ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Лекция 1

Тема № 1.1

Информация и информационные технологии

«*Технология* – совокупность производственных методов и процессов отрасли производства, а также научное описание способов производства...» *(Ожегов С.И. Толковый словарь русского языка / С.И. Ожегов, Н.Ю. Шведова. – М.: ООО «ИТИ Технологии», 2003.)*

Информация существует в виде различных материальных форм, тогда как сама по себе является общим абстрактным содержанием в различных формах представления.

В различных областях науки существуют свои определения понятия «*информация*».

Формы представления информации – *формы сообщения:*

- *сигналы* (физические величины или свойства физической среды)
- *изображения* (многомерные пространственные сигналы)
- *знаки* (повторяющиеся образцы сигналов)

- анализ и обработка сигналов
- распознавание образов
- информатика – сбор, хранение, поиск, обработка и выдача информации в знаковой форме

Информационные технологии можно понимать как совокупность средств и методов сбора, обработки и передачи данных для получения информации нового качества о состоянии объекта, процесса или явления.

Данные — факты, идеи, сведения в знаковой форме, позволяющей производить их передачу, обработку и интерпретацию

Информационные технологии

Постоянные изменения

Современные информационные технологии

- Динамичное развитие микропроцессорной техники
- Автоматизированная обработка информации
- Развитие средств связи
- Накопление информации на электронных носителях

Классификации информационных технологий:

- ИТ обработки данных
- ИТ управления
- ИТ автоматизации офиса
- ИТ поддержки принятия решений
- ИТ экспертных систем

- функционально-ориентированные ИТ для реализации определенных задач
- предметно-ориентированные ИТ для решения конкретны задач в определенной сфере
- проблемно-ориентированные ИТ для решения типовых прикладных задач

- технологии обработки текстовой информации
- технологии обработки числовой информации
- технологии обработки графической информации
- технологии обработки звуковой информации
- технологии работы в глобальных сетях
- социальные информационные технологии

Этапы информатизации общества

- Изобретение письменности
- Изобретение книгопечатания (середина XVI века)
- Прогресс средств связи (конец XIX века)
- Появление микропроцессорной техники (70-ые гг. XX века)
- Информационное общество

Андрей Петрович Ершов: «О фазах продвижения к информационному обществу следует судить по совокупным пропускным способностям каналов связи»

Информатизация общества влечет за собой отток людей из сферы прямого материального производства в информационную сферу (вторая половина XX века)

Информационная сфера

- деловая информация
  (биржевая, финансовая, статистическая, коммерческая информация);

- профессиональная информация
  (научно-техническая информация, первоисточники и пр.);

- потребительская информация
  (новости, всевозможные расписания, развлекательная информация);

- услуги образования

- другое

информационный кризис

применение информационных технологий

Символ — знак, который имеет специальный смысл.

Исходных знаков для представления информации не достаточно.

Получить из конечного множества знаков неограниченный запас сообщений можно путем составления последовательностей первичных знаков.

Алгебраический подход:

Теория информации — абстрактная теория слов со своими специфическими задачами, связанными с их хранением в памяти компьютера, обработкой и передачей по каналам связи.

Формальная математическая модель — *формальные языки*.

Информатика

Термин «*информатика*» согласно А.П.Ершову вводился в российскую науку 3 раза:

I. [середина XX века] обозначение некоторой дисциплины, занимающейся обработкой научно-технической информации. "*информатика — наука о научно-технической информации*".

I. [1976 г.] после издания перевода книги Ф.Л. Пауэра и Т. Гооза "Введение в информатику", термин начал соответствовать определению «*Informatique* » (фр.) и «*Computer science* »(англ.).

I. [Ершов] Информатика — *фундаментальная естественная наука, изучающая процессы передачи и обработки информации.*

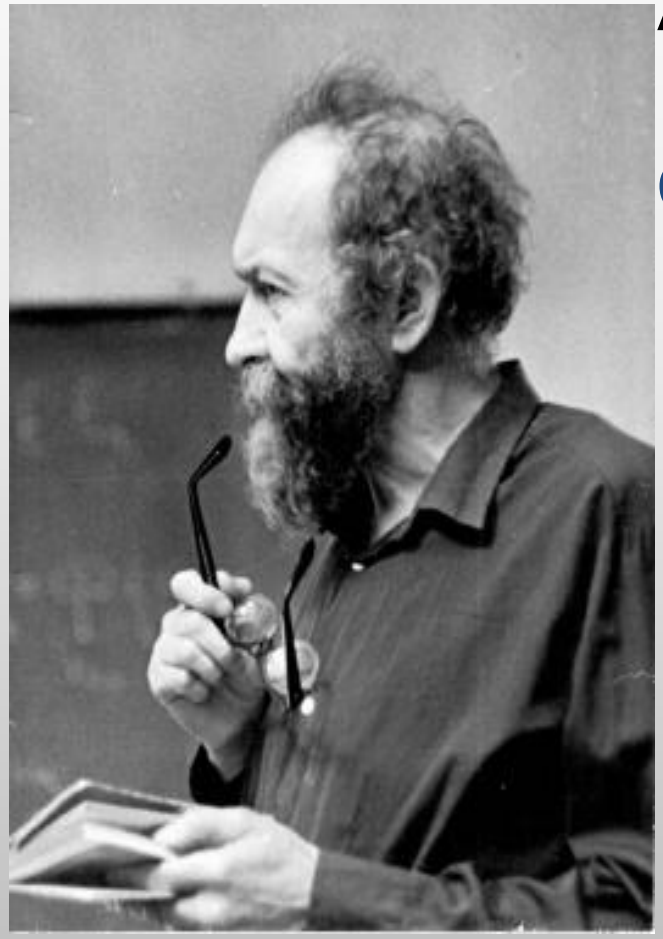
- Информатика завязана на создание *информационной модели*. То есть сама информатика — методология создания информационной модели и методов использования таких моделей.

- Общие принципы принадлежат информатике, в то время как сами модели — результат частных наук, дающих материал и исходные данные для создания моделей.

- Понятие модели в технике связано с "имитационным моделированием", но не ограничено им. В информатике — модели информационные.

«Программирование — это искусство, поскольку оно является приложением накопленных знаний для практических целей, поскольку оно требует умения и мастерства, и в особенности потому, что продукты программирования могут представлять эстетическую ценность»

Дональд Кнут,
Тьюринговская лекция 1974 г.



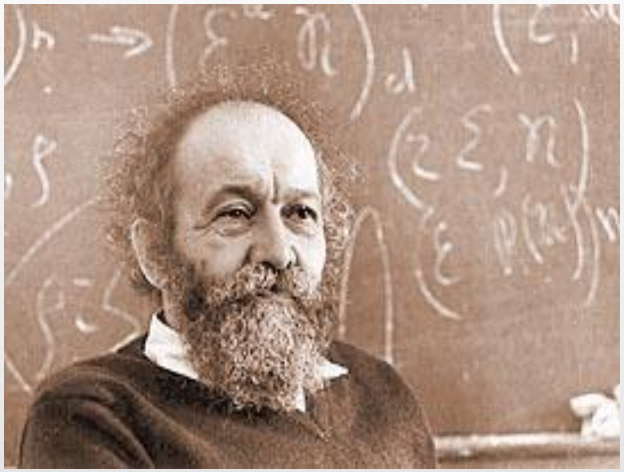
Чл.-корр. АН СССР

Алексей Андреевич Ляпунов (1911-1973)

Основатель советской кибернетики и программирования

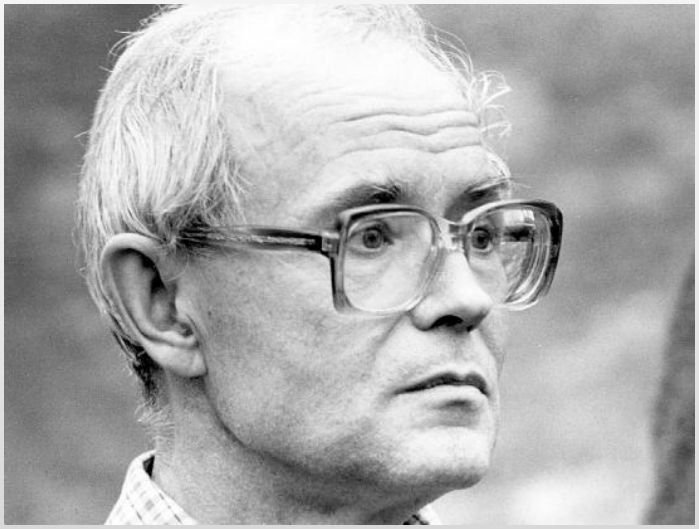
- общие вопросы кибернетики
- математические основы программирования
- теория алгоритмов
- математическая лингвистика
- 1954 г. МГУ Большой семинар по кибернетике

Алексей Андреевич Ляпунов



В начале 50-х годов создает основы программирования на ЭВМ: операторный метод — язык программирования

Алгебраическая теория программирования
Автоматическое программирование:
«программирующие программы» — трансляторы
Механико-математический факультет МГУ:
1953 – семинар по программированию
1954 – первый выпуск по специальности «программирование»



Андрей Петрович ЕРШОВ
(1931-1988)

Андрей Петрович ЕРШОВ (1931-1988)

1958 - первая в мировой практике монография по трансляции:

Программирующая программа для быстродействующей электронной счетной машины. – М.: Изд. АН СССР, 1958. - 116с.
Programming programme for the BESM computer. –London a.o.: Pergamon Press, 1959. – 158p.

Первый оптимизирующий транслятор Альфа:

Многопроходная система трансляции. Оптимизирующие преобразования промежуточного представления программ. АЛЬФА-6 (1973), АИСТ, проект БЕТА: внутренний язык. Язык СИГМА – символьная обработка, генерация программ

Проблема IT-Индустрии:

«индустриализация»

труда программиста

Противоречие между традиционно высоким уровнем фундаментального образования в Российской высшей школе и недостаточным уровнем базового образования на программистских специальностях: подмена базового образования «тренингом»

Проблема индустриализации программирования (взаимоотношения с «промышленностью»)

«Конвейерный метод в программировании может либо убить интеллектуальную компоненту в труде программиста, либо вызвать неврозы…»

А.П.Ершов, 1972 г.

«Программирование – это слишком сложное интеллектуальное занятие, чтобы можно было надеяться навязать ему узы иерархической системы, которая душит всякую инициативу»

Б.Мейер, К.Бодуэн, 1982 (1978) г.

«Программист должен обладать способностью первоклассного математика к абстракции и логическому мышлению в сочетании с эдисоновским талантом сооружать все, что угодно, из нуля и единицы»

А.П.Ершов

Выполнение программы

Программа — последовательность команд, которые заставляют выполнять указанные в них действия с содержимым указанных ячеек памяти.

Программа на алгоритмическом языке также состоит из последовательности команд, но записанных без явного описания ячеек памяти.

Эквивалентные программы — программы, решающие одну и ту же задачу и дающие на выходе одинаковые результаты при одинаковых исходных данных.

Компиляция (трансляция) /compilation (translation)/ — перевод программы с одного языка на другой язык:

Преобразование текста с одного языка в семантически эквивалентный текст на другом языке.

Компилятор — языковой процессор (программа), который воспринимает программу на некотором входном языке в качестве входных данных, а на выходе выдает эквивалентную программу на другом языке.

Задача – получить ***эквивалентную программу***

Работа компилятора

Компилятор предназначен для преобразования программы, написанной на алгоритмическом языке, в эквивалентную программу на машинном языке

Анализирующая часть компилятора разбивает исходную программу на составляющие ее элементы (конструкции языка) и создает промежуточное представление исходной программы (выделяет более «крупные» единицы для последующего разбора)

Синтезирующая часть компилятора, в соответствии с некоторой грамматикой, разбивает промежуточное представление программы и создает программу в машинных кодах

Этапы компиляции:

I. Анализ исходного кода

Лексический анализ — символы группируются в лексические единицы (идентификаторы, служебные слова, операторы, знаки препинания)

Синтаксический анализ — составление из лексических единиц иерархических структур, результат — деревья синтаксического анализа

Семантический анализ — генерация промежуточного кода — программа на языке абстрактного синтаксиса

Этапы компиляции:

II. Синтез выходного кода

Оптимизация кода — преобразование последовательности команд с целью уменьшения их количества, выделение инвариантов, исключение повторяющихся фрагментов и т.д.

Генератор кода — формирование текста программы на новом языке.

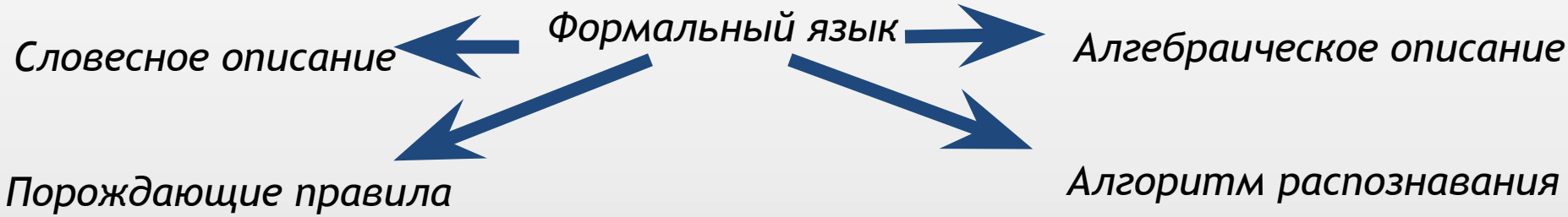
Тема № 1.2

Формальные языки и формальные грамматики

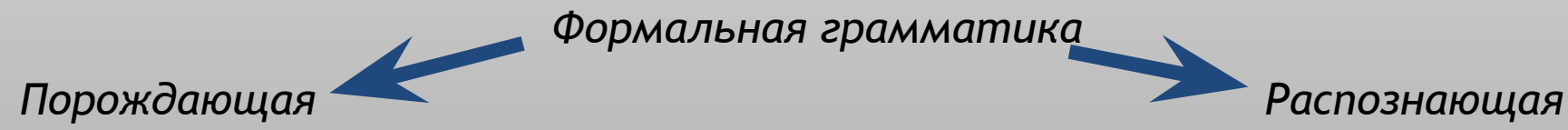
Основные определения

- Алфавит — конечное непустое множество знаков Σ
- Слово (или цепочка) над алфавитом — конечная упорядоченная последовательность элементов множества Σ
- Пустая цепочка ε в алфавит не входит!
- Для цепочек α и β определена операция конкатенции $\alpha \bullet \beta = \alpha\beta$
- Для любой цепочки α выполняется $\alpha \bullet \varepsilon = \varepsilon \bullet \alpha = \alpha$
- Конкатенция – ассоциативная операция $\alpha \bullet (\beta \bullet \gamma) = (\alpha \bullet \beta) \bullet \gamma = \alpha \bullet \beta \bullet \gamma$
- Длина цепочки ω обозначается $|\omega|$, полагаем $|\varepsilon| = 0$
- Степень цепочки $\alpha^k = \underbrace{\alpha\alpha \ldots \alpha\alpha}_{k}$ и обращение цепочки α^R
- Степень алфавита множество всех слов соответствующей длины
- Через Σ^* обозначается множество всех возможных слов над алфавитом Σ и пу
- Множество всех возможных непустых слов над алфавитом Σ обозначается как
- Получаем $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

- Формальный язык — это множество конечных слов (строк, цепочек) над конечным алфавитом
- Т.е. формальный язык L — это подмножество Σ*



- Формальная грамматика — это способ описания формального языка (выделения некоторого подмножества из множества всех слов некоторого конечного алфавита)



- Распознать — в результате процедуры специального вида по заданной цепочке определить, принадлежит ли она языку.
- Примеры:
  - *Машина Тьюринга (МТ)*
  - *Линейно ограниченный автомат (МТ с конечной лентой, ограниченной длиной входного слова) - не детерминированный*
  - *Автомат с внешней памятью (имеется дополнительная бесконечная память)*
  - *Конечный автомат*

- Порождать — на основе набора инструкций из начального множества символов построить все цепочки языка

Порождающая грамматика

Порождающая грамматика задает правила, с помощью которых можно построить любое слово языка

$$G = (T, N, P, S)$$

- T — алфавит терминальных символов
- N — алфавит нетерминальных символов
- P — конечное множество правил вывода
- S — начальный символ грамматики ($S \in N$)

Терминал (терминальный символ) – это объект, непосредственно присутствующий в словах языка, соответствующего грамматике и имеющий конкретное, неизменяемое значение *(0, 1, 2, 3, 4, 5, 6, a, b, c и т.д.)*

Нетерминал (нетерминальный символ) – это объект, обозначающий какую-либо сущность языка (ФОРМУЛА, ВЫРАЖЕНИЕ, КОМАНДА и т.д.) и не имеющий конкретного символьного значения

Вывод цепочки

Вывод цепочки — это последовательность строк, состоящих из терминальных и нетерминальных символов, где первая строка состоит из одного начального нетерминального символа, а каждая последующая строка получена из предыдущей путем замены некоторой подстроки по одному (любому) из правил.
Конечная строка полностью состоит из терминальных символов и является словом языка

Понятие выводимости:

Если αYβ последовательный набор символов языка G, а
Y→x («x непосредственно выводится из Y») правило этого языка, то набор символов αxβ непосредственно выводится из набора символов αYβ в языке G
αYβ → αxβ

Выводимость

- Формально P — конечное подмножество декартова произведения $(T \cup N)^+ \times (T \cup N)^*$
- Элемент (α, β) множества P называется правилом вывода и записывается

$\alpha \to \beta$, где α - левая часть (обязательно содержит нетерминал) и β - правая часть

в записи $\alpha \to \beta_1 | \beta_2 | \ldots | \beta_k$

каждое β_i назовем альтернативой правила вывода из цепочки α

Цепочка β непосредственно выводима из цепочки α в грамматике G = (T, N, P, S) если

$\alpha = \sigma_1\gamma\sigma_2$ и $\beta = \sigma_1\delta\sigma_2$, где $\delta, \sigma_1, \sigma_2 \in (T \cup N)^*$ и $\gamma \in (T \cup N)^+$

и правило вывода $\gamma \to \delta$ содержится в P

Обозначение $\alpha \to \beta$

Цепочка β выводима из цепочки α в грамматике G = (T, N, P, S) если существуют цепочки $\alpha = \gamma_0 \to \gamma_1 \to \ldots \to \gamma_n = \beta$

Обозначение $\alpha \Rightarrow \beta$

Последовательность $\gamma_0, \gamma_1, \ldots, \gamma_n$ называется выводом длины n

Языком L(G), порождаемым грамматикой G = (T, N, P, S) называется множество всех цепочек в алфавите T которые выводимы из S с помощью правил P $L(G) = \{\alpha \in T^* \mid S \Rightarrow \alpha\}$

Примеры

$G_{example} = \langle \{0, 1\}, \{A, S\}, P, S \rangle$, где P состоит из правил:

$$\begin{aligned} S &\to 0A1 \\ 0A &\to 00A1 \\ A &\to \varepsilon \end{aligned}$$

цепочка $00A11$ непосредственно выводима из $0A1$ в грамматике $G_{example}$

$S \Rightarrow 000A111$ в грамматике $G_{example}$ так как существует вывод: $S \to 0A1 \to 00A11 \to 000A111$

$$L(G_{example}) = \{0^n 1^n \mid n > 0\}$$

- Грамматики G_1 и G_2 называются эквивалентными, если $L(G_1) = L(G_2)$

$G_1 = \langle \{0,1\}, \{A,S\}, P_1, S \rangle$

P_1:

$$\begin{aligned} S &\to 0A1 \\ 0A &\to 00A1 \\ A &\to \varepsilon \end{aligned}$$

$G_2 = \langle \{0,1\}, \{S\}, P_2, S \rangle$

P_2:

$$S \to 0S1 \mid 01$$

$$L(G_1) = L(G_2) = \{0^n 1^n \mid n > 0\}$$

- Грамматики G_1 и G_2 называются почти эквивалентными, если языки $L(G_1)$ и $L(G_2)$ отличаются не более чем $L(G_1) \cup \{\varepsilon\} = L(G_2) \cup \{\varepsilon\}$

Вывод цепочки

Построить вывод цепочки для грамматики с правилами

T = {a, b, +}

N = {S, T}

G = (a+b+a)

P: S → T | T+S

T → a | b

1 способ

$$S \rightarrow T+S \rightarrow \boldsymbol{a}+S \rightarrow a+\boldsymbol{T+S} \rightarrow a+\boldsymbol{b}+S \rightarrow a+b+\boldsymbol{T} \rightarrow \underline{a+b+a}$$

2 способ

$$S \rightarrow T+S \rightarrow T+\boldsymbol{T+S} \rightarrow T+T+\boldsymbol{T} \rightarrow T+T+\boldsymbol{a} \rightarrow T+\boldsymbol{b}+a \rightarrow \underline{a+b+a}$$

3 способ

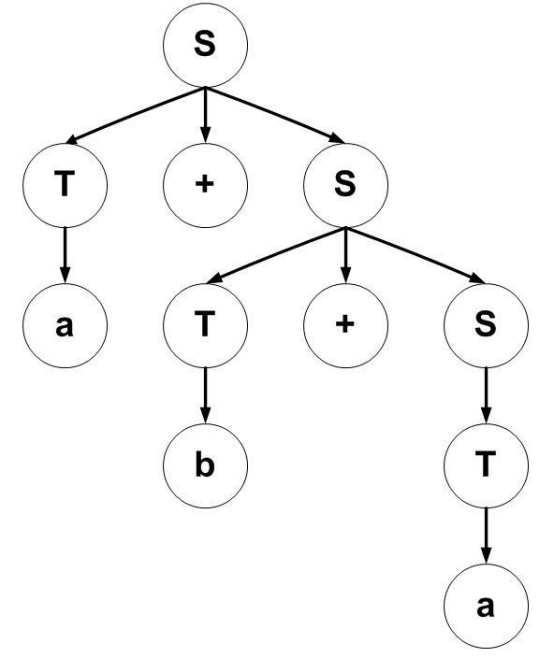
$$S \rightarrow T+S \rightarrow T+\boldsymbol{T+S} \rightarrow T+T+\boldsymbol{T} \rightarrow \boldsymbol{a}+T+T \rightarrow a+\boldsymbol{b}+T \rightarrow \underline{a+b+a}$$

Дерево вывода – это способ представления множества выводов одной и той же цепочки, различающихся лишь порядком применения правил

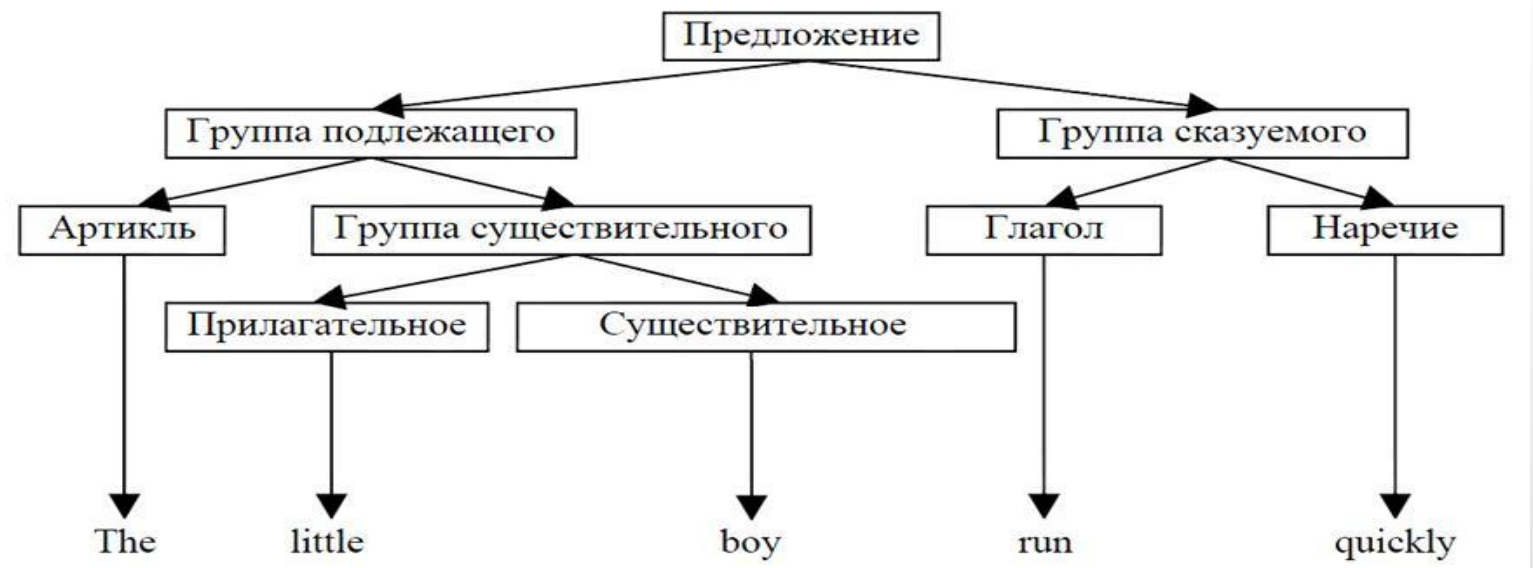
$$S \rightarrow T+S \rightarrow \mathbf{a}+S \rightarrow a+\mathbf{T+S} \rightarrow a+\mathbf{b}+S \rightarrow a+b+\mathbf{T} \rightarrow \underline{a+b+a}$$

$$S \rightarrow T+S \rightarrow T+\mathbf{T+S} \rightarrow T+T+\mathbf{T} \rightarrow T+T+\mathbf{a} \rightarrow T+\mathbf{b}+a \rightarrow \underline{a+b+a}$$

$$S \rightarrow T+S \rightarrow T+\mathbf{T+S} \rightarrow T+T+\mathbf{T} \rightarrow \mathbf{a}+T+T \rightarrow a+\mathbf{b}+T \rightarrow \underline{a+b+a}$$



Пример: The little boy runs quickly



Грамматический разбор предложений:

<предложение> → <группа подлежащего> <группа сказуемого>
<группа подлежащего> → <артикль> <группа существительного>
<группа существительного> → <прилагательное> <существительное>
<группа сказуемого> → <глагол> <наречие>
<артикль> → The
<прилагательное> → little
<существительное> → boy
<глагол> → runs
<наречие> → quickly

Терминальный алфавит: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)}

Нетерминальный алфавит: {ФОРМУЛА, ЗНАК, ЧИСЛО, ЦИФРА}

Правила:

1. ФОРМУЛА → ФОРМУЛА ЗНАК ФОРМУЛА *(формула есть две формулы, соединенные знаком)*

2. ФОРМУЛА → ЧИСЛО *(формула есть число)*

3. ФОРМУЛА → (ФОРМУЛА) *(формула есть формула в скобках)*

4. ЗНАК → + | - | * | / *(знак есть плюс или минус или умножить или разделить)*

5. ЧИСЛО → ЦИФРА *(число есть цифра)*

6. ЧИСЛО → ЧИСЛО ЦИФРА *(число есть число и цифра)*

7. ЦИФРА → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 *(цифра есть 0 или 1 или … 9)*

Начальный нетерминал: ФОРМУЛА

Результат: (12 + 5)

ФОРМУЛА → **3** (ФОРМУЛА)

(ФОРМУЛА) → **1** (ФОРМУЛА ЗНАК ФОРМУЛА)

(ФОРМУЛА ЗНАК ФОРМУЛА) → **4** (ФОРМУЛА + ФОРМУЛА)

(ФОРМУЛА + ФОРМУЛА) → **2** (ФОРМУЛА + ЧИСЛО)

(ФОРМУЛА + ЧИСЛО) → **5** (ФОРМУЛА + ЦИФРА)

(ФОРМУЛА + ЦИФРА) → **7** (ФОРМУЛА + 5)

(ФОРМУЛА + 5) → **2** (ЧИСЛО + 5)

(ЧИСЛО + 5) → **6** (ЧИСЛО ЦИФРА + 5)

(ЧИСЛО ЦИФРА + 5) → **5** (ЦИФРА ЦИФРА + 5)

Классификация грамматик по Хомскому

- Рассматривается четыре типа грамматик, называемых:
  - тип 0
  - тип 1
  - тип 2
  - тип 3

- Тип 0 — неограниченная грамматика

любая порождающая грамматика является грамматикой типа 0

- Тип 1 — контекстно-зависимая грамматика (КЗ)

каждое правило имеет вид: $\gamma_1 A \gamma_2 \rightarrow \gamma_1 \beta \gamma_2$, где γ_i из Σ^* , β из Σ^+ , A из N

- Тип 1 — неукорачивающая грамматика

если правая часть каждого правила вывода не короче левой части

- Если L – формальный язык, то следующие утверждения эквивалентны:
  - Существует контекстно-зависимая грамматика G_1: $L = L(G_1)$
  - Существует неукорачивающая грамматика G_2: $L = L(G_2)$

Классификация грамматик по Хомскому

- Тип 2 — контекстно-свободная грамматика (КС)

каждое правило имеет вид: $A \to \beta$, где β из Σ^* , A из N

- Для любой контекстно-свободной грамматики G существует неукорачивающая грамматика

$G`: L(G) = L(G`)$

- Тип 3 — регулярная (праволинейная, леволинейная) грамматика

праволинейная: каждое правило из P имеет вид $A \to wB$ или $A \to w$, где A и B из N, w из T^*

леволинейная: каждое правило из P имеет вид $A \to Bw$ или $A \to w$, где A и B из N, w из T^*

- Если L — формальный язык, то следующие утверждения эквивалентны:
  - Существует праволинейная грамматика G_1: $L = L(G_1)$
  - Существует леволинейная грамматика G_2: $L = L(G_2)$

т.е. праволинейные и леволинейные грамматики определяют один и тот же класс языков, такие языки называются регулярными

Иерархия Хомского для грамматик

- Любая регулярная грамматика является КС-грамматикой
- Любая неукорачивающая КС-грамматика является КЗ-грамматико
- Любая неукорачивающая грамматика является грамматикой типа

$$\text{неукорачивающие Регулярные} \subset \text{неукорачивающие КС} \subset \text{КЗ} \subset \text{Тип } 0$$

Иерархия Хомского для классов языков

- каждый регулярный язык является КС-языком, но существуют КС-языки, которые не являются регулярными, например:

$$L = \{a^n b^n \mid n > 0\};$$

- каждый КС-язык является КЗ-языком, но существуют КЗ-языки, которые не являются КС-языками, например:

$$L = \{a^n b^n c^n \mid n > 0\};$$

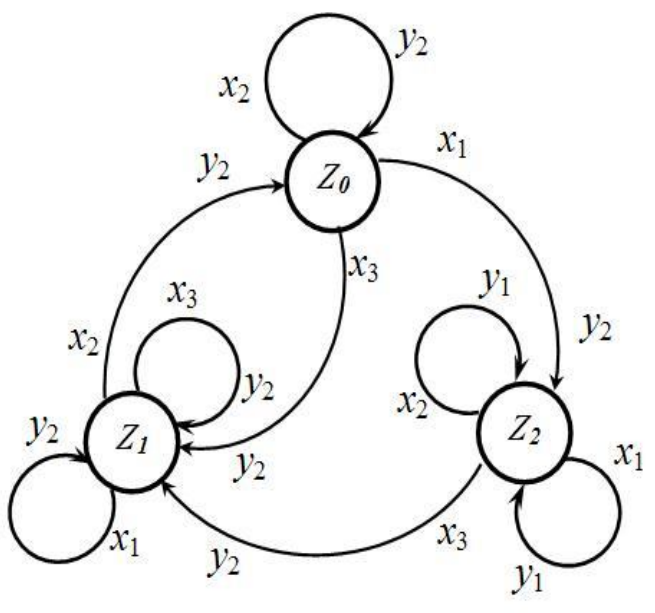
- каждый КЗ-язык является языком типа 0 (т. е. рекурсивно перечислимым языком), но существуют языки типа 0, которые не являются КЗ-языками, например: язык, состоящий из записей самоприменимых алгоритмов Маркова в некотором алфавите. [6)]

иерархия классов языков:

Тип 3 (Регулярные) $\subset$ *Тип 2 (КС)* $\subset$ *Тип 1 (КЗ)* $\subset$ *Тип 0*

Распознающая грамматика

Распознающая грамматика позволяет по данному слову определить, входит оно в язык или нет



x_i	z_k			
	z_0	z_1	…	z_k
Переходы				
x_1	$\varphi(z_0, x_1)$	$\varphi(z_1, x_1)$	…	$\varphi(z_k, x_1)$
x_2	$\varphi(z_0, x_2)$	$\varphi(z_1, x_2)$	…	$\varphi(z_k, x_2)$
…	…	…	…	…
x_I	$\varphi(z_0, x_I)$	$\varphi(z_1, x_I)$		$\varphi(z_k, x_I)$
Выходы				
x_1	$f(z_0, x_1)$	$f(z_1, x_1)$	…	$f(z_k, x_1)$
x_2	$f(z_0, x_2)$	$f(z_1, x_2)$	…	$f(z_k, x_2)$
…	…	…	…	…
x_I	$f(z_0, x_I)$	$f(z_1, x_I)$	…	$f(z_k, x_I)$

Тема № 1.3

Автоматы и деревья

Основные определения

Рассматриваем два алфавита – конечные множества *A* и *B*
И последовательности букв из *A* и *B*:
(a(1), a(2), …, a(t), …) и (b(1), b(2), …, b(t), …)

И отображения $f: A^* \to B^*$

Пусть $A = B = \{0, 1\}$
Рассмотрим *f* такую, что нулевая последовательность переходит в себя, а остальные переводятся в последовательность из одних единиц.

Тогда если *w = (0,0, …)* для определения *f(w)* недостаточно знать конечное число элементов *w*.

Функция *f*: (a(1), a(2), …, a(t), …) → (b(1), b(2), …, b(t), …)
называется детерминированной, если b(t) однозначно определяется первыми t членами.
a(1), a(2), … a(t)

Примеры детерминированных функций:

$$(0,\ldots,0,\underset{t}{1},?,\ldots,?,\ldots) \rightarrow (0,\ldots,0,\underset{t}{1},\ldots,1,\ldots)$$

$$b(t) = a(1) \oplus a(2) \oplus \ldots \oplus a(t)$$

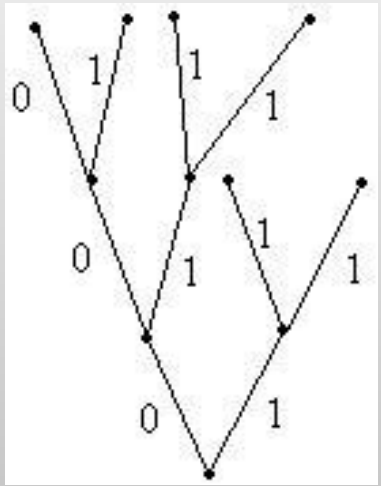
$$(a(1), a(2), a(3)\ldots, a(t), a(t+1)\ldots) \rightarrow (0, a(1), a(2),\ldots, a(t-1), a(t),\ldots)$$

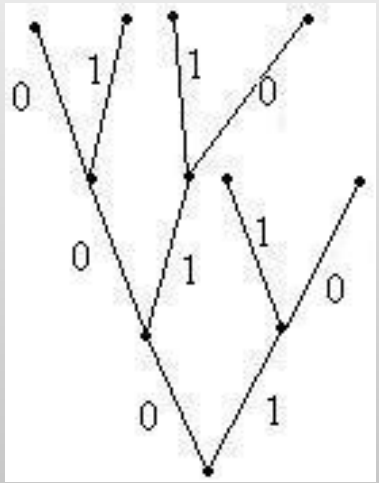
$$b(t) = \begin{cases} 1,\ t = 2^m \\ 0,\ t \neq 2^m \end{cases}$$

Деревья

Детерминированные функции можно задавать на бесконечных деревьях:

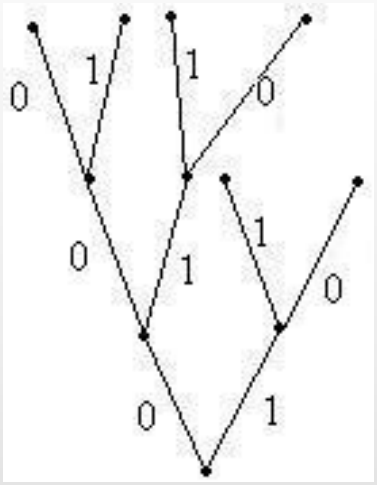
На ребрах записаны элементы образа при условии, что
0 элементу прообраза соответствует движение влево,
1 элементу прообраза соответствует движение вправо.





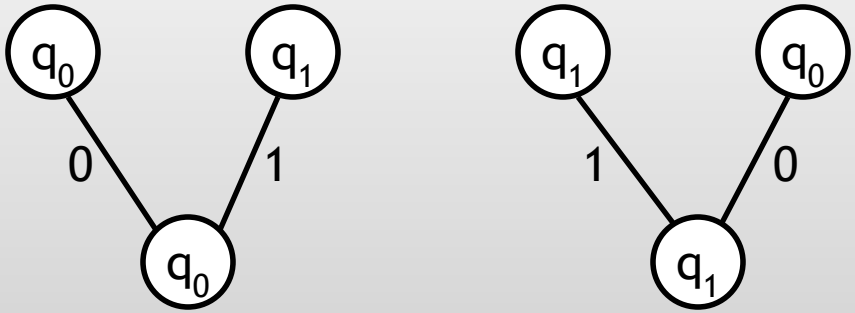
Конечно детерминированные функции или автоматы – детерминированные функци
в деревьях которых содержится лишь конечное число различных *(не изоморфных
собой)* поддеревьев. Изоморфизм здесь подразумевается как биекция, сохраняющ
записанные на ребрах буквы.

От деревьев к диаграммам переходов



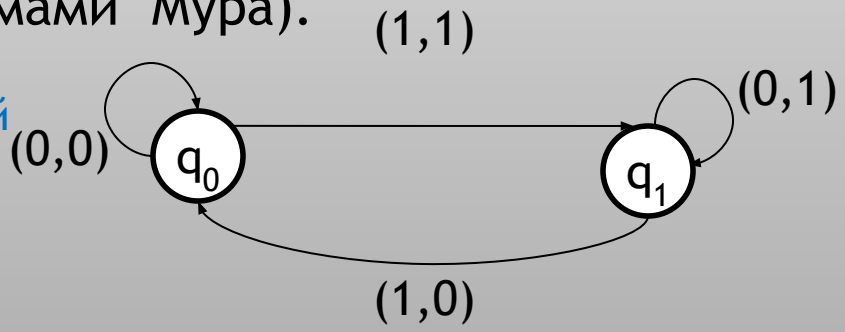
Введем нумерацию поддеревьев.
Для функции четности: $b(t) = a(1) \oplus \ldots \oplus a(t)$

Тут всего две разные картины возможны
(в вершинах стоит нумерация поддеревьев):



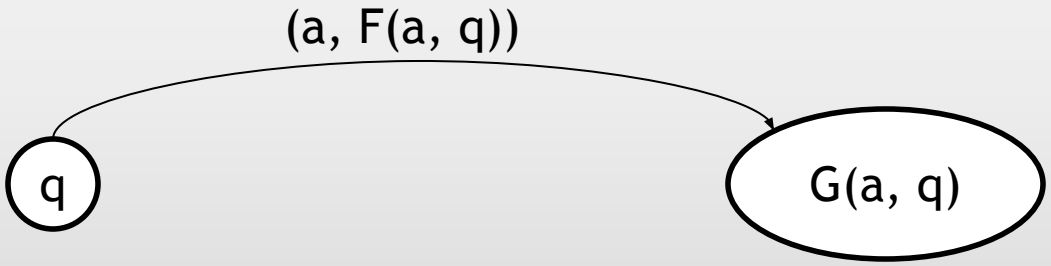
В общем случае для детерминированных функций достаточно знать конечное число конечных фрагментов дерева, чтобы найти образ любой последовательности букв исходного алфавита. Эти части могут задаваться диаграммами переходов (диаграммами Мура).

Каждому ребру приписывается пара символов, первая компонента которой соответствует направлению движения, а вторая - элементу алфавита, приписанному ребру, по которому происходит движение.



Функции перехода и выхода

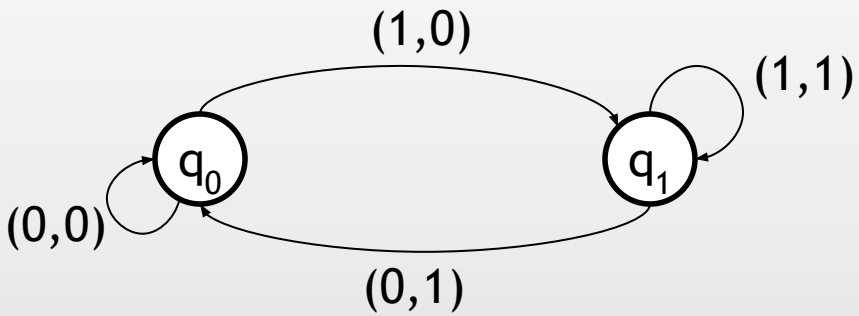
В каждой паре первая компонента – это какая-нибудь буква из A, а вторая – буква из B, которая получается, если в состоянии (оно же номер поддерева), записанном в вершине, из которой выходит ребро, поддать эту букву из A:



Автоматные функции можно задавать двумя функциями:
функцией перехода $G(a(t), q(t)) = q(t+1)$
функцией выхода $F(a(t), q(t)) = b(t)$
$q(t)$ – состояние в момент t
$a(t)$ – очередная буква, подданная на вход,
$b(t)$ – очередная буква на выходе.

Удобно по умолчанию считать $q(0) = 0$, когда это имеет смысл.
Для функции четности $G(x,z) = x \oplus z \quad F(x,z) = x \oplus z$

Найдем функции перехода и выхода для «функции единичной задержки»:
b(0) = 0, b(t) = a(t-1), t > 1



функция перехода G(a(t), q(t)) = a(t)
функция выхода F(a(t), q(t)) = q(t)

q(t+1) = a(t)
b(t) = q(t)
q(0) = 0

Задача №1

Построить вывод цепочки для грамматики с правилами:

$T = \{a, b, c\}$ $N = \{B, C, S\}$ $G = (aaabbbccc)$

P: $S \rightarrow aSBC \mid abC$

$CB \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

Задача №2

Построить вывод цепочки для грамматики с правилами:

$T = \{a, b, c\}$ $N = \{A,B,S\}$ $G = (aabcaab)$

P: $S \rightarrow aAS \mid bBS \mid c$

$Aa \rightarrow aA$

$Ab \rightarrow bA$

$Ac \rightarrow ca$

$Ba \rightarrow aB$

$Bb \rightarrow bB$

$Bc \rightarrow cb$

Задача №3

Построить вывод цепочки для грамматики с правилами:

$T = \{a, b, c\}$ $N = \{A, B, S\}$ $G = (aabbab)$

P: $S \rightarrow aB \mid bA \mid \varepsilon$

$B \rightarrow bS \mid aBB$

$A \rightarrow aS \mid bAA$

Задача №4

Построить вывод цепочки для грамматики с правилами:

$T = \{a, b\}$ $N = \{T, F, S\}$ $G = (a\text{-}b*a+b)$

P: $S \rightarrow T \mid T+S \mid T\text{-}S$

$T \rightarrow F \mid F*T$

$F \rightarrow a \mid b$

Задача №5

Построить вывод цепочки для грамматики с правилами:

$T = \{a, b, c\}$ $N = \{A, B, C, S\}$ $G = (aaaabccc)$

P: $S \rightarrow aA$

$A \rightarrow aA$

$A \rightarrow bC$

$C \rightarrow cC$

$C \rightarrow c$