых в обоих состояниях, которые КА преобразовывает в этих состояниях по-разному.

Определить пары совместимых состояний можно по треугольной таблице. В ней, помимо определения совместимости, применяются следующие правила:

1. Пара состояний совместима безусловно, если у них нет перехода по одному и тому же символу или переход идет в одно и то же состояние. В ячейке на пересечении ставится галка.

2. Пара состояний совместима условно, если у них есть переход по одному и тому же символу в разные состояния, но пара этих состояний является совместимой.

3. В остальных случаях пара является несовместимой.

После заполнения треугольной таблицы состояния надо сгруппировать. Для этого есть два способа: на интуиции или по алгоритму Полла и Ангера. Преимущество алгоритма в том, что он будет давать максимальную минимизацию, а минус – он очень тяжелый и муторный.

В алгоритме есть два этапа:

1. Определение классов максимальной совместимости:

- ❖ Начинаем с множества A – множества всех состояний;

- ❖ После этого циклически, для i от 0 до $n-1$ состояния:

- Каждое множество с i -м состоянием делим на 2: в первом просто убираем i -е состояние, а во втором – все несовместимые с i -м состоянием;

- После этого удаляем все множества, которые являются подмножествами других имеющихся множеств;

2. Поиск минимального замкнутого покрытия:

- ❖ Из всего набора множеств 1-го этапа выбираем минимальный набор множеств так, чтобы каждое состояние исходного КА фигурировало ХОТЯ БЫ в одном из них;

- ❖ Проверяем условия совместимости пар в этих множествах;

- ❖ Если условия нарушены, то выбираем другие надмножества/подмножества.

После этого строим новый КА по результатам группировки.

ПРИМЕЧАНИЕ: если по итогам группировки одно состояние фигурирует в нескольких множествах, то по нему будет переход в то множество, где есть пара совместимых состояний, от которого оно зависит.

8. Канонический метод структурного синтеза. Способы кодирования состояний: с минимальным числом бит, соседнее, унитарный код.

Канонический метод структурного синтеза – метод проектирования структурного автомата, состоящий из 6 этапов:

1. Кодирование входных, выходных символов и состояний. На этом этапе определяется количество структурных входных каналов, выходных каналов и количество триггеров для хранения состояний автомата.

2. Формирование кодированной таблицы переходов (КТП). Тут просто выполняется соответствующая подстановка вместо оригинальных обозначений их закодированных версий из предыдущего пункта. В этой таблице должны фигурировать только входные символы и состояния.

3. Формирование кодированной таблицы функций возбуждения (КТФВ). Она строится на основании КТП из предыдущего пункта. У нас каждый триггер отвечает за свой разряд в кодировке состояний. В заголовке столбцов указаны входные сигналы триггеров, а в ячейках – выходные сигналы. Далее просто сопоставляем данные для значений функций возбуждения соответствующего триггера.

4. Построение диаграмм Вейча и логических выражений (ЛВ) для КС1. Для каждого триггера, на основании данных из КТФВ строится диаграмма Вейча, на которой выделяются контуры – составляющие части ЛВ, которые и являются функциями возбуждения (ФВ). Полученные ФВ уже приводятся в соответствующий базис.

5. Формирование кодированной таблицы выходов (КТВ). Она строится из таблицы выходов автомата с заменой символов на их закодированные версии. Тут и всплывает разница между автоматами Мили и Мура: в автомате Мура выходной символ привязан к состоянию и никак не зависит от входного, поэтому КТВ будет занимать всего две строки; а в автомате Мили – наоборот – выходной алфавит привязан к переходу между состояниями и зависит и от входного символа, и от состояния, поэтому и КТВ у него будет больше.

6. Построение диаграмм Вейча и логических выражений для КС2. По аналогии с п.4 получаем ЛВ для выходных сигналов.

Есть несколько вариантов кодирования состояний:

1. Соседнее кодирование – вид кодирования, когда коды соседних состояний отличаются только на одной позиции. Наиболее популярный способ соседнего кодирования – код Грея, в котором расстояние Хэмминга (разница в кодах между «соседями») равно 1. В этой кодировке при n разрядах можно получить 2^n кодов для состояний. Она же, по совместительству, является кодировкой с минимальным количеством бит.

2. Кодирование с минимальным числом бит – вид кодирования, когда для кодирования выделяется минимально возможное количество бит, которое можно определить по формуле $\log_2 n$ (n – число состояний).

3. Кодирование унитарным кодом бывает двух видов: one-hot (только в одном разряде 1, в остальных – 0) и one-cold (только в одном разряде 0, в остальных – 1). Например, в one-hot кодировке вес Хэмминга (количество «1» в кодировке) всегда равен 1, а расстояние – равно 2. Из недостатков можно отметить, что по итогу количество триггеров определяется количеством состояний.

Третий вопрос (задача)

(на решение задачи дается примерно 15 минут. Ее уже можно попробовать списать)

1. Постройте граф или таблицу переходов машины Тьюринга, выполняющей заданную операцию над словом.

Порядок действий для построения МТ:

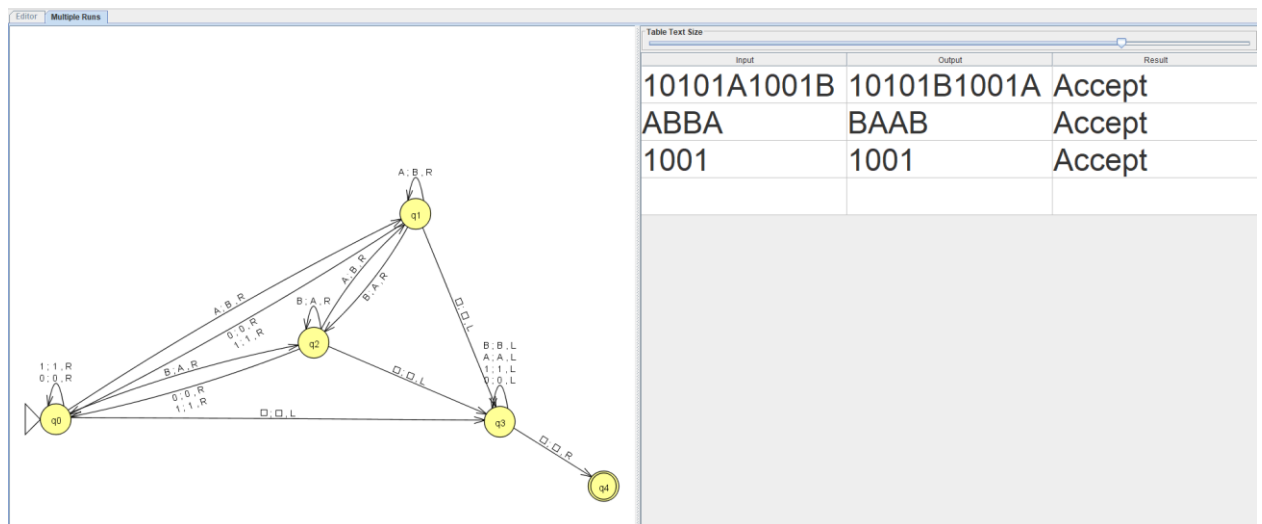
1. Придумайте для начала алгоритм, который будет выполнять эту самую операцию над словом.
2. Придуманный алгоритм разбейте на максимально простые этапы. В дальнейшем, эти самые этапы (если вы правильно раздробили алгоритм) могут стать основой для состояний МТ.
3. Раздробленный алгоритм уже нужно переопределить в состояния МТ и переходы между ними. На этом этапе уже будет получаться нужная МТ. НАСТОЯТЕЛЬНО рекомендую оформлять МТ как граф, т.к. это нагляднее и, следовательно, проще.

Пример (на основе контрольного вопроса из metody):

На ленте расположено слово в алфавите $\{0, 1, A, B\}$, например, 10101A1001B. Составьте таблицу или граф переходов МТ, которая заменит символы A на B, а символы B на A.

Решение:

1. Так как нам нужно тупо заменить A на B, а B – на A, то мы можем игнорировать символы 0 и 1, просто проматывая их. Дойдем до одного из символов для замены – заменим на нужный и пойдем по слову дальше.
2. Предложенную идею можно раздробить на следующие действия:
 - Смотрим текущий символ слова. Если он является 0 или 1, то просто движемся по слову дальше. Дошли до конца слова – перешли в финальное состояние (т.е. алгоритм слово обработал и завершил работу).
 - Если мы встретили символ A, то мы переходим к этапу замены A на B, после этого смотрим следующий символ. Встретили A – опять заменили. Встретили 0 или 1 – вернулись на первый этап алгоритма. Встретили – перешли в следующее состояние. Слово закончилось – идем в финальное состояние.
 - Если мы встретили символ B, то действуем по аналогии с символом A.
3. Строим МТ. На предыдущем этапе у нас шикарно получилось разбить алгоритм на состояния МТ, да еще и переходы между состояниями описали. Поэтому МТ в виде графа будет выглядеть примерно так:



ПРИМЕЧАНИЕ: в случае JFLAP слово надо отмотать на самый первый символ, поэтому финальное состояние пришлось разбить на 2. Уточняем, что в случае с МТ из финального состояния не может быть НИКАКИХ переходов.

Если попросят, то можно этот граф переделать в таблицу:

	q0	q1	q2	q3
0	q0, 0, R	q0, 0, R	q0, 0, R	q3, 0, L
1	q0, 1, R	q0, 1, R	q0, 1, R	q3, 1, L
A	q1, B, R	q1, B, R	q1, B, R	q3, A, L
B	q2, A, R	q2, A, R	q2, A, R	q3, B, L
□	q3, □, L	q3, □, L	q3, □, L	q4, □, R

Расшифровка таблицы:

1. Заголовок столбца указывает, из какого состояния осуществляется переход.
2. Заголовок строки указывает, по какому символу на ленте осуществляется переход.
3. В ячейке указывается (слева направо): состояние, в которое мы переходим; символ, на который мы меняем заполнение текущей ячейки ленты; направление сдвига (R – вправо, L – влево, S – стоим на месте).

2. Постройте граф или таблицу переходов детерминированного конечного автомата, распознающего заданный язык.

По сути, порядок действий тут идентичен порядку действий из [предыдущего вопроса](#). Разница лишь в том, что теперь мы только считываем данные, никак их не меняя.

Пример (в этот раз будет вариант из лабораторной):

Построить КА-распознаватель, распознающий десятичная запись вещественного числа с десятичной точкой и необязательными запятыми, отделяющими каждые три разряда целой части. Например: -5; 45.0; +4,650.03; -100,000,000.0.

Решение:

1. Рассмотрим допустимые записи чисел и определим примерный паттерн правильной их записи:

- Число может начинаться со знака, цифры или вообще точки (последний вариант – чисто программистское допущение, но будем рассматривать именно с ним)
- Если число началось со знака, то после него может идти цифра или точка;
- После точки могут идти только цифры, остальные символы недопустимы;
- В целочисленной части могут идти цифры. Если после первых трех цифр в числе не встретились запятые, то их дальше в числе быть не может. В ином случае, запятые должны повторяться после каждой третьей цифры, начиная с той, которая идет после первой запятой.

2. На основании этих правил можно выделить следующие состояния и переходы для автомата:

- a_0 – начальное состояние автомата, проверка на первый символ. Если первый символ «.», то осуществляется переход в состояние a_1 . Если первый символ «+» или «-» – переход в a_3 . Если первый символ цифра – переход в a_4 .
- a_1 – проверка, что после точки есть цифры. Переход в a_2 .
- a_2 – проверка, что в десятичной части числа записаны только цифры, осуществляет переход в само себя до конца слова по цифрам. Финальное состояние.
- a_3 – проверка, что после знака есть цифра. Переход в a_4 .
- a_4 – проверка текущего символа. Если символ «.», то осуществляется переход в состояние a_1 . Если символ цифра – переход в a_5 . Если символ «,» – переход в a_8 . Финальное состояние.

Звездочкой помечаются финальные состояния. Это свойственно только для таблиц переходов КА-распознавателей.

3. По заданному оператору соответствия постройте автомат Мура (Мили)

Оператор соответствия (ОС) – одна из форм задания автомата-преобразователя в виде таблицы, сопоставляющей выходное слово входному.

Зачастую этот ОС не приведен к автоматному виду. Для этого ОС должен выполнять два условия:

1. Равенство длин. Означает только то, что длина выходного слова должна соответствовать длине входного.

2. Однозначность отображения. Если у нескольких входных слов начала совпадают, то и выходные от них слова так же должны совпадать, причем на всю длину совпадающей части.

Теперь пройдемся по алгоритму построения автоматов, сразу дополняя его примером:

1. Привести ОС к автоматному виду. Этот этап осуществляется путем дописывания «пустых» символов (у входного слова – символ α , а у выходного - β). Пустые символы дописываются у входного слова справа (т.е. в конец), а у выходного – слева (в начало).

Пусть у нас есть следующий оператор соответствия:

ВХОДНЫЕ СЛОВА			ВЫХОДНЫЕ СЛОВА		
0	0	0	0	1	1
0	1	1	1	0	1
1	0	0	1	0	0
1	1	0	1	1	0
1	1	1	0	1	1
1	0	1	1	1	1

Как видно, у нас входные и выходные слова начинаются на 0 и 1, причем нельзя установить зависимость между их началами. Поэтому мы допишем пустые символы к каждому из слов:

z(0)	z(1)	z(2)	z(3)	z(4)	w(0)	w(0 1)	w(1 2)	w(2 3)	w(3 4)
0	0	0	α			β	0	1	1
0	1	1	α			β	1	0	1
1	0	0	α			β	1	0	0
1	1	0	α			β	1	1	0
1	1	1	α			β	0	1	1
1	0	1	α			β	1	1	1

Теперь будем рассматривать начала по двум символам. Начала у входных слов «00» и «01» являются уникальными, поэтому на эти слова можно больше не обращать внимания (оригинальные слова имели длину 3 и были уникальными, поэтому начала длиной в 3 символа будут уникальными).

Начало «10» встречается уже у двух входных слов, но начала соответствующих им выходных слов так же совпадают ($\beta 1$). Так что и эти слова можно дальше не рассматривать.

Начало «11» так же встречается у двух слов, но начала их выходных слов уже не совпадают. Поэтому мы опять дополняем их пустыми символами.

По итогу у нас получается следующий ОС, который был приведен к автоматному виду:

z(0)	z(1)	z(2)	z(3)	z(4)	w(0)	w(0 1)	w(1 2)	w(2 3)	w(3 4)
0	0	0	α			β	0	1	1
0	1	1	α			β	1	0	1
1	0	0	α			β	1	0	0
1	1	0	α	α	β	β	1	1	0
1	1	1	α	α	β	β	0	1	1
1	0	1	α			β	1	1	1

ПОЯСНЕНИЕ: в результате приведения ОС к автоматному виду у нас получились разные по длине слова. Поэтому, чисто технически (хоть и не заметно для нас) эти слова будут обрабатываться с разным временем (время, aka условный тик, указан в скобках)

2. Построить по ОС автомат. Суть построения, что у Мили, что у Мура, одинаковая, и заключается в создании дерева.

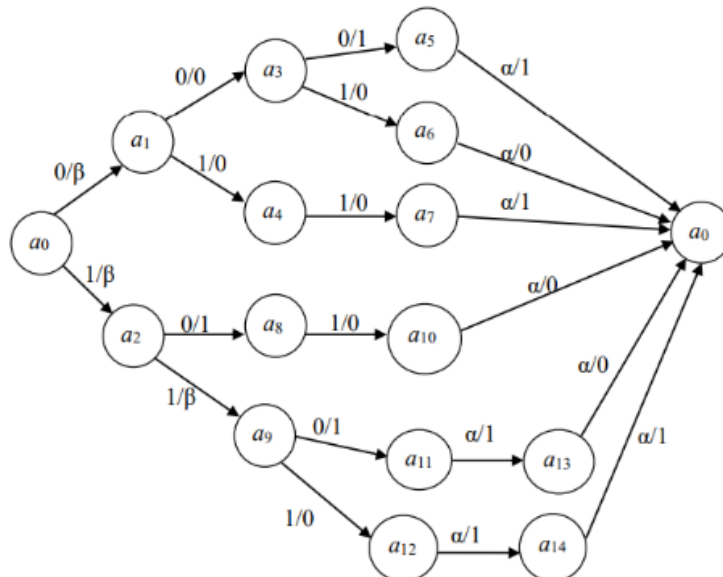
Построение графа для автомата Мили:

1. Ставим начальное состояние.

2. От него уже начинаем проводить разветвление дерева. У нас слова начинаются на 0 и 1. Поэтому ставим два новых состояния и подписываем им переходы, указывая, по какому символу мы переходим и на какой мы его меняем. Если же на вход по оператору соответствия может поступить только один символ, то такое разветвление уже не требуется.

Повторять до предпоследнего символа слова. После него переход должен вести в начальное состояние, заменяя последний символ входного слова на соответствующий последний символ выходного. Для удобства мы создадим состояние, дублирующее начальное, но перенесем этот дубликат в конец.

По итогу должен получиться граф, внешне похожий на этот:



На основании графа уже строится СТПВ ([вопрос 5 из блока «Второй вопрос»](#)):

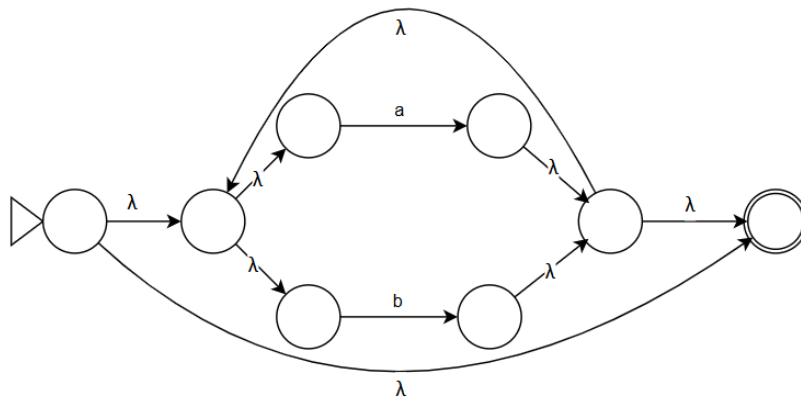
	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
0	a_1/β	$a_3/0$	$a_8/1$	$a_5/1$		-	-	-	-	$a_{11}/1$	-	-	-	-	-
1	a_2/β	$a_4/0$	a_9/β	$a_6/0$	$a_7/0$	-	-	-	$a_{10}/0$	$a_{12}/0$	-	-	-	-	-
α	-	-	-	-	-	$a_0/1$	$a_0/0$	$a_0/1$	-	-	$a_0/0$	$a_{13}/1$	$a_{14}/1$	$a_0/0$	$a_0/1$

Граф для автомата Мура строится почти так же, но имеются два отличия в оформлении:

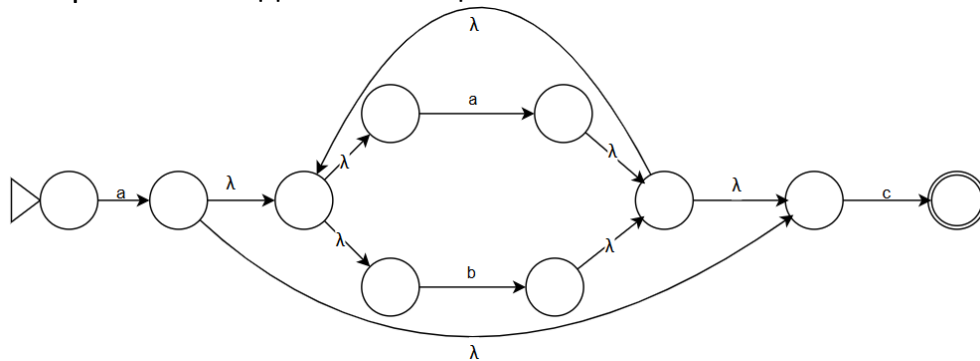
1. Выходные символы записываются в вершинах графа вместе с состояниями, так как выходной символ автомата Мура определяется только состоянием и не зависит от входного символа.

2. Число начальных состояний в автомате Мура определяется количеством различных последних символов в выходных словах ОС.

С учетом этих различий, граф для Мура будет выглядеть внешне схож с этим:



Теперь остается добавить еще «а» и «с» в соответствии с РВ:



Автомат по РВ построен успешно.

Пример 2. (по алгоритму Глушкова)

Для РВ « $(a+b)^*ab$ » постройте автомат-распознаватель.

РЕШЕНИЕ:

1. Произведем линеаризацию РВ. Получим:

$$R_1 = (a_1 + b_2)^* a_3 b_4$$

2. После линеаризации РВ вычислим следующие множества:

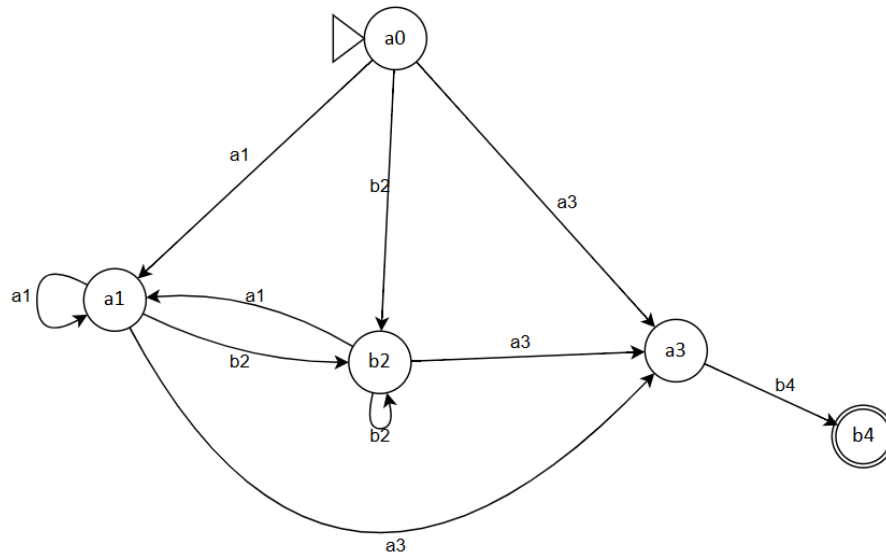
$S = \{a_1, b_2, a_3\}$ (множество первых символов, по ним определяются переходы из начального состояния)

$F = \{b_4\}$ (множество последних символов, определяет количество финальных состояний)

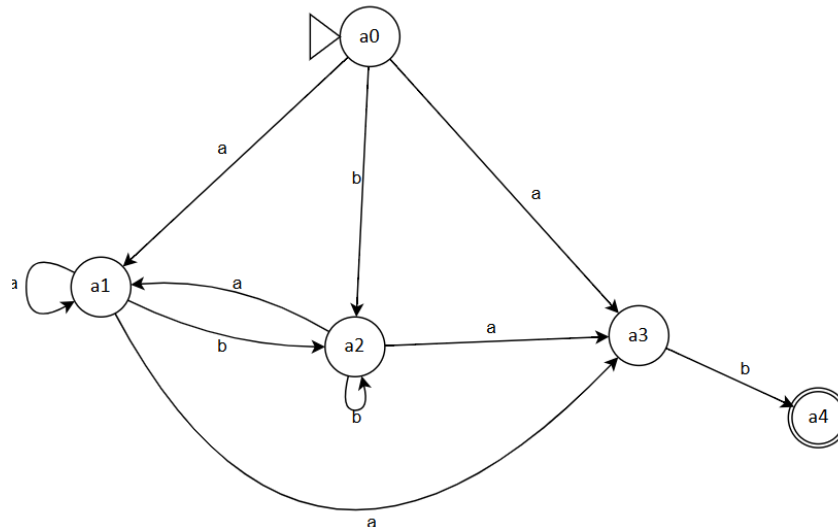
$P = \{a_1 b_2, a_1 a_3, a_1 a_1, b_2 b_2, b_2 a_3, b_2 a_1, a_3 b_4\}$ (множество всех пар символов, определяет переходы между состояниями)

$\Lambda = \emptyset$ (автомат не может обработать пустую строку)

3. Построение КА, распознающего R_1 . Добавляем состояние a_0 , остальные вершины получают имя по соответствующим символам из R_1 . Переходы в эти состояния должны осуществляться по этим же линеаризованным символам. По итогу получается следующий ДКА:



4. Удаляем линейризацию. Именуем состояния как положено (по канону стоит использовать « a_i », где i – номер состояния). По сути, мы просто удаляем индексы у символов РВ, по которым осуществляются переходы между состояниями; ну а если вершина подписана не как « a » с индексом, а как другая буква, то меняем букву на a . По итогу мы получим уже НКА:

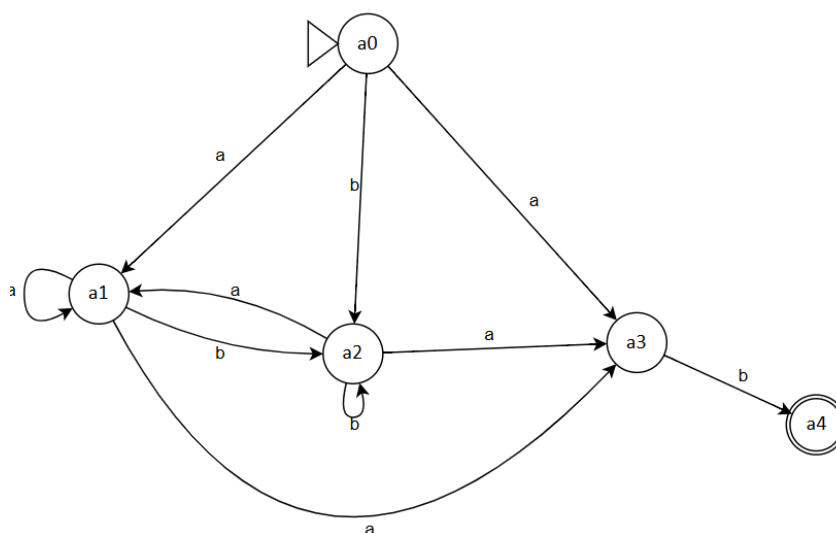


А о том, как из НКА получить ДКА, рассказывается в следующем пункте.

5. Для заданного недетерминированного конечного автомата постройте эквивалентный ему детерминированный.

Так как задача стоит его только детерминировать, то мы остановимся только на детерминации по теории множеств.

Для примера будем детерминировать автомат из предыдущего пункта:



Для начала выделим множество начальных состояний. В нем будет только одно состояние (a_0), а для удобства все результаты будем сводить в таблице, где в первом столбце будут состояния ДКА, а в остальных – множества состояний, в которые осуществляется переход из исходного множества, которое мы уже записали как состояние, ПО ОДНОМУ СИМВОЛУ. И когда у нас перестанут появляться новые множества, мы закончим заполнение таблицы и, следовательно, построение ДКА:

Состояние ДКА	a	b
$>b_0 = \{ a_0 \}$	$\{ a_1, a_3 \}$	$\{ a_2 \}$

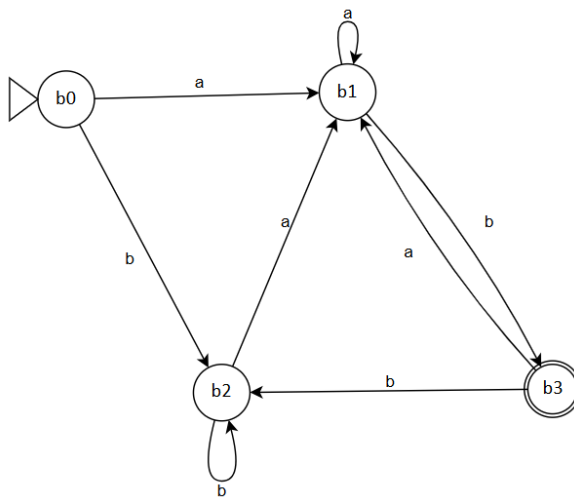
У нас появились два новых множества, занесем их в таблицу и рассмотрим переходы из них:

Состояние ДКА	a	b
$>b_0 = \{ a_0 \}$	$\{ a_1, a_3 \}$	$\{ a_2 \}$
$b_1 = \{ a_1, a_3 \}$	$\{ a_1, a_3 \}$	$\{ a_2, a_4 \}$
$b_2 = \{ a_2 \}$	$\{ a_1, a_3 \}$	$\{ a_2 \}$

Теперь у нас появилось только одно новое состояние. Добавим и его в таблицу:

Состояние ДКА	a	b
$>b_0 = \{ a_0 \}$	$\{ a_1, a_3 \}$	$\{ a_2 \}$
$b_1 = \{ a_1, a_3 \}$	$\{ a_1, a_3 \}$	$\{ a_2, a_4 \}$
$b_2 = \{ a_2 \}$	$\{ a_1, a_3 \}$	$\{ a_2 \}$
$b_3 = \{ a_2, a_4 \}^*$	$\{ a_1, a_3 \}$	$\{ a_2 \}$

В результате новых множеств не появилось, поэтому алгоритм закончен. Теперь строим граф по таблице. По сути, у нас получилась своеобразная транспонированная таблица переходов, только состояния в ней мы зашифровали множествами. Поэтому сразу построим граф:



Готово, мы построили ДКА по НКА. Радостно.

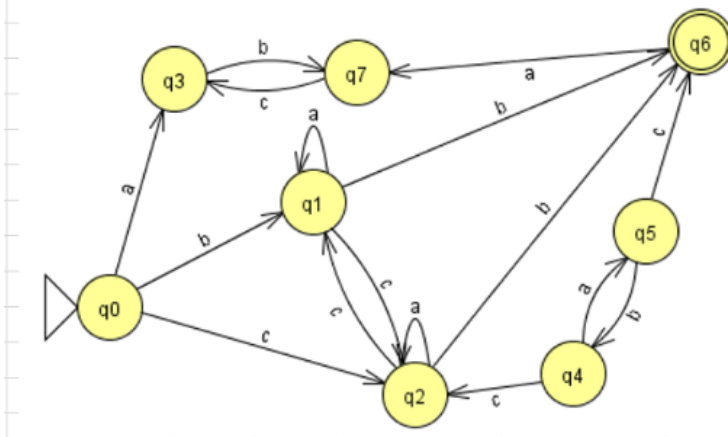
6. Минимизируйте заданный детерминированный конечный автомат.

Минимизация ДКА отличается в зависимости от того, какой КА мы пытаемся минимизировать: распознаватель ([вопрос 6 из блока «Второй вопрос»](#)) или преобразователь ([вопрос 7 из блока «Второй вопрос»](#)).

Пример 1. Минимизация КА-распознавателя.

Минимизируйте автомат-распознаватель:

	q0	q1	q2	q3	q4	q5	q6*	q7
a	q3	q1	q2	-	q5	-	q7	-
b	q1	q6	q6	q7	-	q4	-	-
c	q2	q2	q1	-	q2	q6	-	q3



РЕШЕНИЕ:

Для минимизации автомата-распознавателя нужно выполнить три шага:

1. Удалить недостижимые состояния.
2. Удалить тупиковые состояния.
3. Объединить неразличимые состояния.

Пойдем по порядку:

1. Для начала создадим множество достижимых состояний R.

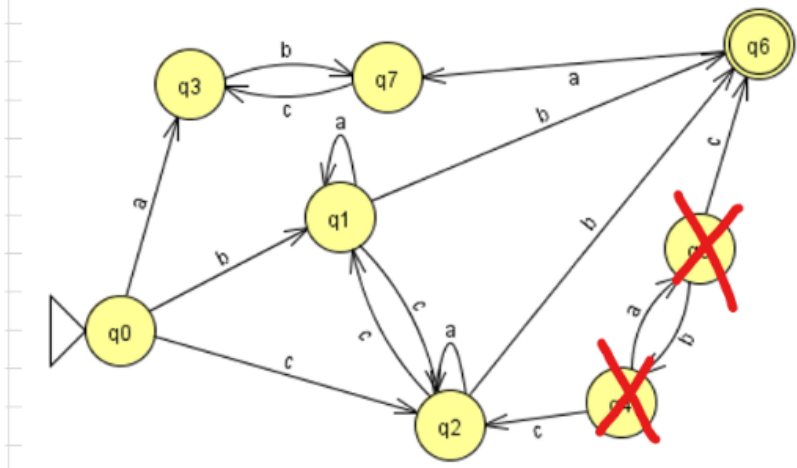
Шаг 0: $R = \{ q_0 \}$

Шаг 1: $R = \{ q_0, q_1, q_2, q_3 \}$

Шаг 2: $R = \{ q_0, q_1, q_2, q_3, q_6, q_7 \}$

Шаг 3: $R = \{ q_0, q_1, q_2, q_3, q_6, q_7 \}$ - Конец. Убираем q_4, q_5 .

	q0	q1	q2	q3	q4	q5	q6*	q7
a	q3	q1	q2	-	q5	-	q7	-
b	q1	q6	q6	q7	-	q4	-	-
c	q2	q2	q1	-	q2	q6	-	q3



1. Для начала создадим множество нетупиковых состояний N.

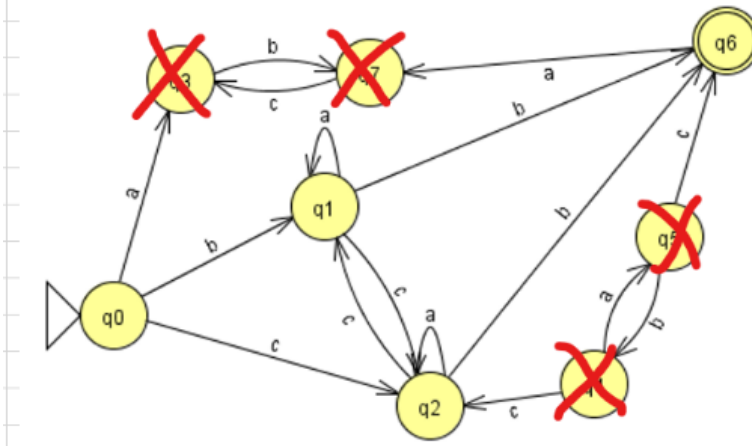
Шаг 0: $N = \{ q_6 \}$

Шаг 1: $N = \{ q_2, q_1, q_6 \}$

Шаг 2: $N = \{ q_0, q_2, q_1, q_6 \}$

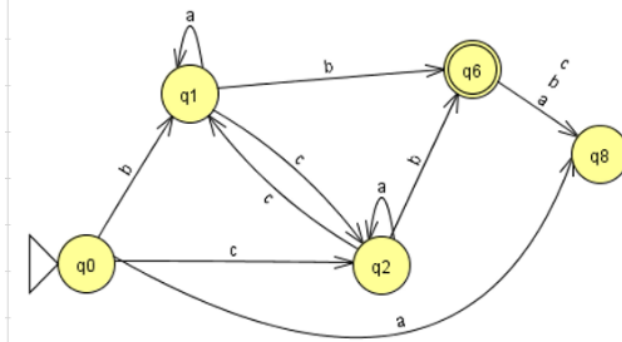
Шаг 3: $N = \{ q_0, q_2, q_1, q_6 \}$ - Конец. Убираем q_7, q_3

	q0	q1	q2	q3	q4	q5	q6*	q7
a	q3	q1	q2	-	q5	-	q7	-
b	q1	q6	q6	q7	-	q4	-	-
c	q2	q2	q1	-	q2	q6	-	q3



3. Для объединения эквивалентных состояний нам нужно, чтобы автомат был полностью определен. Поэтому мы намеренно добавим тупиковое состояние q_8 , чтобы доопределить автомат. Потом мы его уберем

	q_0	q_1	q_2	q_6^*	q_8
a	q_8	q_1	q_2	q_8	q_8
b	q_1	q_6	q_6	q_8	q_8
c	q_2	q_2	q_1	q_8	q_8



Для начала воспользуемся методом расщепления классов:

- Шаг 0: $E_0 = \{ F, A-F \} = \{ b_0, b_1 \}$
 - $F = \{ q_6 \} = b_0$
 - $A-F = \{ q_0, q_1, q_2, q_8 \} = b_1$

	A-F b_1				F b_0
	q_0	q_1	q_2	q_8	q_6^*
a	b_1	b_1	b_1	b_1	b_1
b	b_1	b_0	b_0	b_1	b_1
c	b_1	b_1	b_1	b_1	b_1

- Шаг 1: $E_1 = \{ c_0, c_1, c_2 \}$
 - $c_0 = \{ q_6 \}$
 - $c_1 = \{ q_0, q_8 \}$
 - $c_2 = \{ q_1, q_2 \}$

	c_2		c_1		c_0
	q_1	q_2	q_0	q_8	q_6^*
a	c_2	c_2	c_1	c_1	c_1
b	c_0	c_0	c_2	c_1	c_1
c	c_2	c_2	c_2	c_1	c_1

• **Шаг 2:** $E2 = \{ d0, d1, d2, d3 \}$

- $d0 = \{ q6 \}$
- $d1 = \{ q0 \}$
- $d2 = \{ q8 \}$
- $d3 = \{ q1, q2 \}$

	d3		d2	d1	d0
	q1	q2	q8	q0	q6*
a	d3	d3	d2	d2	d2
b	d0	d0	d2	d3	d2
c	d3	d3	d2	d3	d2

• **Шаг 3:** $E3 =$ получится то же самое - разбиение окончено

Как видно, неразличимыми являются состояния $q1$ и $q2$. Они и будут объединены в одно. Отбросим $q8$ – и получим наш минимальный автомат-распознаватель.

Теперь рассмотрим альтернативный способ – с помощью треугольной таблицы:

Рисуем треугольную таблицу для пар состояний. По вертикали от $q1$ до q_n , по горизонтали от $q0$ до q_{n-1} .

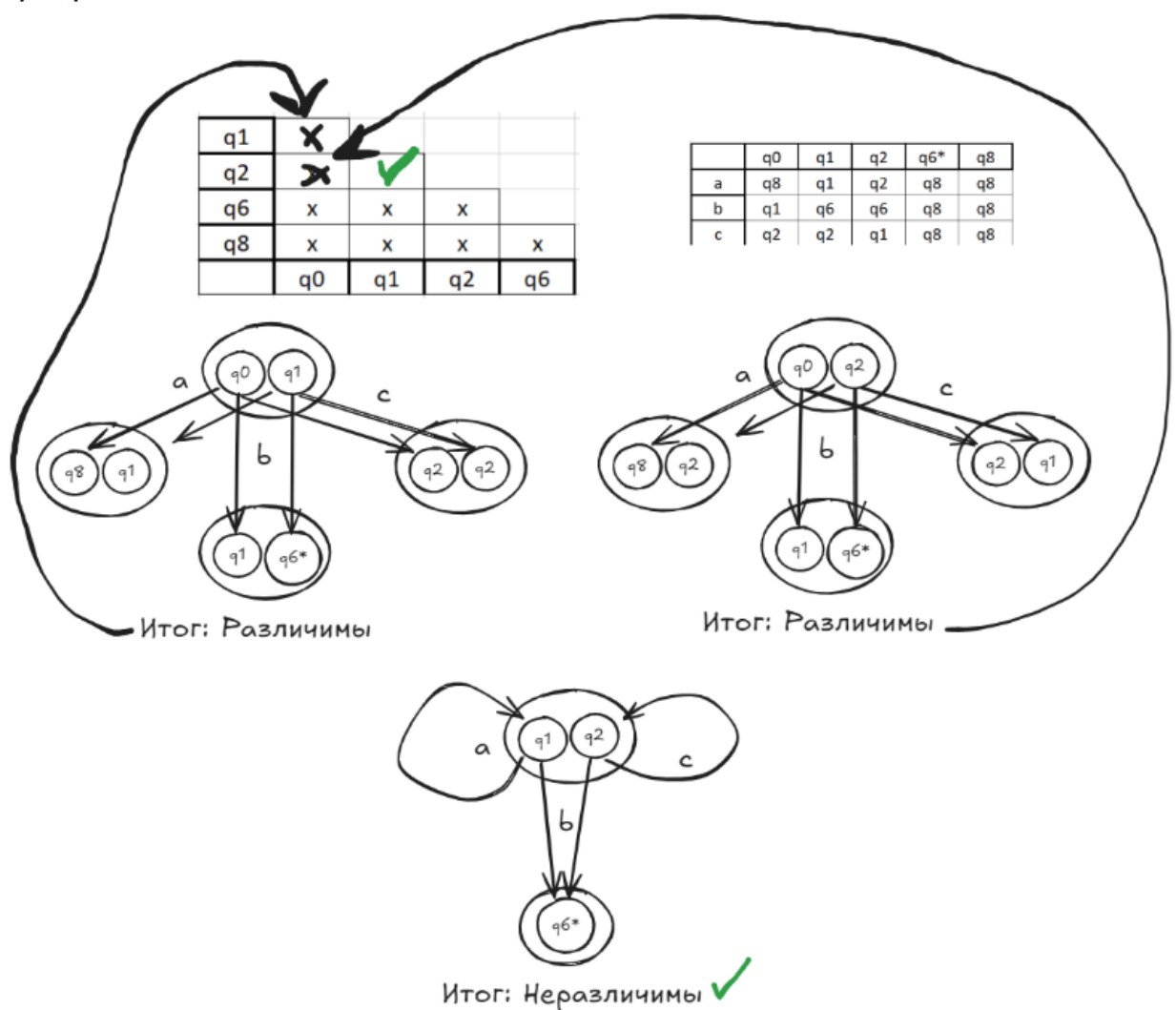
q1				
q2				
q6				
q8				
	q0	q1	q2	q6

Теперь берём пересечение состояний, то есть пару (например, $q0 q1$) и заполняем их пересечение либо x крестиком, если состояния различимы, либо v галочкой, если состояния не различимы.

Определяется Различимость по вот таким правилам:

1. При одинаковом символе на вход состояния переходят в разные состояния - Различимы
2. Если состояние переходит в уже Различимую пару, тогда эта пара тоже различима.

Действия повторять до полного заполнения таблицы



Итог: состояния q1 и q2 эквивалентны

Как видно, оба метода дали один и тот же результат. Нихуя не радостно, так как эту хуйню выучить еще надо.

Пример 2. Минимизация КА-преобразователя.

Минимизируйте автомат Мили, заданный следующей СТПВ:

	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
0	a1/6	a3/0	a5/1	a7/1	-	a10/0	a8/1	-	-	-	-
1	a2/6	a4/1	a6/6	-	a7/0	a7/1	a9/0	-	-	-	-
α	-	-	-	-	-	-	-	a0/1	a10/1	a7/1	a0/0

В случае с автоматом-преобразователем будет лишь разница в том, что мы вместо классов эквивалентности будем искать классы совместимости. Ищутся они по треугольной таблице. Всегда. Без исключений.

В случае с автоматом Мили пара состояний считается совместимой безусловно, когда у них нет перехода по одному и тому же входному символу, или такой переход есть, но он ведет в одно и то же состояние с одним и тем

же выходным символом. В случае такой совместимости в ячейке пары ставится V.

Пара состояний считается условно совместимой, когда у них есть переход по одному и тому же символу в разные состояния, но выходной символ у них один и тот же, а совместимость данной пары установлена. В таком случае в ячейку вписывается пара состояний, которая должна быть совместимой, а рядом указывается, совместима она (V) или нет (X).

Например, по СТПВ видно, что состояния a_7 - a_{10} безусловно совместимы с состояниями a_0 - a_6 . А пара a_7 a_8 совместима, если пара a_0 a_{10} является совместимой (но ее совместимость была доказана в прошлом предложении).

Применяя данные правила, получим следующее заполнение для треугольной таблицы:

a_1	X									
a_2	X	X								
a_3	X	X	5,7V							
a_4	X	X	X	V						
a_5	X	3,10V 4,7V	X	X	X					
a_6	X	X	X	7,8V	7,9V	X				
a_7	V	V	V	V	V	V	V			
a_8	V	V	V	V	V	V	V	0,10V		
a_9	V	V	V	V	V	V	V	0,7V	7,10X	
a_{10}	V	V	V	V	V	V	V	X	X	X
	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9

После заполнения таблицы стоит вынести пары совместимых состояний. Если пара совместима условно, то такую пару стоит пометить *:

(0, 7)	(1, 5)*	(2, 3)*	(3, 4)	(4, 6)*	(5, 7)	(6, 7)	(7, 8)*
(0, 8)	(1, 7)	(2, 7)	(3, 6)*	(4, 7)	(5, 8)	(6, 8)	(7, 9)*
(0, 9)	(1, 8)	(2, 8)	(3, 7)	(4, 8)	(5, 9)	(6, 9)	
(0, 10)	(1, 9)	(2, 9)	(3, 8)	(4, 9)	(5, 10)	(6, 10)	
	(1, 10)	(2, 10)	(3, 9)	(4, 10)			
			(3, 10)				

А теперь идет самая веселая часть – мы будем группировать состояния, причем просто методом тыка (меньше времени потратим). Если пара состояний совместима безусловно, то их при любом раскладе можно пихать в

одну группу. Главное, чтобы в рамках группы все состояния были совместимы друг с другом.

В случае же с условной совместимостью нам нужно, чтобы пара, от которой зависит наша совместимость, уже в какой-либо из групп фигурировала.

Например, следуя всем этим правилам, можно получить следующую группировку:

$b_0 = \{a_0, a_7, a_9\}$, $b_1 = \{a_1, a_8\}$, $b_2 = \{a_2, a_{10}\}$, $b_3 = \{a_3, a_4\}$, $b_4 = \{a_5\}$, $b_5 = \{a_6\}$.

По этой группировке составим итоговую СТПВ:

	b_0	b_1	b_2	b_3	b_4	b_5
0	$b_1/6$	$b_3/0$	$b_4/1$	$b_0/1$	$b_2/0$	$b_1/1$
1	$b_2/6$	$b_3/1$	$b_5/6$	$b_0/0$	$b_0/1$	$b_0/0$
α	$b_0/1$	$b_2/1$	$b_0/0$	-	-	-

Для автомата Мура суть и последовательность действий одна и та же. Единственное отличие в том, что требование совпадения по символам у нас переползает с переходов на сами состояния: если состояния возвращают разный выходной символ, то они несовместимы.

7. По заданной кодированной таблице переходов автомата постройте минимальное выражение для функции возбуждения заданного триггера.

Итак, тут нам наступает просто пиздец (т.к. это тема шестой лабы ☠).

В общем, рассмотрим следующий пример:

Дана следующая кодированная таблица переходов (КТП):

$Q_2Q_1Q_0$	000	001	010	011	100
x_1x_0					
00	001	010	100	001	010
01	010	011	011	100	100
10	000	010	000	-	000

Нам нужно построить функцию возбуждения (ФВ) для триггеров типа Т.

Для начала нам нужно вспомнить три факта:

1. Как работает нужный нам триггер. В нашем случае – это Т-триггер, он реагирует на переключение (когда логическое значение $Q(t)$ не совпало со значением $Q(t+)$, значение в следующем такте не совпало со значением на входе).

2. На каждый разряд кодировки состояний задается свой триггер. У нас состояние кодируется тремя разрядами, поэтому нужно 3 триггера.

3. В заголовке столбцов КТП у нас указано исходное состояние ($Q(t)$ для каждого из разрядов), а в ячейке – следующее состояние ($Q(t+1)$ для каждого из разрядов).

А теперь на основе этих данных нам нужно заполнить кодированную таблицу функций возбуждения (КТФВ):

$Q_2Q_1Q_0$ x_1x_0	000			001			010			011			100		
	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0
00															
01															
10															

Будем рассматривать все значения для триггера T_2 . Для определения его значения нам достаточно смотреть только на старшие разряды кодировок состояний.

Рассмотрим ВСЕ переходы из состояния 000: 001, 010, 000. Как видно старший разряд у нас никак не поменялся. Тогда значения ФВ для триггера T_2 будет везде равно 0.

$Q_2Q_1Q_0$ x_1x_0	000			001			010			011			100		
	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0
00	0														
01	0														
10	0														

Аналогично и для переходов из состояния 001: 010, 011, 010. Получим:

$Q_2Q_1Q_0$ x_1x_0	000			001			010			011			100		
	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0
00	0			0											
01	0			0											
10	0			0											

Теперь рассмотрим переходы из 010: 100, 011, 000. Как видно, в переходе по символу 00 у нас старший разряд кодировки состояния поменялся. Поэтому там триггер примет значение 1:

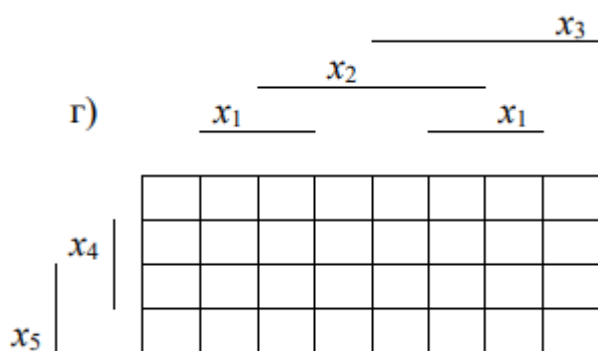
$Q_2Q_1Q_0$ x_1x_0	000			001			010			011			100		
	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0
00	0			0			1								
01	0			0			0								
10	0			0			0								

Теперь по аналогии дозаполним таблицу для триггера T_2 . Прочерк для триггера будет ставиться, если нет перехода из состояния по данному символу:

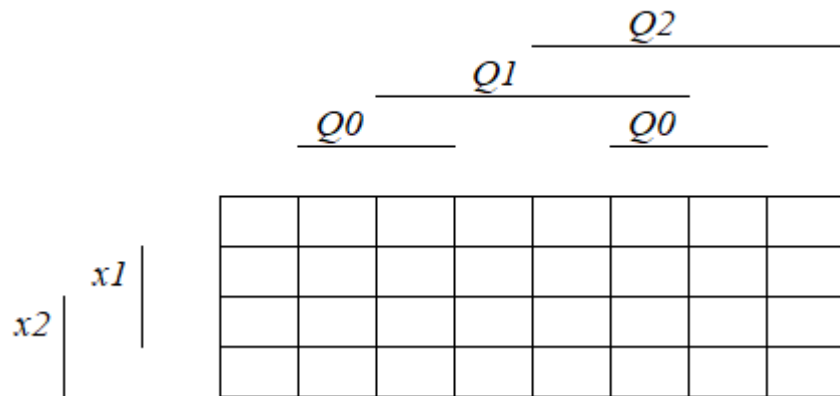
$Q_2Q_1Q_0$ x_1x_0	000			001			010			011			100		
	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0	T_2	T_1	T_0
00	0			0			1			0			1		
01	0			0			0			1			0		
10	0			0			0			-			1		

Теперь для определения минимальной ФВ для нашего триггера построим диаграмму Вейча.

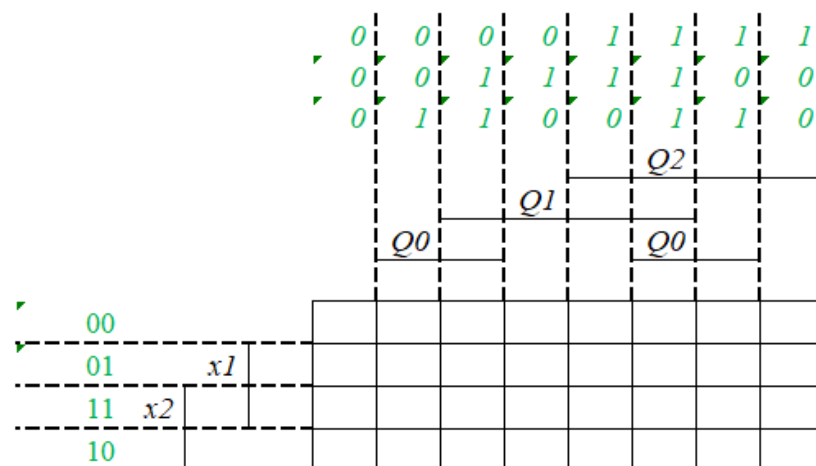
Для начала мы берем нужный нам паттерн диаграммы. В нашем случае нас интересует паттерн на 5 неизвестных. Интересует нас именно он, т.к. у нас 3 символа кодировки задают состояние, а остальные 2 – символы входного алфавита. Итого, 5 переменных:



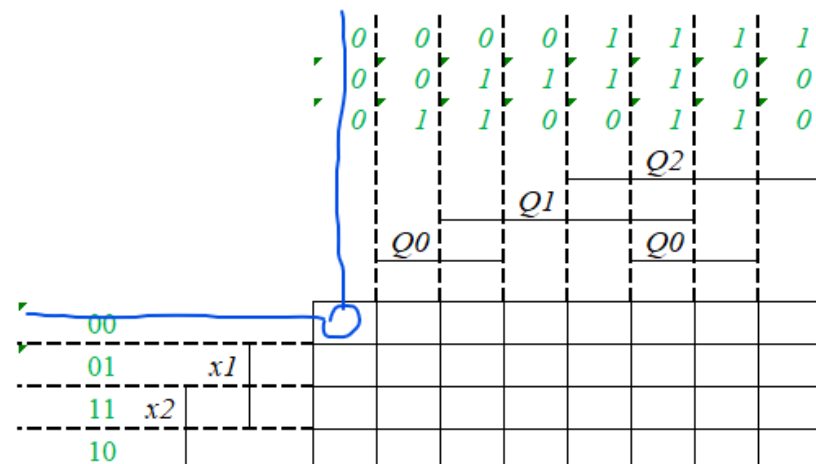
Теперь мы обозначим переменные нужным нам образом. Сверху у нас будут обозначения для кодировки состояний, а слева – кодировки входного алфавита:



А теперь самое неприятное. Ее надо заполнить. Для упрощения процесса заполнения мы «достроим» ей разметку для подписи состояний и символов входного алфавита (зачем мы это сделали, поймете позже):



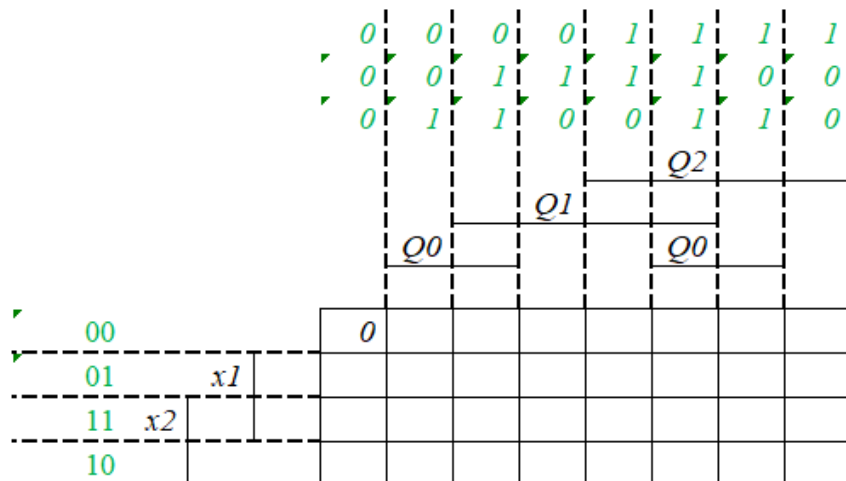
Допустим, мы хотим заполнить левую верхнюю ячейку, отпустим от нее лучи вверх и влево:



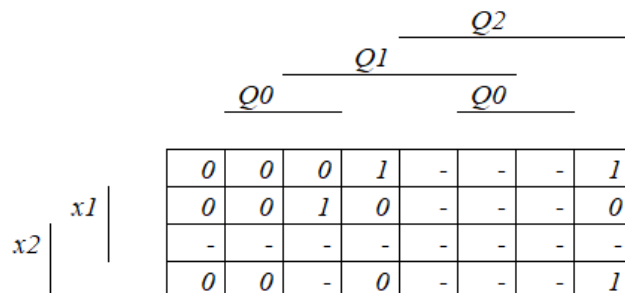
А теперь вопрос: нахрена мы эти лучи отпустили?

Ответ прост, для получения информации, из какой ячейки КТФВ нам нужно взять значение ФВ нашего триггера. Тут действует простое правило: если луч пересек какую-либо из длинных черточек (которые у нас были по диаграмме Вейча), то значение этой переменной, которой черточка принадлежит, равно 1. Если же мы не пересекли черточку, то значение этой

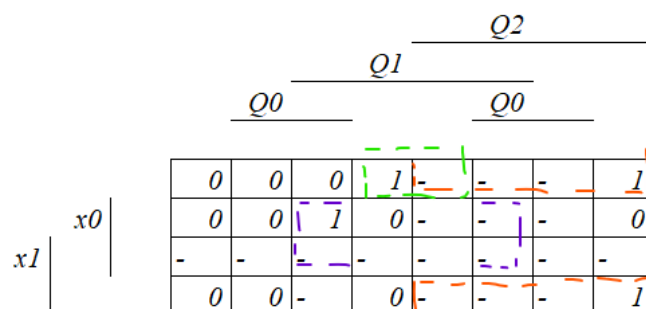
переменной равно 0. Синими лучами мы не пересекли ни одну из черточек. А это значит, что нас интересует переход, где $Q_2=0$, $Q_1=0$, $Q_0=0$, $x_2=0$, $x_1=0$. Т.е. нам нужно глянуть в ячейку для триггера T_2 , где осуществляется переход из состояния 000 по символу 00. А там у нас записан 0:



Кто-то уже мог заметить, что у нас нет перехода по символу, который закодирован как 11. Это значит, что триггер для этого значения не определен. Поэтому всю строку можно заполнить прочерками. По итогу должна получиться следующая диаграмма Вейча:



Теперь, самая простая часть – выделить на диаграмме «секторы». Их нужно делать максимально большими и в минимальном количестве. Это единственные рекомендации, т.к. четкого алгоритма нет. Конечно, есть сайты, которые просчитывают значения диаграмм Вейча, причем они получаются минимальными. По итогу получилась такая разметка по секторам:



Последнее, что нам нужно сделать – это выписать ФВ по этим секторам. Она будет выглядеть как логическое выражение из операций логического

пересечения, соединения и отрицания. Например, рассмотрим фиолетовый сектор. По кодировке символов ему обязательно нужно, чтоб x_0 принимал значение 1, а на x_1 ему похуй (оно может быть равно 1, а может и 0, от этого оно не зависит). Если смотреть по вертикали, то нам нужно обязательно, чтобы Q_1 и Q_0 принимали значение 1, а на Q_2 так же похуй. По итогу, по фиолетовому сектору получим такую часть ФВ:

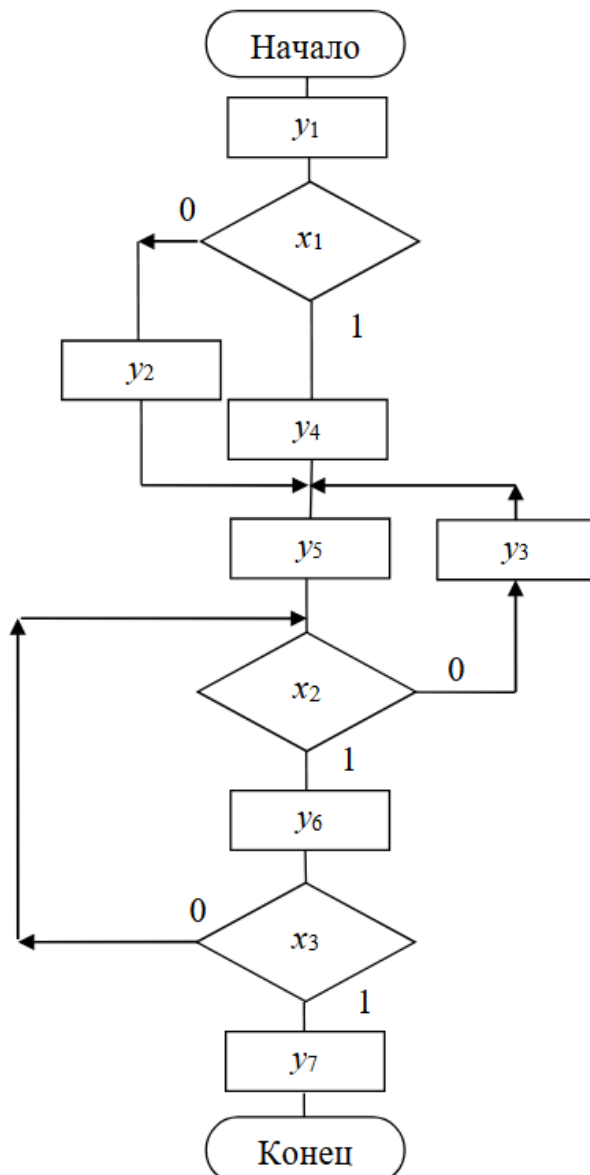
$$Q_1 Q_0 x_0$$

По аналогии с другими секторами получим следующее значение ФВ для триггера T_2 :

$$T_2 = Q_1 Q_0 x_0 + Q_2 \overline{x_0} + Q_1 \overline{Q_0} \overline{x_1} \overline{x_0}$$

8. По заданной графической схеме алгоритма постройте граф автомата Мили (Мура), выполните кодирование состояний заданным способом.

Пример. Дана графическая схема алгоритма (ГСА):



Нужно по нему построить автомат Мили и закодировать состояния.

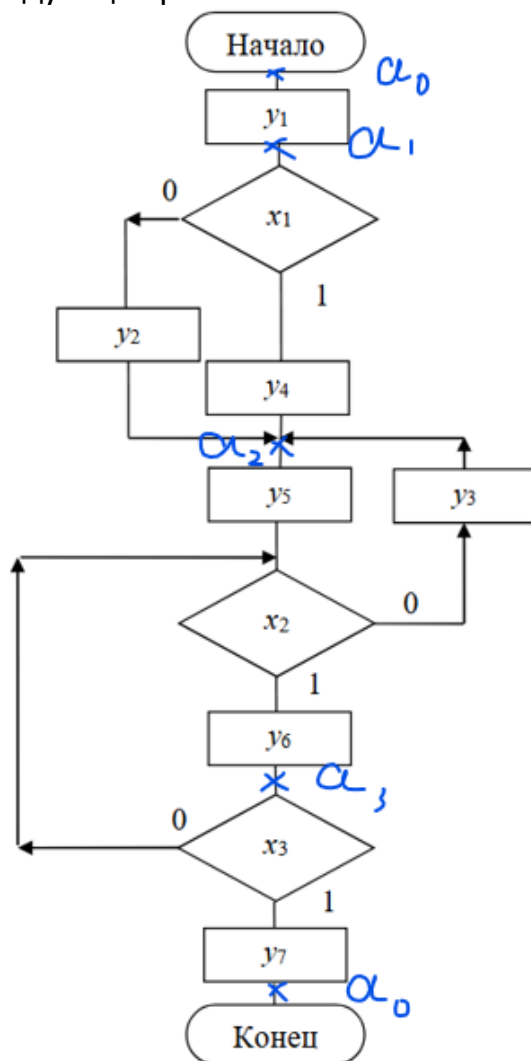
РЕШЕНИЕ:

Для начала сделаем разметку ГСА.

В случае с автоматом Мили она делается по следующему паттерну:

- Вход каждой вершины после y_i – это a_i состояние нашего автомата;
- Вход вершины после начальной и перед конечной вершинами – это состояние a_0 .

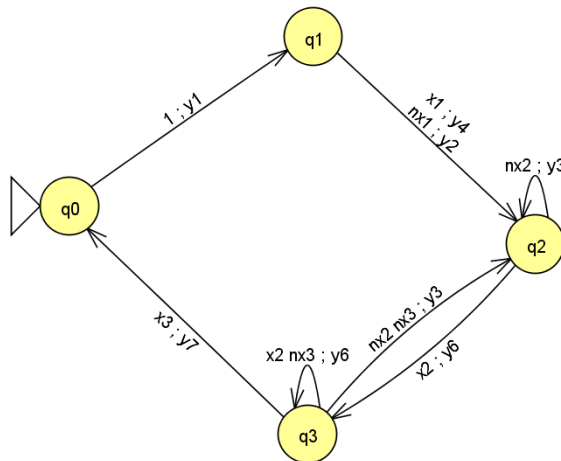
Нам нужно на ГСА поставить состояния между y_i так, чтобы их было по минимуму. Например, у нас точно будет идти состояние a_1 после y_1 , а вот переходы по y_2 , y_3 и y_4 у нас будут вести в одно состояние – a_2 . По итогу у нас будет следующая разметка:



Теперь по этой разметке нужно построить автомат.

Если с выходными символами все понятно (это y_i), то входные будут сниматься как значения по x_j . Если же переход безусловный (как из a_1 в a_1), то на вход будет подаваться 1. Например, переход из a_1 всегда будет вести в a_2 , только если x_1 равен 1 (условие « x_1 »), то на выходе должно вывести y_4 , иначе (условие « $\neg x_1$ ») выведет y_2 .

Должно получиться следующее:



Теперь нужно закодировать наши состояния. Вариантов несколько, поэтому попробуем свести их в таблицу:

Состояние	Последовательная	Код Грея	Код Джонсона	one-hot
a0	00	00	00	0001
a1	01	10	01	0010
a2	10	11	11	0100
a3	11	01	10	1000

Краткое описание: кодировок из примера

1. Последовательная – в представлении не нуждается, просто записываем по порядку в двоичном формате: от 0 и пока состояния не закончатся (только длина кодировки должна быть всегда одинакова у состояний). При n разрядах позволяет сделать 2^n кодировок

2. Код Грея – самая популярная соседняя кодировка. В этом случае все соседние состояния отличаются друг от друга только по одному символу кодировки (тогда говорят, что расстояние Хэмминга равно 1). При n разрядах позволяет сделать 2^n кодировок.

3. Код Джонсона – вид кодировки, когда ее разрядность выше минимальной на 1. Паттерн у кодировки простой: нулевое состояние кодируется $n+1$ нулями (n – минимальное число разрядов для кодирования, определяется как логарифм от кол-ва состояний по основанию 2), после этого каждый разряд, начиная с младшего, заполняется единицами. Когда все разряды заполнились 1, они (так же с младшего разряда) заполняются нулями, пока не останется только одна единица – в самом старшем разряде.

4. one-hot – унарная кодировка. Один разряд равен 1, остальные – равны 0. Количество разрядов задается количеством состояний напрямую (сколько состояний – столько и разрядов).

ПРИМЕЧАНИЕ: Получить разметку для автомата Мура сравнительно проще. Начало и конец ГСА определяются как a_0 , а остальные состояния получают свой индекс у игроков (y_i).