GeekBrains

Конечный автомат

Диаграммы состояний

Оглавление

Введение

Термины, используемые в лекции

Введение в конечные автоматы

- Конечный автомат и диаграмма состояний
- Основные элементы конечного автомата
  - Пример таблицы переходов
  - Пример. Электрический чайник
  - Пример. Система контроля доступа в здание
  - Пример. Проверка синтаксиса арифметического выражения
- Применение конечных автоматов
  - Полезность для процессов
  - Полезность для команды разработки
- Типы конечных автоматов
  - Детерминированные конечные автоматы
  - Недетерминированные конечные автоматы
  - Конечные автоматы с памятью
  - Машина Тьюринга
  - Пример. Конечный автомат для описания статуса обучения

Моделирование жизненного цикла объекта

- Модель CRUD
- Усложнение конечных автоматов

Домашнее задание

- Кейс 1: программа для записи к врачу

Что можно почитать еще?

Введение

На прошлой лекции мы узнали, что такое диаграммы состояний, из каких элементов они состоят и чем будут полезны разным специалистам. На этой лекции рассмотрим одно из прикладных применений диаграмм — теорию конечных автоматов.

Конечные автоматы — один из основных концептов в области компьютерных наук и инженерии. Это математическая модель для описания динамики и поведения дискретных систем. Конечные автоматы применяются в автоматизации, алгоритмах, компиляции, коммуникационных сетях и других областях.

В этой лекции мы рассмотрим основы конечных автоматов: их концепцию, классификацию, конструкцию и примеры использования. Эта тема нужна, чтобы показать многообразие подходов при работе с диаграммами состояний.

Во второй части мы усложним работу с конечными автоматами и рассмотрим дополнительные вводные к описаниям систем. В частности, модель CRUD (create, read, update, delete) в контексте конечных автоматов и исключения при разработке автоматов. Также разберем ряд усложнений, которые позволят охватить больше кейсов при проектировании систем.

На этой лекции вы узнаете:

- Что такое конечные автоматы.
- Какие есть типы конечных автоматов.

Термины, используемые в лекции

Конечные автоматы — это математическая модель, которая используется для описания конечных состояний, в которых может находиться система. Часто используется в компьютерной науке для моделирования автоматических процессов.

Граф — структура данных, состоящая из ребер, которые связывают узлы. Их можно использовать для моделирования связей между элементами или для представления сетей и других взаимосвязанных систем.

Стек — это структура данных, которая поддерживает принцип LIFO (Last-In-First-Out), то есть последний добавленный элемент выходит первым. Используется для хранения данных или для вычислений.

Введение в конечные автоматы

Конечный автомат — это математическая модель, которая используется для описания компьютерных алгоритмов или систем управления. Представляет собой конечное число состояний, переходов между состояниями и входных символов, которые вызывают переходы.

Простейший пример — автомат, который проверяет, является ли входная строка четным числом.

> 📌 Чтобы определить, является ли число четным, можем перевести его из десятичной системы счисления в двоичную и посмотреть, что в конце — 1 или 0. Например:
>
> - $6_{10} = 110_2$ — в конце 0, значит число четное.
> - $7_{10} = 111_2$ — в конце 1, значит число нечетное.
>
> У автомата могут быть состояния «Четное» и «Нечетное», а входными символами могут быть «0» и «1». Если входной символ равен «0», автомат переходит в состояние «Четное», если «1» — «Нечетное».

Конечный автомат и диаграмма состояний

Конечный автомат и диаграмма состояний — это модели для представления автоматических систем.

Конечный автомат — это математическая модель, которая описывает поведение автомата с помощью состояний, событий и переходов между состояниями.

Диаграмма состояний — это графическое представление конечного автомата, которое показывает состояния (в виде вершин) и переходы между состояниями (в виде ребер).

То есть конечный автомат представляет собой математическую модель, а диаграмма состояний — графическое представление этой модели.

Конечный автомат можно использовать для реализации автоматического контроля в программировании, а диаграмму состояний — для визуального представления процессов и переходов в системе.

Диаграмму состояний можно использовать для анализа процессов и для понимания того, как система будет реагировать на разные события. А конечный автомат — для реализации и автоматизации этих процессов.

Основные элементы конечного автомата

Разберем основные элементы конечного автомата — таблицы переходов и состояния: стартовое, текущее и заключительное.

Таблица переходов — это таблица для описания переходов между состояниями автомата в зависимости от входных символов. Каждая строка соответствует одному состоянию автомата, а каждый столбец — одному символу входной последовательности. Ячейки таблицы содержат номер следующего состояния автомата для данного состояния и входного символа.

Стартовое состояние — состояние, в котором находится конечный автомат в начале работы.

Текущее состояние автомата — состояние, в котором находится автомат в данный момент.

Заключительное состояние — состояние, в котором конечный автомат завершает работу после прохождения всех состояний и входных символов.

Пример таблицы переходов

Текущее состояние	Введенный символ	Следующее состояние
Старт	0	Состояние 1
Состояние 1	1	Состояние 2
Состояние 2	0	Состояние 1

Если автомат находится в стартовом состоянии и получает входной символ «0», он переходит в «Состояние 1». Если автомат находится в «Состоянии 1» и получает входной символ «1», он переходит «Состояние 2» и так далее.

Пример. Электрический чайник

Электрический чайник — отличный пример конечного автомата с двумя состояниями: «В режиме ожидания» и «В процессе нагрева воды».

Стартовое состояние: «В режиме ожидания».

- Пользователь подключает чайник к электросети.
- Чайник находится в ожидания и готов к работе.
- У чайника есть одна кнопка «Включение / выключение». Она в положении «Выключено».
- В этом состоянии чайник не потребляет электроэнергию.

Событие: пользователь нажимает кнопку «Включение / выключение».

- Чайник переходит в состояние «В процессе нагрева воды».
- Чайник начинает потреблять электроэнергию и нагревать воду до заданной температуры.
- Кнопка «Включение / выключение» остается в положении «Включено».

Событие: вода нагрелась до заданной температуры.

- Чайник переходит обратно в состояние «В режиме ожидания».
- Чайник перестает потреблять электроэнергию и готов к использованию.
- Кнопка «Включение / выключение» автоматически переключается в положение «Выключено».

Как только чайник вернулся в исходное состояние «В режиме ожидания», он снова готов к работе и пользователь может повторить процесс.

Таким образом, чайник может занимать два состояния: «В режиме ожидания» и «В процессе нагрева воды», между которыми постоянно перемещается, пока не отключен от электросети.

Конечный автомат для электрического чайника можно представить в виде таблицы переходов. Она показывает, какие события переводят автомат из одного состояния в другое:

Текущее состояние	Событие	Следующее состояние
В режиме ожидания	Нажата кнопка «Включение / выключение»	В процессе нагрева воды

В процессе нагрева воды	Вода нагрелась до заданной температуры	В режиме ожидания

Как видно из таблицы, при стартовом состоянии «В режиме ожидания» чайник переходит в состояние «В процессе нагрева воды», когда пользователь нажимает кнопку «Включение / выключение». А когда вода достигает заданной температуры, чайник автоматически возвращается в исходное состояние «В режиме ожидания». И так далее, пока чайник не отключат от электросети.

Пример. Система контроля доступа в здание

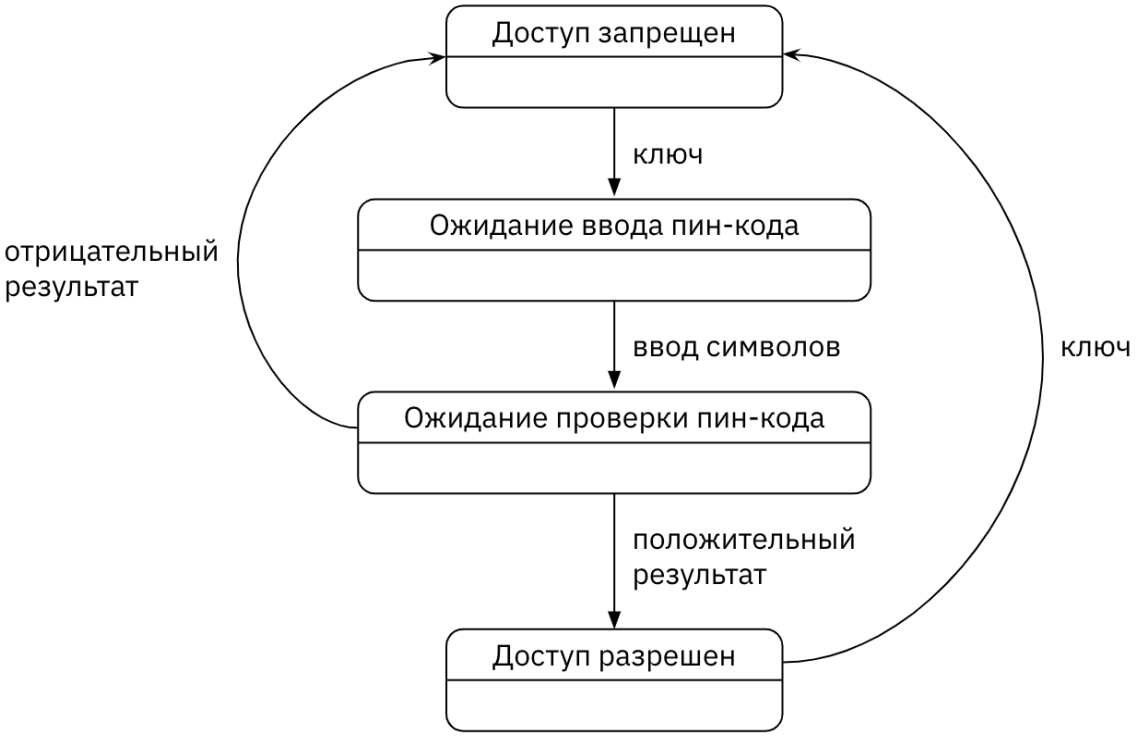
Пример сложного конечного автомата — система контроля доступа в здание. Ее состояния: «Доступ запрещен», «Ожидание ввода пин-кода», «Ожидание проверки пин-кода», «Доступ разрешен».

Таблица переходов:

Текущее состояние	**Входные данные**	**Следующее состояние**
Доступ запрещен	Ключ	Ожидание ввода пин-кода
Ожидание ввода пин-кода	Ввод символов	Ожидание проверки пин-кода
Ожидание проверки пин-кода	Положительный результат	Доступ разрешен
Ожидание проверки пин-кода	Отрицательный результат	Доступ запрещен
Доступ разрешен	Ключ	Доступ запрещен

Стартовое состояние в этом случае — «Доступ запрещен», а заключительное — «Доступ разрешен».

Отобразить этот автомат и все его состояния мы также можем с помощью графических схем.



Пример. Проверка синтаксиса арифметического выражения

Рассмотрим пример конечного автомата, который реализует проверку синтаксиса арифметического выражения.

Состояния автомата:

- start — начальное состояние;
- number — если в выражении встречена цифра;
- operator — если в выражении встречен оператор;
- error — в случае ошибки синтаксиса.

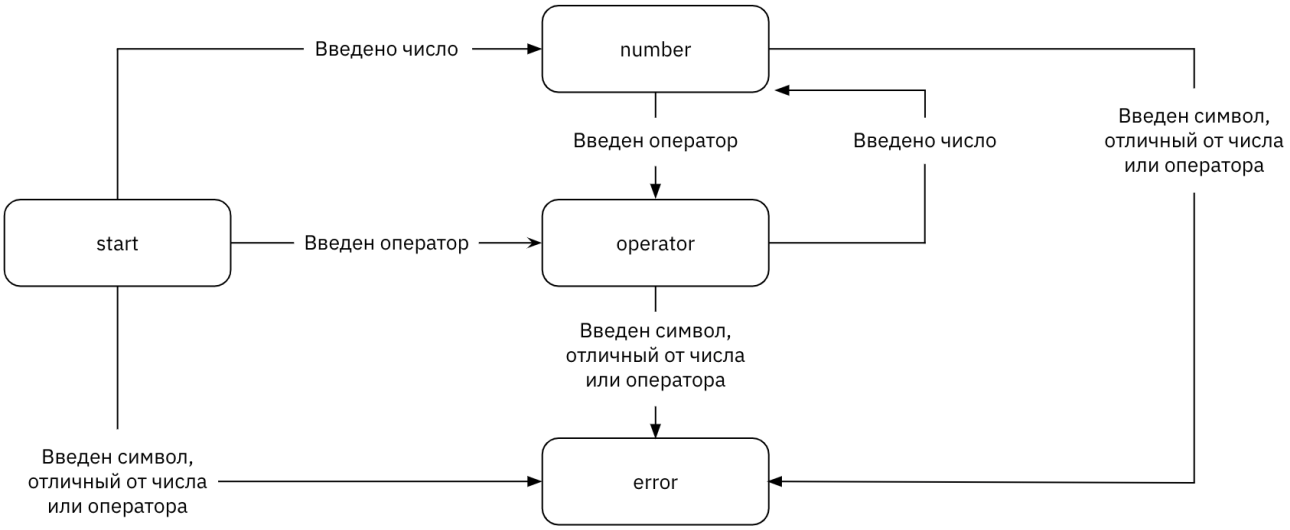
Таблица переходов:

Текущее состояние	Символ	Следующее состояние
start	цифра	number
start	оператор	operator
number	цифра	number

number	оператор	operator
operator	цифра	number
operator	оператор	error
error	*	error

Заключительное состояние — number. Это значит, что выражение считается корректным, если в конце работы автомата текущее состояние станет number.

Теперь нарисуем диаграмму состояний.



Еще раз определим связь диаграммы состояний и конечных автоматов.

> 📌 Конечный автомат состоит из таблицы переходов. А один из инструментов для описания таблицы переходов — диаграмма состояния.
>
> То есть диаграмма состояния — это составная часть конечных автоматов, она может графически описать их состояния.

Применение конечных автоматов

Конечные автоматы часто используются в инженерии и компьютерных науках: компьютерной лингвистике, компьютерной графике, алгоритмах и структурах данных, в системах программирования и компьютерной безопасности. Однако они важны и в повседневных рабочих процессах.

К примеру, с их помощью можно реализовать программу для проверки домашнего задания от студентов GeekBrains.

Прежде чем приступить к разработке, составим конечный автомат. Определим состояния:

- **Не начато** — студент не начал выполнять домашнее задание.
- **В процессе** — студент в процессе выполнения домашнего задания.
- **Отправлено** — студент отправил домашнее задание на проверку.
- **Оценено** — домашнее задание проверили и оценили.

В примере мы немного упростили систему. В конечном автомате состояние «Оценено» может быть не последним: все зависит от требований заказчика или функционала, который мы хотим заложить в систему. Например, в зависимости от требований, мы можем добавить возможность пересдать домашнюю работу.

Таблица переходов для этого конечного автомата:

Текущее состояние	Входные данные	Следующее состояние
Не начато	Начало работы	В процессе
В процессе	Отправка	Отправлено
Отправлено	Оценка	Оценено

Таблица переходов показывает, что из текущего состояния мы переходим в следующее в зависимости от входного сигнала.

Полезность для процессов

Рассмотрим еще один пример, который подчеркивает важность этих инструментов, — **воронку продаж.**

Конечный автомат для воронки продаж можно представить в виде последовательности шагов, которые проходит потенциальный клиент на пути от первого контакта до покупки.

Пример конечного автомата для воронки продаж:

Текущее состояние	Событие	Следующее состояние
Потенциальный клиент	Клиент оставляет контактную информацию	Заинтересованный клиент
Заинтересованный клиент	Клиент просматривает предложения	Потенциальный покупатель
Потенциальный покупатель	Клиент задает вопросы и запрашивает дополнительную информацию	Заинтересованный клиент, обладающий информацией о продукте
Заинтересованный клиент, обладающий информацией о продукте	Клиент получает индивидуальное предложение	Окончательный покупатель
Окончательный покупатель	Клиент подтверждает свою покупку	Завершенная продажа

Преимущества решения:

1. **Процесс прозрачный и контролируемый.** Конечный автомат для воронки продаж помогает учитывать каждый шаг, который проходит потенциальный клиент на пути к завершенной продаже.
2. **Упрощает определение взаимосвязей.** Воронка продаж — простой концепт, но, если не использовать диаграммы состояний для ее формализации, правильно определить уровни воронки и учесть их взаимосвязи на практике бывает сложно.
3. **Помогает генерировать и формализовать идеи.** Конечные автоматы полезны для бизнеса не только в практическом плане, но и для формализации продуктов или идей. Например, для проверки домашних заданий в онлайн-школе можно использовать диаграмму состояний, которая позволит менеджерам эффективно донести свое видение разработчиками и избежать ошибок при реализации.

4. **Помогает наладить взаимодействие между отделами.** Диаграммы состояний позволяют формализовать взаимодействие разных отделов в компании и подружить гуманитариев и технарей.
5. **Помогает обнаружить ошибки.** Диаграммы состояний подсвечивают ошибки, которые сложно заметить, благодаря детальному рассмотрению процессов.

Таким образом, диаграммы состояний могут принести большую пользу для бизнеса, помогут лучше понять процессы, формализовать взаимодействие разных отделов и создать практически используемые инструменты.

Полезность для команды разработки

Диаграммы состояний и конечные автоматы полезны для решения конкретных рабочих задач.

Продуктовым и проектным менеджерам они обеспечат понятную визуальную модель процессов и состояний и будут полезны для управления продуктами и проектами.

Примеры задач:

- Управление жизненным циклом продукта — отображение разных состояний продукта и переходов между ними, в том числе в релизах, версиях и исправлениях.
- Управление проектом — отображение и отслеживание этапов проекта, в том числе планирования, разработки, тестирования и выпуска.
- Оценка рисков — моделирование рисков и выявление возможных сценариев для их минимизации.
- Отслеживание сроков исполнения, например, чтобы определять, в какой стадии находится проект.
- Автоматизация рутины, например, отслеживание задач или выполнение действий в ответ на события.

Аналитики могут использовать конечные автоматы для анализа сложных данных, определения тенденций и зависимостей, идентификации паттернов и выявления аномалий. Например, для анализа данных о поведении пользователей в системе и их пути принятия решений. Или для анализа логов транзакций в финансовой системе.

Другие примеры:

- Обработка текстовых данных — анализ текстов, определение тональности, классификация текстов или элементов в них.
- Обнаружение мошенничества — анализ транзакций, определение аномалий в поведении пользователей.
- Маркетинг — анализ данных о поведении покупателей, идентификация паттернов поведения и сегментация аудитории.
- Биоинформатика — анализ генетических данных, поиск зависимостей между генами и заболеваниями.
- Управление производством — мониторинг и контроль производственных процессов, определение аномалий.
- Телекоммуникации — мониторинг сетевых устройств, определение аномалий или ошибок.

Программисты могут использовать конечные автоматы для следующих задач:

- Реализация систем, основанных на состояниях, например, игр, медиаплееров или устройств умного дома.
- Улучшение кода: его читаемости и управляемости. Устранение ненужной сложности и дублирования кода.
- Реализация протоколов, например, протоколов передачи данных по сети.
- Автоматизация процессов, например, рутинных задач, таких как обработка данных.

Тестировщики тоже могут использовать конечные автоматы в разных областях работы:

- Проектирование тестовых случаев для проверки правильной работы состояний в программных приложениях.
- Моделирование ожидаемого поведения программных систем и использование этой информации для написания более эффективных тестов.
- Тестирование производительности и масштабируемости программных систем, использующих состояния.
- Отладка проблем, связанных с переходами состояний в программных системах.

Моделируя программные системы как конечные автоматы, тестировщики могут глубже понять поведение системы и создать более эффективные тесты.

Типы конечных автоматов

Существуют разные типы конечных автоматов. На лекции мы опишем некоторые из них:

- детерминированные конечные автоматы,
- недетерминированные конечные автоматы,
- автоматы с памятью
- машина Тьюринга.

Рассмотрим примеры использования этих типов и поговорим о том, как выбрать подходящий для вашей задачи. После вы сможете лучше понять, как использовать конечные автоматы для решения разных задач при проектировании систем.

Детерминированные конечные автоматы

Детерминированный конечный автомат (DFA) — это конечный автомат, у которого есть единственный переход из каждого состояния при данном входном символе. Поведение автомата предсказуемо, его легко описать с помощью таблицы переходов.

Простой пример — автомат, который определяет, является ли входная строка целым числом. Состояния автомата: «Начало», «Цифра» и «Ошибка». Если символ является цифрой, автомат переходит в состояние «Цифра», если не является — в состояние «Ошибка». Далее мы завершаем выполнение автомата. Если входная строка заканчивается в состоянии «Цифра», то выражение является целым числом, в противном случае — нет.

Таблица переходов для этого примера показывает, какие действия выполняет автомат на каждом этапе обработки строки:

Текущее состояние	Входной символ	Следующее состояние
Начало	Цифра	Цифра
Начало	Не цифра	Ошибка

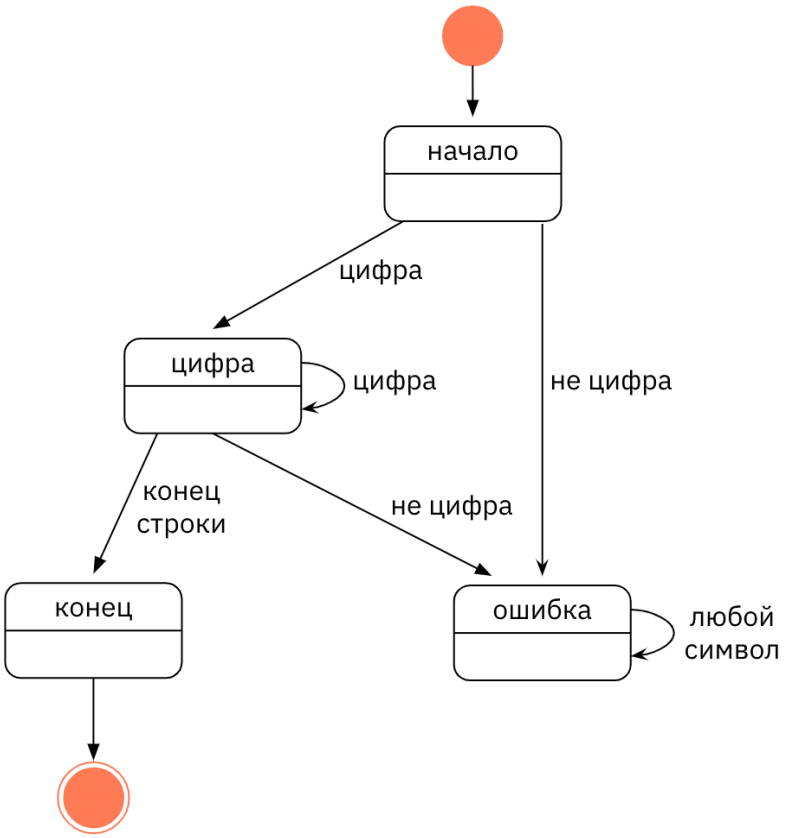
Цифра	Цифра	Цифра
Цифра	Не цифра	Ошибка
Ошибка	Любой символ	Ошибка

Как видно из таблицы, когда автомат находится в состоянии «Начало», он переходит в состояние «Цифра», если входной символ является цифрой, или в состояние «Ошибка», если входной символ не является цифрой.

Аналогично, когда автомат находится в состоянии «Цифра», он переходит в состояние «Цифра», если входной символ является цифрой, или в состояние «Ошибка», если входной символ не является цифрой.

Если автомат завершает выполнение в состоянии «Ошибка», входная строка не является целым числом.

Детерминированный конечный автомат для определения целого числа



Недетерминированные конечные автоматы

Недетерминированный конечный автомат (NFA) — это модель конечного автомата, в которой возможен переход из одного состояния в несколько других одновременно при получении одного и того же символа входной строки. NFA может рассматривать несколько возможных путей одновременно, что делает его более гибким, но более сложным в реализации и анализе.

Пример — автомат, проверяющий, является ли входная строка палиндромом. Состояния: «Начало строки», «Середина строки» и «Конец строки». На каждом шаге автомат сравнивает символ с начала и конца строки. Если символы совпадают, автомат переходит в следующее состояние, если нет, заканчивает работу и выдает «Нет».

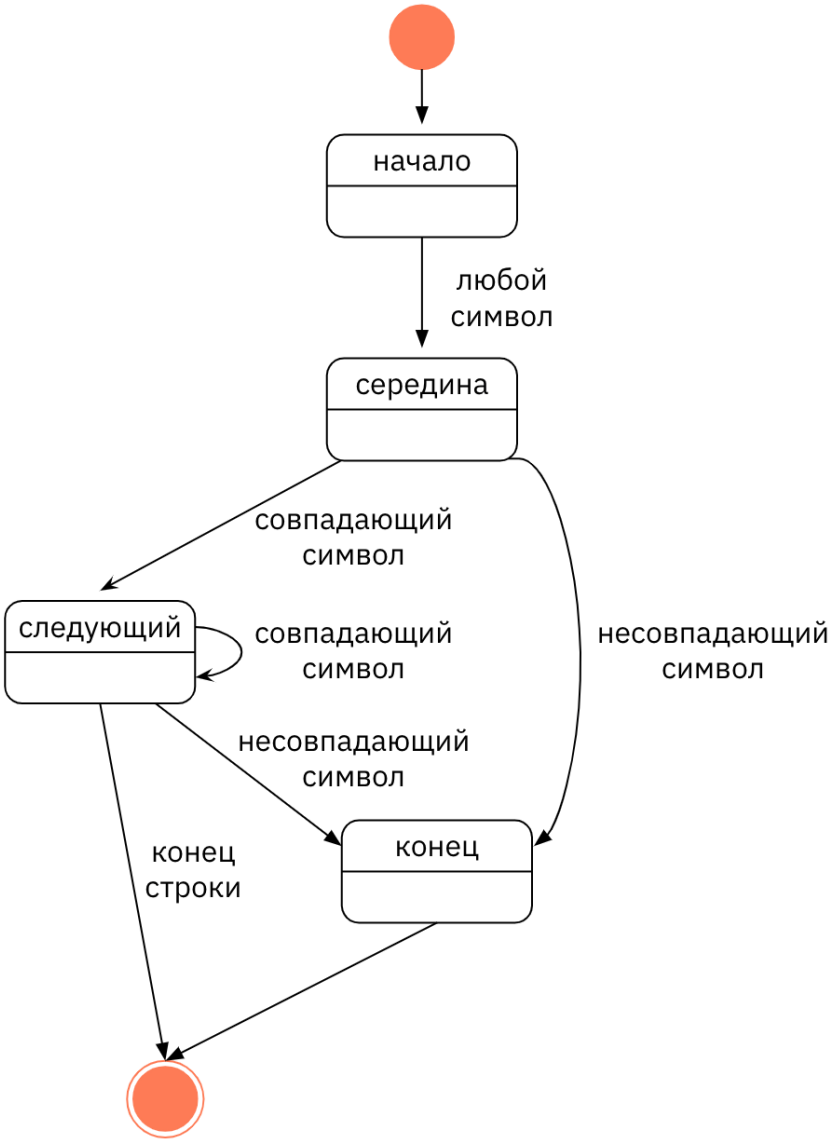
Представим таблицу для начала из середины строки, которая показывает, какие переходы возможны на каждом шаге:

Текущее состояние	Входной символ	Следующее состояние
Начало строки	Любой символ	Середина строки
Середина строки	Совпадающий символ	Следующий символ
Середина строки	Несовпадающий символ	Конец

Как видно из таблицы, когда автомат находится в состоянии «Начало строки», он переходит в состояние «Середина строки» для любого входного символа.

Аналогично, когда автомат находится в состоянии «Середина строки», он переходит в следующее состояние для совпадающего символа или в состояние «Конец строки» для несовпадающего символа.

Недетерминированный конечный автомат палиндрома



Конечные автоматы с памятью

Конечный автомат с памятью (Pushdown Automata, PDA) — это расширенный вариант конечного автомата, у которого есть дополнительный стек для хранения информации. Это позволяет выполнять более сложные операции парсинга, например разбор выражений или вложенных скобок.

Конечный автомат с памятью можно использовать для разработки компиляторов, трансляторов и других систем, которые требуют углубленного анализа структуры текста.

Прежде чем перейдем к примеру, дадим еще одно определение.

Стек — это структура данных, которая реализует принцип LIFO (Last-In-First-Out), то есть последний добавленный элемент выходит первым. Используется в разных

алгоритмах, в том числе в поиске пути в графах и реализации математических выражений.

Пример конечного автомата с памятью — автомат для проверки суммы чека в магазине.

- **Стартовое состояние:** проверка пуста.
- **Текущее состояние:** добавление товара в чек.
- **Заключительное состояние:** проверка завершена, выдача результата (соответствует сумма или нет).

Таблица переходов:

- Если товар добавлен, сумма чека увеличивается.
- Если чек завершен, проверяется сумма с помощью алгоритма вычисления суммы.
- Если сумма соответствует, выдается результат «Соответствует».
- Если сумма не соответствует, выдается результат «Не соответствует».

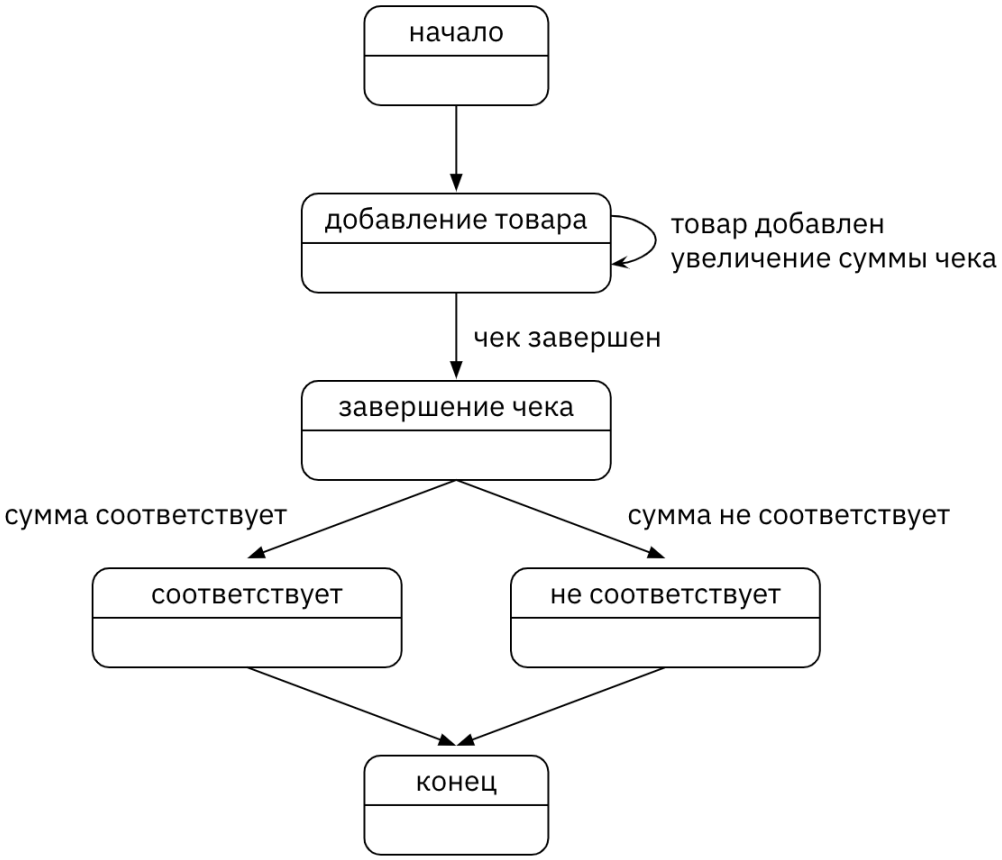
Состояние	Входной сигнал	Действие	Следующее состояние
Начало	-	-	Добавление товара
Добавление товара	Товар добавлен	Увеличение суммы чека	Добавление товара / Завершение чека
Завершение чека	Чек завершен	Проверка суммы чека	Соответствует / Не соответствует
Соответствует	-	-	Конец
Не соответствует	-	-	Конец

В начале программы система находится в состоянии «Начало». При добавлении товара в состоянии «Добавление товара» увеличивается сумма чека. При завершении чека в состоянии «Завершение чека» сумма чека проверяется с помощью алгоритма вычисления суммы. Если сумма соответствует, система

переходит в состояние «Соответствует», иначе — в состояние «Не соответствует». В состоянии «Соответствует» выдается результат «Соответствует», в состоянии «Не соответствует» — результат «Не соответствует». В обоих случаях система переходит в конечное состояние «Конец».

Нарисуем схему диаграммы переходов:

Конечные автоматы с памятью для проверки суммы чека в магазине



Машина Тьюринга

Машина Тьюринга (ТМ) — математическая модель вычислительной системы, предложенная Аланом Тьюрингом в 1936 году. Представляет собой абстрактный компьютер, который может выполнять вычисления, используя два понятия: последовательность команд и память с перезаписываемым содержимым (обычно реализуется в виде стека). Машина Тьюринга считается первым примером модели типа «машина» и является одним из фундаментальных концептов в информатике.

Простой пример — программа, которая выполняет сложение двух чисел. Машина Тьюринга начинает с одной точки и движется в зависимости от входных данных и правил. Например, если входные данные являются двумя числами, машина Тьюринга может выполнить следующие действия:

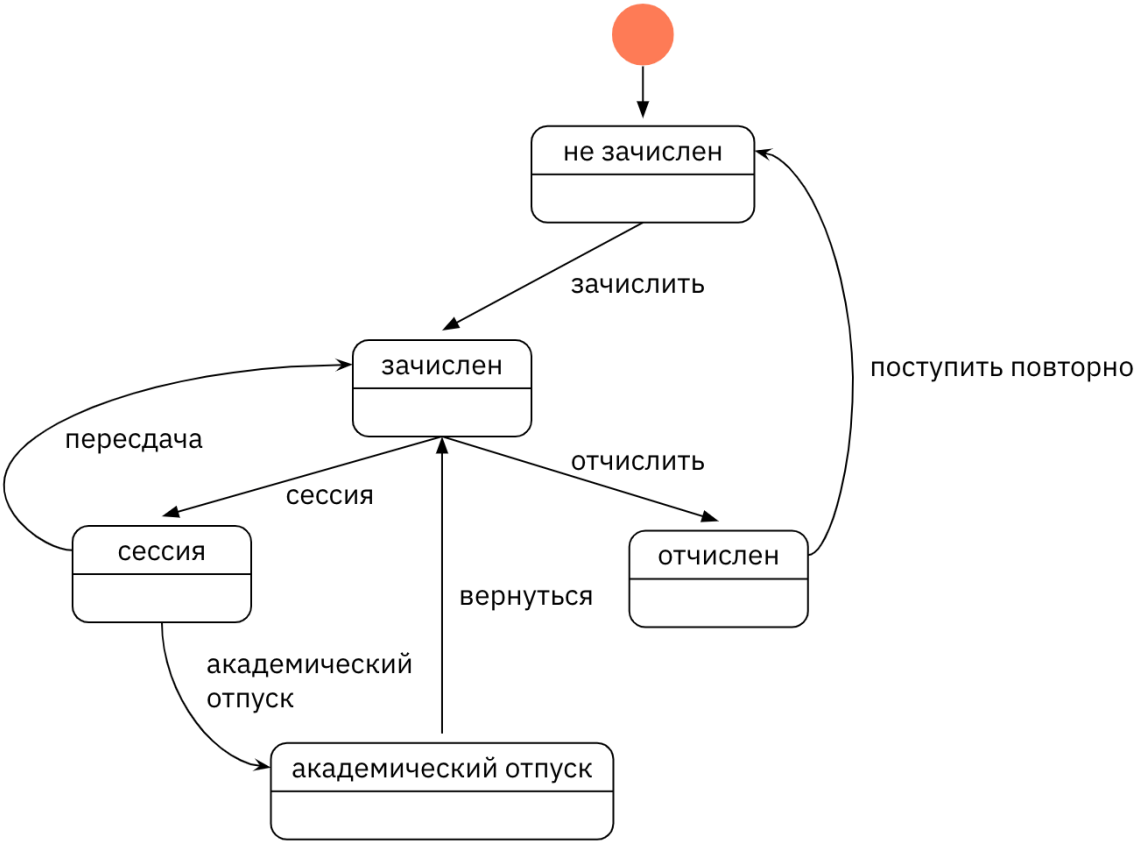
1. Прочитать первое число.
2. Поместить его в стек.
3. Прочитать второе число.
4. Поместить его в стек.
5. Извлечь верхний элемент из стека и сложить его со следующим верхним элементом.
6. Результат сложения поместить в стек.
7. Выдать результат из стека.

Это простой пример, но можно расширить его для выполнения более сложных действий.

Существуют и другие типы автоматов, которые позволяют описать более сложные системы. В рамках этого курса мы их не рассматриваем.

Пример. Конечный автомат для описания статуса обучения

Рассмотрим еще один пример использования диаграмм состояния и конечных автоматов. Конечный автомат для описания статуса обучения студента может выглядеть так:



В этом автомате используются следующие состояния:

- **Не зачислен:** начальное состояние, студент не зачислен в учебное заведение.
- **Зачислен:** студент успешно зачислен и начал учебу.
- **Сессия:** студент проходит сессию.
- **Отчислен:** студент отчислен.
- **Академический отпуск:** студент находится в академическом отпуске.

Создадим таблицу переходов для этого конечного автомата. Она будет описывать все переходы между состояниями и события, которые могут их инициировать:

Текущее состояние	Событие	Следующее состояние
Не зачислен	Зачислить	Зачислен
Зачислен	Сессия	Сессия
Сессия	Пересдача	Зачислен
Сессия	Академический отпуск	Академический отпуск
Зачислен	Отчислить	Отчислен
Отчислен	Поступить повторно	Не зачислен
Академический отпуск	Вернуться	Зачислен

Таким образом, таблица переходов позволяет описать все возможные переходы между состояниями и действия, которые могут быть выполнены пользователем, чтобы перейти из одного состояния в другое.

Мы рассмотрели пример использования конечных автоматов в бизнесе. Этот инструмент позволяет формализовать процессы и задачи в компании, что делает их более прозрачными и управляемыми. Конечные автоматы будут полезны в разных отделах компании, например, в производстве, логистике, продажах и маркетинге. Благодаря этому инструменту разные отделы могут лучше понимать друг друга и работать вместе более эффективно.

Подведем итог. На практике знание теории конечных автоматов пригодиться для ряда задач:

- **Моделирование** — для моделирования сложных систем и процессов, таких как управление памятью, процессором или сетью.
- **Разработка ПО** — для реализации алгоритмов и проектирования программного обеспечения.
- **Разработка игр** — для управления игровыми персонажами и их поведением.
- **Оптимизация** — для оптимизации сложных систем и процессов.
- **Тестирование** — для тестирования и отладки программ.

Моделирование жизненного цикла объекта

Моделирование жизненного цикла объекта — важный аспект программирования. Диаграммы состояний помогут специалистам лучше понять, как объекты взаимодействуют и как изменения в состояниях и переходах могут повлиять на поведение программы.

Диаграммы состояний и их реализация с использованием моделирования жизненного цикла помогают упростить моделирование и тестирование программы тестировщикам, менеджерам и аналитикам, позволяют предсказать, как программа будет работать в разных ситуациях.

Примеры задач, в которых диаграммы состояний могут быть полезны: проектирование пользовательского интерфейса, моделирование поведения системы, разработка игр. В целом, диаграммы состояний можно использовать в любом проекте, где нужно описать жизненный цикл объекта и где важно понимать, как этот объект взаимодействует с другими объектами в системе.

Модель CRUD

Контекст диаграммы состояний можно применить к разным аспектам программирования, включая модель CRUD (create, read, update, delete — создать, прочитать, обновить, удалить).

Концепция CRUD используется в диаграммах состояний для описания возможных переходов между состояниями объекта.

Разберем несколько примеров для каждого аспекта модели CRUD:

- **Создание (Create)** — при создании объекта диаграмма состояний может показать, какой процесс используется для создания объекта и какие состояния могут возникнуть в этом процессе. Например, при создании нового пользователя диаграмма состояний может показать, что пользователь находится в состоянии «Создание», пока вводит данные. Затем, после сохранения данных, пользователь может перейти в состояние «Готовность» для использования учетной записи.

- **Чтение (Read)** — при чтении данных из базы данных диаграмма состояний может показать, что объект находится в состоянии «Запрос» при получении данных из базы данных, а затем переходит в состояние «Отображение» для отображения данных на экране. Диаграмма состояний также может показать, как объект может вернуться к состоянию «Запрос» при обновлении данных.

- **Обновление (Update)** — при обновлении данных, диаграмма состояний может показать, что объект находится в состоянии «Редактирование» для изменения данных. Затем, после сохранения изменений, объект может перейти в состояние «Готовность» для использования обновленных данных.

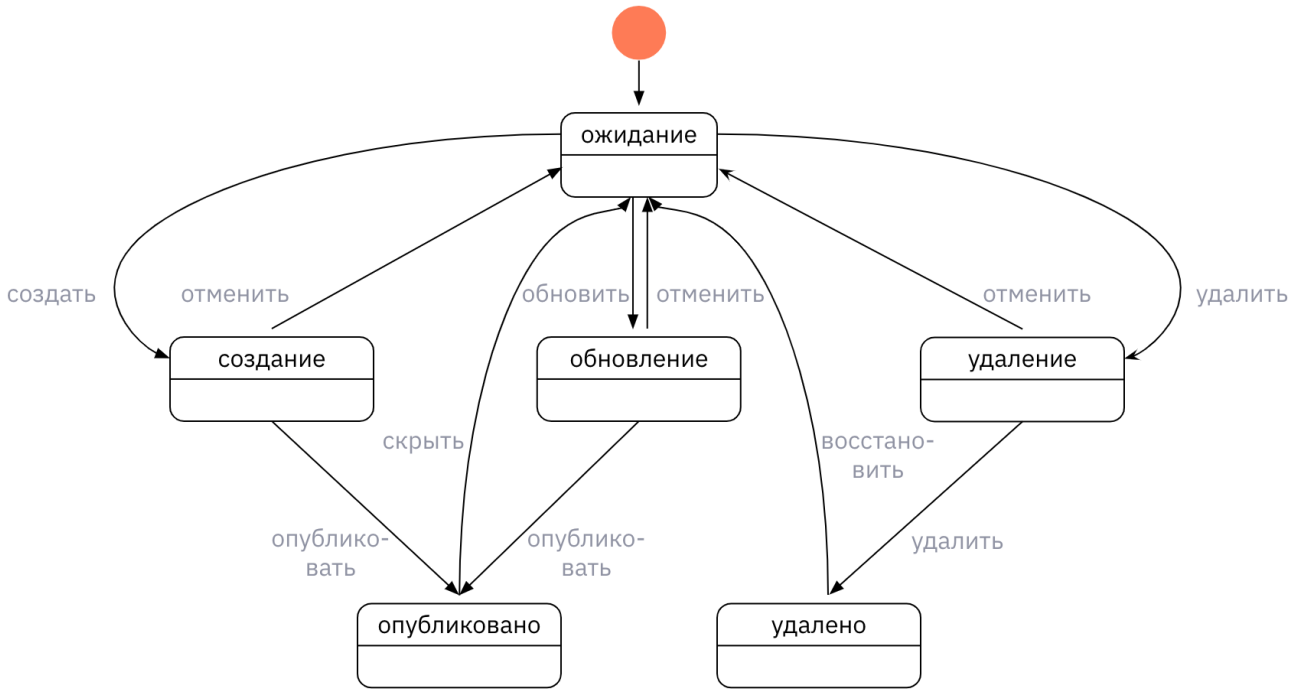
- **Удаление (Delete)** — при удалении объекта диаграмма состояний может показать, что объект находится в состоянии «Удаление», пока он не будет удален из базы данных. Затем, после удаления, объект может перейти в конечное состояние, которое означает, что он больше не существует в приложении.

Таким образом, диаграммы состояний могут помочь специалистам лучше понять и описать процессы создания, чтения, обновления и удаления объектов в приложении, включая модель CRUD. Они также могут помочь в оптимизации и тестировании приложения на каждом этапе жизненного цикла объекта.

Усложнение конечных автоматов

Теперь на примере конечных автоматов разберем все усложнения диаграммы состояний, с которыми сегодня столкнулись. Попробуем добавить к ним концепцию CRUD с исключениями. Представим, что нам нужно реализовать сайт с блогами, и попробуем описать конечный автомат **для управления контентом в своем блоге.**

Конечный автомат, моделирующий процесс управления контентом блога, может выглядеть так:



В этом автомате используются следующие состояния:

- **Ожидание:** начальное состояние, в котором ожидается действие пользователя.
- **Создание:** состояние, в котором пользователь может создать новый контент.
- **Обновление:** состояние, в котором пользователь может обновить существующий контент.
- **Удаление:** состояние, в котором пользователь может удалить существующий контент.
- **Опубликовано:** состояние, в котором контент был успешно опубликован.
- **Удалено:** состояние, в котором контент был успешно удален.

Пользователь может выполнить следующие действия:

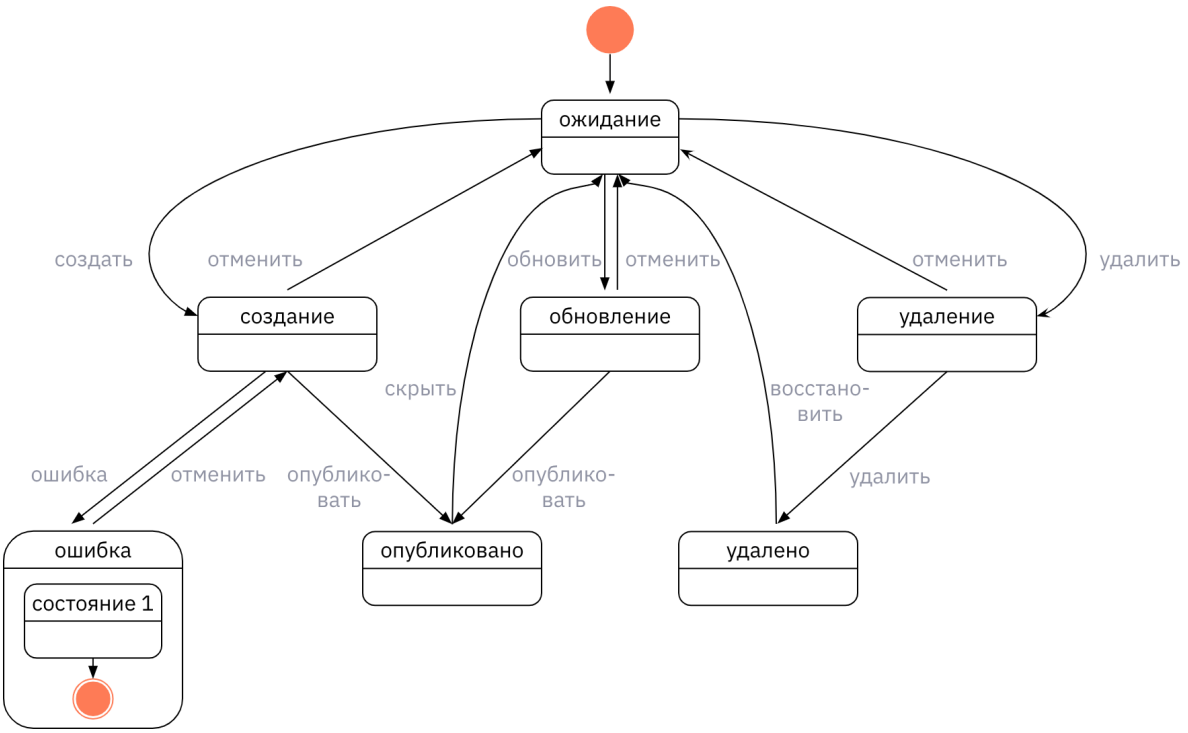
- **Создать:** переход в состояние «Создание», где пользователь может создать новый контент.
- **Удалить:** переход в состояние «Удаление», где пользователь может удалить существующий контент.
- **Обновить:** переход в состояние «Обновление», где пользователь может обновить существующий контент.
- **Отменить:** переход обратно в состояние «Ожидание» и отмена текущего действия.

- **Опубликовать:** переход в состояние «Опубликовано», где контент будет опубликован и доступен для просмотра.
- **Скрыть:** переход обратно в состояние «Ожидание», где контент будет скрыт и недоступен для просмотра.
- **Восстановить:** переход обратно в состояние «Ожидание», где удаленный контент будет восстановлен.

Таким образом, этот конечный автомат позволяет пользователям создавать, обновлять, удалять и управлять контентом блога с помощью набора предопределенных действий и состояний.

Конечный автомат для управления контентом блога можно дополнить обработкой исключительных ситуаций. Например, можно добавить обработку исключения, которое возникает, если пользователь пытается создать контент, который уже существует.

Пример диаграммы состояний с обработкой исключения:



В примере добавлено новое состояние «Ошибка». Оно обрабатывает исключительную ситуацию, когда пользователь пытается создать контент, который уже существует. Если такое исключение возникает, автомат переходит в состояние «Ошибка», где отображается соответствующее сообщение об ошибке. Затем автомат переходит обратно в состояние «Создание», где пользователь может отменить текущее действие.

Таким образом, добавление обработки исключений позволяет сделать конечный автомат более надежным и защищенным от ошибок, которые могут возникнуть в процессе работы с приложением.

Для этой схемы конечного автомата с учетом исключений можно создать таблицу переходов, которая будет описывать все возможные переходы между состояниями, а также условия, необходимые для совершения каждого перехода:

Текущее состояние	**Событие**	**Следующее состояние**	**Условия**
[*]	Начало работы	Ожидание	Нет
Ожидание	Создать	Создание	Нет
Ожидание	Удалить	Удаление	Нет
Ожидание	Обновить	Обновление	Нет
Создание	Отменить	Ожидание	Нет
Создание	Опубликовать	Опубликовано	Нет
Создание	Ошибка	Ошибка	Контент уже существует
Ошибка	Отменить	Создание	Нет
Обновление	Отменить	Ожидание	Нет
Обновление	Опубликовать	Опубликовано	Нет
Удаление	Отменить	Ожидание	Нет
Удаление	Удалить	Удалено	Нет
Опубликовано	Скрыть	Ожидание	Нет

Удалено	Восстановить	Ожидание	Нет

Эта таблица переходов описывает все возможные переходы между состояниями: от начального состояния [*] до конечных состояний «Опубликовано» и «Удалено». В таблице также указаны события, инициирующие переходы и условия, которые должны быть выполнены, чтобы переход состоялся. Если переход невозможен, автомат остается в текущем состоянии, ожидая следующего события.

Таким образом, таблица переходов — это удобный способ описания логики работы конечного автомата, который позволяет программистам легко проверять и управлять переходами между состояниями в процессе разработки приложения.

Домашнее задание

Постройте конечный автомат по работе с состояниями врача в программе для онлайн-записи из бизнес-кейса.

Кейс 1: программа для записи к врачу

Программное обеспечение для онлайн-записи к врачу позволяет пациентам записываться на прием через интернет без необходимости обращаться в медицинское учреждение лично.

Функционал, который может включать программа:

- просмотр доступных врачей и их расписания,
- выбор времени и даты приема,
- подтверждение записи через электронную почту или SMS,
- отмена или изменение записи,
- напоминание о записи,
- предоставление доступа с информацией о медицинской карте пациента и истории записей.

Вы можете дополнять этот кейс недостающей информацией.

Что можно почитать еще?

1. Теория вычислений. Введение в конечные автоматы
2. Конечный автомат: теория и реализация

3. Введение в конечные автоматы