

Содержание

1 Языки и их свойства, операции над языками	1
1.1 Введение	1
1.2 Операции над языками	2
1.2.1 Операции над словами	2
1.2.2 Операции над языками как множествами	3
1.2.3 Операции над языками как множествами, содержащими последовательности	3
2 Конечные автоматы	3
2.1 Сведение НКА к ДКА	5
2.2 Минимизация ДКА	6
3 Регулярные выражения и языки	8
3.1 Регулярные выражения	8
3.2 Регулярные языки	10
3.2.1 Свойства замкнутости регулярных языков	10
3.2.2 Проверка на нерегулярность	10
3.3 Регулярные выражения на практике	10
4 Лексический анализ	10
5 КС-грамматики и языки	11
5.1 Необходимые определения из близких областей	11
5.1.1 Графы	11
5.1.2 Языки и грамматики	12
5.1.3 Матрицы	12

Аннотация

Это вводный абзац в начале документа.

1 Языки и их свойства, операции над языками

1.1 Введение

Назовём множество абстрактных объектов – символов – алфавитом Σ . Пусть алфавит конечный. Пустой и бесконечный алфавиты нам неинтересны.

Введём слово над алфавитом Σ : $w(A) = a_i, a_i \in \Sigma, \forall i = 0..|w(A)|$ – последовательность (строка) символов из алфавита, $0 \leq |w(\Sigma)| < +\infty$.

Чтобы оперировать словами длины 0, вводят специальный символ длины $0 - \varepsilon : |\varepsilon^n| = 0, n = 0.. + \infty$; Его называют пустым.

Обозначим множество таких последовательностей из символов алфавита Σ , включая слово длины 0, как Σ^* . Тогда некоторый язык $L(\Sigma)$ над алфавитом Σ можно задать как подмножество слов над алфавитом: $L(\Sigma) \subset (\Sigma^*)$.

Таким образом, математически мы определили объекты, с которыми будем работать, это последовательности конечной длины и множества.

Теория формальных языков – математический способ конструктивного описания множеств последовательностей (слов) элементов некоторых множеств (алфавитов). Почему конструктивного? Потому что, в принципе, все слова языка можно просто перечислить, если:

1. любое слово – конечной длины.
2. множество слов конечно.
3. нет ограничений на временную сложность алгоритмов, используемых в работе с таким языком.

Нарушения 1) и 2), соответственно, говорят о том, что мы будем перечислять слова бесконечно, 3) это практическая хотелка – нам нужны алгоритмы, которые работают, по крайней мере, за полином небольшой степени и по времени, и по памяти, так как мы хотим работать с относительно мощными языками, и нам важна масштабируемость.

В нашем курсе 1) будет всегда выполняться: считаем, что любое слово языка – конечной длины. Но пусть 2) не выполняется, а 3) нас просят строго соблюсти. Тогда задача конструктивного, то есть 'сжатого' и точного описания множества слов обретает куда более глубокую практическую значимость.

Кроме перечисления, можно предложить еще 2 способа задания языка:

- Формальный распознаватель – все слова языка можно распознать некоторой вычислительной машиной.
- Генератор – все слова языка можно вывести посредством формальной процедуры переписывания строк по системе правил. Система математических объектов, позволяющих это сделать, называется формальной грамматикой.

С этими двумя способами теория формальных языков и работает. Мы начнём с первого, в последствии переключимся на второй, а затем синхронно двинемся дальше с обеими способами, усложняя и рассматриваемые методы, подходы и задачи.

1.2 Операции над языками

Начнём с базовых операций над элементами языков – словами.

1.2.1 Операции над словами

Опр. 1.1 Конкатенация – склеивание¹ строк. Если $u = a_1 \dots a_m$ и $v = b_1 \dots b_n$ – две строки, то их конкатенация – это строка $u \cdot v = uv = a_1 \dots a_m b_1 \dots b_n$. Знак \cdot , как правило, опускают.

¹устоявшегося русского термина пока нет, увы

Конкатенация строки сама с собой обозначается как возведение в степень: w^n – n раз повторяемая w . $w^1 = w, w^0 = \varepsilon$, то есть конкатенация играет роль умножения с единицей ε , и превращает язык в свободную группу.

Опр. 1.2 *Взятие префикса*

Опр. 1.3 *Взятие суффикса*

Конечно, существует множество других интересных, широкоиспользуемых либо экзотических операций, вроде инверсии слова, но оставим их за рамками.

1.2.2 Операции над языками как множествами

Объединение, пересечение, вычитание, дополнение – как с обычными множествами ... Нам они понадобятся, в особенности, при проверке свойств принадлежности языка некоторому классу.

1.2.3 Операции над языками как множествами, содержащими последовательности

Опр. 1.4 *Конкатенация языков $L_1(\Sigma_1), L_2(\Sigma_2) \subset (\Sigma_1 \cup \Sigma_2)^*$ – это операция склеивания всех возможных слов языков: $L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$.*

Можно взять не 2, а другое число языков k . Если язык конкатенируют сам с собой, то это обозначают L^k . Для $k < 2$ операцию определяют так: если $k = 0$, то это будет язык $\{\varepsilon\}$, что соответствует определению $x^0 = 1$ для чисел. Если $k = 1$, то это будет сам L . Как видим, конкатенация играет роль умножения².

Опр. 1.5 *Итерация языка $L : L^* = \bigcup_{k=0}^{\infty} L^k$.*

Заметим, что множество слов Σ^* – итерация языка Σ .

2 Конечные автоматы

Конечный автомат – математическая модель вычислителя с конечной памятью.

Опр. 2.1 *Недетерминированный конечный автомат (НКА) – это кортеж $\langle Q, \Sigma, \Delta, q_0, F \rangle$:*

- $Q, |Q| < \infty$ – множество состояний
- Σ – алфавит

²это и правда умножение в некотором полукольце с единицей ε (вопрос: а какая операция – сложение в этом полукольце?)

- $\Delta \subset Q \times \Sigma^* \times Q$ – множество переходов³
- $q_0 \in Q$ – стартовое состояние
- $F \subset Q$ – множество финальных состояний

Существует эквивалентное определение автомата, где вместо Δ задают функцию перехода $\delta : Q \times \Sigma^* \rightarrow 2^Q$; будем пользоваться «более графовым» определением через Δ , хотя функция перехода нам ещё понадобится.

Способ распознавания строки автоматом уже лежит в его определении: представим граф автомата. Вершины – это состояния, рёбра – переходы. Если мы находимся в стартовом состоянии, и нам подадут на вход строку, то нам достаточно брать по символу/слову из Σ^* , смотреть, по каким рёбрам графа мы можем перейти (если ε – перейти можем спонтанно), совершать переход(ы), брать следующий символ/слово из Σ^* , смотреть, куда мы по нему можем перейти из текущего состояния, и так далее. Слово распознано, если мы дошли до какого-либо финального состояния и обработали всё слово. То есть распознавание строки автоматом – суть проверка достижимости по рёбрам его графа из q_0 в одно из состояний в F .

Основным недостатком КА служит то, что мы в каждый момент времени знаем только текущее состояние и в какие мы можем из него перейти. У нас нет данных о том, что происходило ранее, и это накладывает ограничения на выразительность⁴. К примеру, нельзя составить КА, распознающий язык $a^n b^n, \forall n \in [0, +\infty)$, хотя для любого фиксированного множества n – можно (Рис. 1).

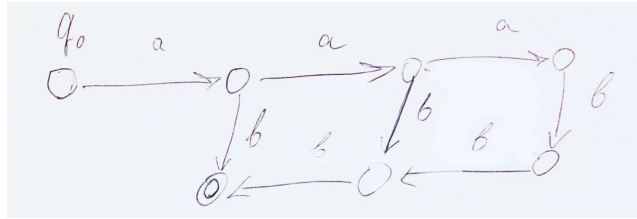


Рис. 1: КА, распознающий язык $a^n b^n, n \in [1, 3]$

О достижимости проще говорить в терминах пар $\langle q_x, v \rangle \in Q \times \Sigma^*$, где q_x – текущее состояние, а v – недоразобранная подстрока входной строки. Такая пара называется конфигурацией автомата⁵. Введём отношение достижимости на конфигурациях.

Опр. 2.2 *Достижимость* (\vdash) – наименьшее рефлексивное транзитивное отношение над $Q \times \Sigma^*$, такое что:

³ Δ задаёт множество двухместных отношений на Q , помеченных элементами Σ^* .

⁴ тем не менее, конечные автоматы широко применяются в технике вокруг нас. Примеры: светофор, лифт, кодовый замок, система контроля воздуха в помещении, компьютерная мышь, аудиоплеер, веб-форма и т.д.

⁵ по мере усложнения моделей вычислителей, мы будем добавлять новые параметры в конфигурацию – например, появится параметр, описывающий стек, и т.д.

1. $\forall w \in \Sigma^* : (\langle q_1, w \rangle \rightarrow q_2) \in \Delta \Rightarrow \langle q_1, w \rangle \Rightarrow \langle q_2, \varepsilon \rangle$
2. $\forall u, v \in \Sigma^* : \langle q_1, u \rangle \vdash \langle q_2, \varepsilon \rangle, \langle q_2, v \rangle \vdash \langle q_3, \varepsilon \rangle \Rightarrow \langle q_1, uv \rangle \vdash \langle q_3, \varepsilon \rangle$
3. $\forall u \in \Sigma^* : \langle q_1, u \rangle \vdash \langle q_2, \varepsilon \rangle \Rightarrow \forall v \in \Sigma^* \langle q_1, uv \rangle \vdash \langle q_2, v \rangle$

Теперь несложно задать язык, распознаваемый КА.

Опр. 2.3 Пусть дан $M = \langle Q, \Sigma, \Delta, q_0, F \rangle$. Язык, распознаваемый автоматом M – $L(M) = \{w \in \Sigma^* \mid \exists q \in F : \langle q_0, w \rangle \vdash \langle q, \varepsilon \rangle\}$.

Опр. 2.4 Язык L называется автоматным, если существует КА $M : L = L(M)$. Множество таких языков L образует класс автоматных языков.

На практике гораздо приятнее работать с детерминированным конечным автоматом (ДКА).

Опр. 2.5 (Неформально) НКА $M = \langle Q, \Sigma, \Delta, q_0, F \rangle$ называется детерминированным КА, если

- Все переходы – однобуквенные: $\forall (\langle q_1, w \rangle \rightarrow q_2) \in \Delta : |w| = 1$
- $\forall a \in \Sigma, q \in Q \mid \delta(q, a) \leq 1$, где $\delta(q, a)$ – множество состояний, достижимых из q по символу a . Задание: расписать $\delta(q, w)$ аккуратно через конфигурации.

Иными словами, для любых фиксированных букв, для любого состояния, переход приводит только в одно результирующее состояние.

Можно ввести ДКА-автоматный язык L_{NFA} по аналогии с тем, как вводили $L(M) = L_{DFA}$. Очевидно, что $L_{DFA} \subseteq L_{NFA}$, так как ДКА – это частный случай НКА.

Если мы покажем, что произвольный НКА сводится к ДКА, то $L_{DFA} = L_{NFA}$.

2.1 Сведение НКА к ДКА

Л. 2.1 («Построение подмножеств», Рабин и Скотт [1959]). Пусть $B = (\Sigma, Q, q_0, \Delta, F)$ – произвольный. Тогда $\exists DFA A = (\Sigma, 2^Q, Q_0, \Delta', F')$, состояния которого – множества Q , который распознаёт тот же язык, что и B . Его переход в каждом состоянии-подмножестве $s \subseteq Q$ по каждому символу $a \in \Sigma$ ведёт во множество состояний, достижимых по a из некоторого состояния s .

Произведём серию упрощений НКА.

Утв. 2.1 В определении НКА можно считать все переходы – однобуквенными. Для этого нужно перестроить множества Δ и Q .

Утв. 2.2 В определении НКА можно считать $|F| = 1$.

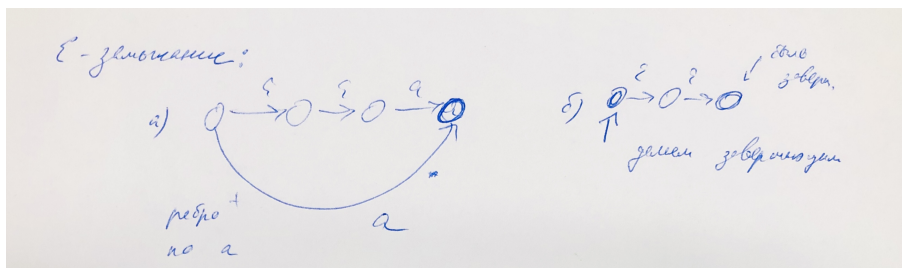


Рис. 2: Основные преобразования при построении ерс-замыкания: последовательность переходов $\epsilon \dots \epsilon a$ заменить на переход a (а), состояние, из которого существует переход $\epsilon \dots \epsilon$ в финальное состояние – обозначить как финальное (б)

Утв. 2.3 (Эпсилон-замыкание) От переходов по ϵ можно избавиться, применив некоторые преобразования (см.Рис. 2).

Эти утверждения доказываются технически, не будем этим заниматься сейчас (рекомендуется попробовать доказать дома или посмотреть в классических книгах и курсах).

TODO: доказательство Л2.1, алгоритм на базе метода «построение подмножеств»

Утв. 2.4 (о корректности Л2.1). Для любой строки $w \in \Sigma^*$, состояние-подмножество, достигаемое DFA по прочтении строки w , содержит элемент q тогда и только тогда, когда хотя бы одно из вычислений NFA на w заканчивается в состоянии q .

Доказывается индукцией по длине строки w .

Далее из утверждения о правильности выводится, что построенный DFA распознаёт строку $w \in \Sigma^*$ тогда и только тогда, когда распознаёт исходный NFA. Построение переводит NFA с n состояниями в DFA с 2^n состояниями-подмножествами. На практике, многие из них обычно бывают недостижимы. Поэтому хороший алгоритм должен строить только подмножества, достижимые из уже построенных, начиная с q_0 .

2.2 Минимизация ДКА

Говорят, что состояния u, v различаются словом s , если одно из них по s переводит автомат в финальное состояние, а другое нет.

Если состояния не различаются никакой строкой, они называются неразличимыми. На Рис.2 изображен ДКА, в котором есть такие: действительно, окажемся мы в финальном состоянии или нет, зависит только от количества нулей в строке, следовательно, В и С – неразличимы.

Л. 2.2 Отношение неразличимости суть отношение эквивалентности.

Рефлексивность очевидна, симметричность следует из определения (попробуйте заменить u и v местами).

Транзитивность: u и v неразличимы, v и w неразличимы, следовательно, u и w неразличимы, тоже очевидно.

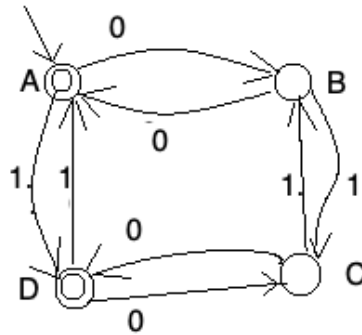


Рис. 3: ДКА, в котором есть неразличимые состояния (найдите их)

По индукции по длине строки доказывается, что модификация автомата как на Рис. 3, если состояния A и B не различимы, не меняет распознаваемый им язык.

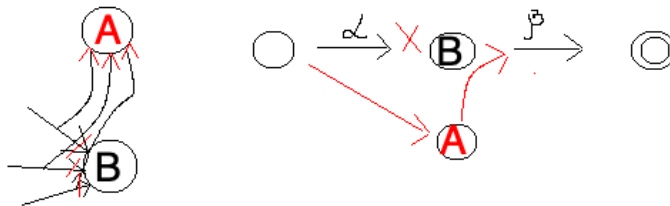


Рис. 4: Вспомогательный рисунок

Повторяя процедуру модификации для всех классов эквивалентности, оставляя какую-то одну вершину для каждого класса, получим некий автомат с возможно меньшим числом состояний. Можно доказать, что это число состояний – минимально.

Л. 2.3 Пусть у ДКА M все состояния различимы и любое достижимо из стартового. Тогда M – минимальный автомат для $L(M)$

Т. 2.1 Для любого ДКА существует и единственный с точностью до изоморфизма ДКА с минимальным числом состояний.

Интуитивно, для выполнения минимизации нужно выделить:

- Недостижимые состояния – их нужно выкинуть⁶
- Неразличимые состояния – их можно объединить в одно для каждого класса эквивалентности

Существует, как минимум, 3 способа выделить и схлопнуть неразличимые состояния:

- Наивный алгоритм основан на построении классов эквивалентности и объединении эквивалентных состояний [1], и рассматривается на семинаре. Он работает за $O(n^2)$.
- Алгоритм Хопкрофта, позволяющий решить задачу за $O(n \log(n))$. [2].
- Также существует алгоритм Бржозовского, который строит минимальный ДКА из НКА [3]

3 Регулярные выражения и языки

3.1 Регулярные выражения

Опр. 3.1 (Клини [1951]). Регулярные выражения над алфавитом Σ определяются так:

- ε — регулярное выражение.
- Всякий символ a , где $a \in \Sigma$ — регулярное выражение.
- Если α, β — регулярные выражения, то тогда $(\alpha|\beta)$, $(\alpha\beta)$ и $(\alpha)^*$ — тоже регулярные выражения.

Всякое регулярное выражение α определяет язык над алфавитом Σ , обозначаемый через $L(\alpha)$.

Всякий символ из Σ обозначает одноэлементное множество, состоящее из односимвольной строки: $L(a) = \{a\}$

Оператор выбора задает объединение множеств: $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$

Конкатенация задает конкатенацию языков: $L(\alpha\beta) = L(\alpha)L(\beta)$.

Символ ε определяет пустое множество.

Оператор итерации задает итерацию: $L(\alpha^*) = L(\alpha)^*$

Приоритеты операций: сперва итерация, затем конкатенация, затем выбор.

Синтаксис регулярных выражений на практике часто расширяется, к примеру:

⁶если этого еще не сделали на этапе построения ДКА, то можно обойти его граф из стартового состояния, например, в глубину, и собрать список достижимых состояний, а остальные удалить, модифицируя при этом остальные элементы автомата

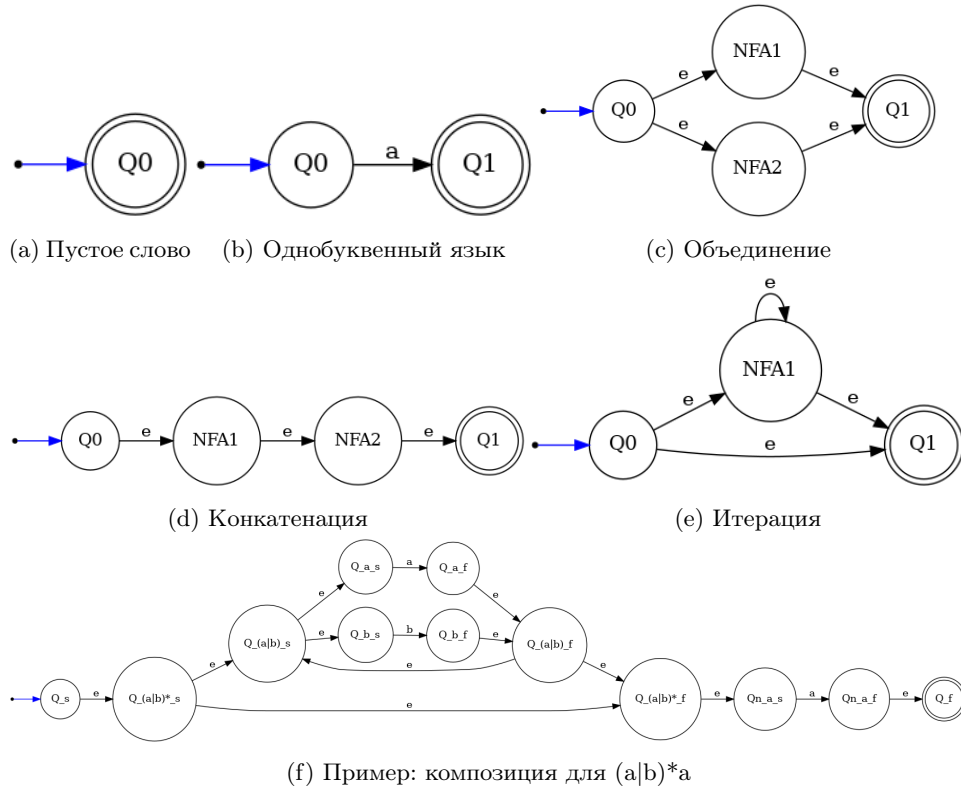


Рис. 5: Базовые автоматы построения Томпсона

- повторение один и более раз (α^+) , $(\alpha^+) = \alpha\alpha^*$
- необязательная конструкция $([\alpha])$, что означает « α или ничего»), $[\alpha] = \alpha|\varepsilon = \alpha\alpha^*$

Л. 3.1 («построение Томпсона»). Для всякого регулярного выражения α , существует NFA C_α с одним начальным и одним принимающим состояниями, распознающий язык, задаваемый α .

Доказательство производится индукцией по структуре регулярного выражения, структурные единицы представлены на Рис. 4:

Так как, по определению, класс регулярных выражений замкнут относительно этих операций, то и композиция этих операций даёт и регулярку, и НКА, её распознающую. Тем не менее, так ли регулярны регулярные выражения в современных ЯП?...

3.2 Регулярные языки

Любое регулярное выражение $reg(\Sigma)$ над алфавитом Σ задает регулярный язык L_{reg} .

Утв. 3.1 Любой регулярный язык задаётся грамматикой $\langle \Sigma, N, S, P \rangle$, где правила из P имеют вид $A \rightarrow a, A \rightarrow \gamma, A \rightarrow \epsilon$, где γ - либо aB (правая регулярная грамматика), либо Ba (левая регулярная грамматика), $a \in \Sigma, A, B, S \in N$.

3.2.1 Свойства замкнутости регулярных языков

Операции, сохраняющие регулярность: ...объединение, пересечение, дополнение, разность, обращение, итерация, конкатенация, гомоморфизм, обратный гомоморфизм.

3.2.2 Проверка на нерегулярность

Лемма о накачке (разрастании)

3.3 Регулярные выражения на практике

Регулярные выражения, входящие в современные языки программирования (в частности, PCRE в Perl), имеют больше возможностей, чем то, что мы рассмотрели: в них есть нумерованные обратные ссылки и т.д. Это позволяет задавать ими не только регулярные языки, но и более сложные, в частности, контекстно-свободные [4].

Пример (из [4]): $/^{\wedge}(a(?1)?b)\$/$ задаёт язык $a^n b^n, n \in [1 \dots \infty)$

Это регулярное выражение очень простое: $(?1)$ ссылается на первую подмаску — $(a(?1)?b)$. Можно заменить $(?1)$ подмасками, формируя таким образом рекурсивную зависимость:

```
/^{\wedge}(a(?1)?b)\$/  
/^{\wedge}(a(a(?1)?b)?b)\$/  
/^{\wedge}(a(a(a(?1)?b)?b)?b)\$/  
/^{\wedge}(a(a(a(a(?1)?b)?b)?b)?b)\$/  
...
```

Очевидно, это выражение способно описать любую строку с одинаковым количеством a и b , а конечный автомат, распознающий язык всех таких строк, построить нельзя.

4 Лексический анализ

Следующее приложение регулярных языков, о котором мы будем говорить – лексический анализ – выделение во входном тексте характерных подстрок, «значащих» что-то для дальнейших действий.

Опр. 4.1 Лексема – последовательность символов, удовлетворяющая некоторому заданному требованию.

Основная проблема выделения лексем – их может быть много и разных. Давайте работать не с лексемами, а с их «классами», на которые они делятся по смыслу нашей задачи.⁷

Опр. 4.2 Токен – последовательность символов, «осмысленно» описывающая класс некоторой лексемы.

Пример: $int \rightarrow TYPE$ (int – лексема, $TYPE$ – токен).

Для задания токенов, как правило, используют регулярные выражения.

Опр. 4.3 Лексер, лексический анализатор, сканер – транслятор, преобразующий входную строку в последовательность токенов.

Схема перехода от регулярки к ДКА: $regex \rightarrow NFA \rightarrow simplifiedNFA \rightarrow DFA \rightarrow minDFA$ была разобрана в разделах 1-3.

Примеры работы с генератором лексических анализаторов `flex` были приведены на семинаре.

5 КС-грамматики и языки

Опр. 5.1 Контекстно-свободная грамматика – это четвёрка $G = (\Sigma, N, R, S)$, где:

- Σ – терминальный алфавит.
- N – нетерминальный алфавит.
- Конечное множество правил R вида $N_i \rightarrow \alpha, N_i \in N, \alpha \in \{\Sigma \cup N\}^* \cup \{\varepsilon\}$
- Начальный символ $S \in N$.

formula (1)

5.1 Необходимые определения из близких областей

5.1.1 Графы

В данном курсе мы будем рассматривать только конечные ориентированные помеченные графы, подразумевая под "графами" именно такие графы, если не указано противное.

Опр. 5.2 Граф $G = (V, E, L)$, где V – конечное множество вершин, E – конечное множество рёбер, L

⁷Здесь считаем такую классификацию однозначной.

Опр. 5.3 Отношением достижимости на графе в смысле нашего определения называется двухместное,

Опр. 5.4 Транзитивным замыканием графа называется транзитивное замыкание отношения достижимости по всему графу.

5.1.2 Языки и грамматики

Опр. 5.5 Словом, или строкой

Опр. 5.6 Множества строк называют «формальными языками» или просто «языками». Если Σ – алфавит, то Σ^* – множество всех строк над ним, и языком называется всякое подмножество Σ^* .

Опр. 5.7 Формальная грамматика – это четвёрка $G = (\Sigma, N, R, S)$, где:

- Конечное множество символов Σ – терминальный алфавит – алфавит определяемого языка.
- Конечное множество символов N – нетерминальный алфавит⁸ – множество промежуточных символов – множество определяемых в грамматике свойств строк, которым всякая строка над алфавитом Σ обладает или не обладает. $\Sigma \cap N \equiv \emptyset$ во избежании путаницы. В лингвистике нетерминалы называются синтаксическими категориями.
- Конечное множество R правил, каждое из которых описывает возможную структуру строк со свойством $A \in N$ в как конкатенацию ..., где $B_1, \dots, B_l (l > 0)$ – все нетерминальные символы, на которые ссылается правило, а любые символы, написанные между ними, образуют строки u_0, u_1, \dots, u_l , $S \in N$ – стартовый нетерминал – обозначает множество всех синтаксически правильных строк, определяемых в грамматике.

5.1.3 Матрицы

Список литературы

- [1] [http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_за_О\(n^5E2\)_с_построением_пар_различимых_состояний](http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_за_О(n^5E2)_с_построением_пар_различимых_состояний)
- [2] [http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_Хопкрофта_\(сложность_О\(n_log_n\)\)](http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_Хопкрофта_(сложность_О(n_log_n)))
- [3] http://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Бржозовского
- [4] <https://habr.com/ru/post/171667>, 2013 (перевод, оригинал тоже гуглится).

⁸ _