Теория формальных языков, IV семестр

Весна 2016, лектор: Дворкин Михаил Эдуардович

Автор: Всеволод Степанов

Собрано: 31 августа 2016 г. 17:59

Оглавление

0.1. Введение

Курс будет читаться придерживаясь первых семи глав книги Хопкрофта, Мотвани и Ульмана "Введение в теорию автоматов языков и вычислений".

На практиках будем решать задачки, разбирать домашки. Помимо этого, говорят, что мы иногда будем что-то кодить, а еще есть вероятность, что будет что-то типа контестов (с автоматической системой проверки), если успеют допилить.

Хочется как-то взять и научиться для языков строить какие-то вменяемые системы формальных правил, описывающих эти языки. Для некоторых это сделать просто (например, языки программирования, у них, как правило, есть весьма четкая и строгая спецификация, а у какого-нибудь brainfuck она вообще занимает всего лишь страницу, при этом он Тьюринг-полный).

С естественными языками все не так гладко — принимались разные попыки их как-то формализовать, структуризировать, или же изобрести новый язык, с которым все это сделать, но все как-то безуспешно. Тем не менее, попытки не прошли совсем зря, какие-то результаты были и этим пользуются.

Работать будем с **конечным** непустым алфавитом Σ. Почти всегда нам его размер будет не особо важен, будем работать с алфавитом $\Sigma = \{0, 1\}$, будем проговаривать, если это не так.

Def 0.1.1. Словом будем называть **конечную** последовательность элементов алфавита. Слово может быть пустым, оно такое одно, обозначается как ε.

Def 0.1.2. Язык — множество слов. Язык может быть пустым.

Замечание 0.1.1. Конечность нам почти везде будет важна, без нее почти все сломается.

Заметим, что языки у нас бывают либо конечными, либо счетными: всегда можно просто выписать все слова, посортировав их по длине, а равные по длине – лексикографически в силу конечности алфавита.

А множество всех языков будет равномощно $2^{\mathbb{N}}$. В частности, это значит, что есть неразрешимые языки (привет, матлогика). Это весьма печально, так как для того, чтобы описать язык, надо, по сути, научиться для слова понимать, принадлежит ли оно этому языку, то есть, решить задачу распознавания.

Можно попробовать ограничиться лишь разрешимыми языками, но это тоже плохая идея. Большинство алгоритмов просто будут работать очень и очень долго, про них лишь будет известно, что когда-то они завершаются, но непонятно, когда. Это все очень плохо и неприменимо на практике. Поэтому, глобальная цель курса — изучать какие-то простые системы для распознавания каких-то подмножеств разрешимых языков. Стоит отметить, что далеко не для всех языков нам удастся это сделать.

Глава 1

Конечные автоматы

В этом параграфе везде будем работать с алфавитом $\Sigma = \{0, 1\}$, но для других алфавитов все будет аналогично.

1.1. Детерминированные конечные автоматы

Def 1.1.1. Детерминированным конечным автоматом (ДКА, DFA) называют пятерку $(\Sigma, Q, q_0, T, \delta)$, где:

Q — конечное множество всех состояний

$q_0 \in Q$ – начальное состояние

$T \subseteq Q$ – множество всех терминальных состояний (может быть и пустым)

$\delta : Q \times \Sigma \to Q$ – функция перехода

Автоматам умеем "скармливать" строчки: итерируемся слева направо по символам строки. Пусть сейчас мы находились в состоянии q, в строке был символ c, тогда наше новое состояние это $\delta(q, c)$.

Def 1.1.2. Обобщенная функция перехода $\hat{\delta}(q, w) = \begin{cases} q, w = \varepsilon \\ \hat{\delta}(\delta(q, c), s), w = cs, c \in \Sigma, s \in \Sigma^* \end{cases}$

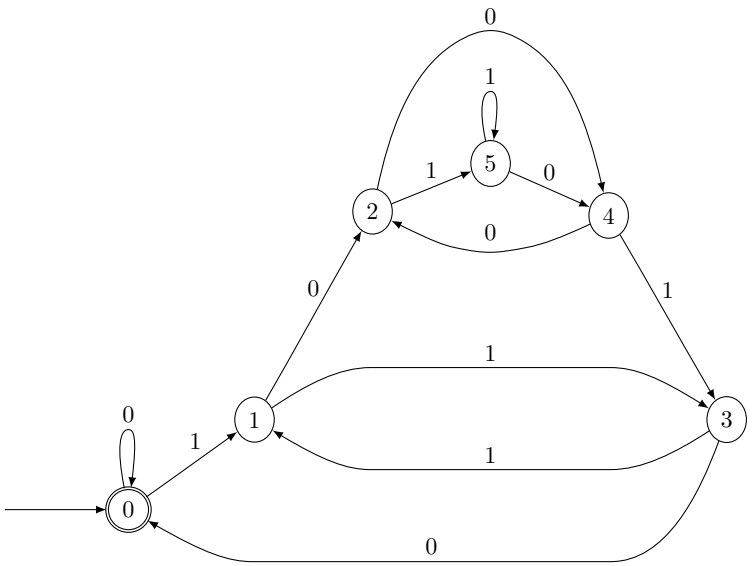
Def 1.1.3. Автомат принимает слово w, если $\hat{\delta}(q_0, w) \in T$.

Замечание 1.1.1. На самом деле, если мы прогнали слово через автомат, то мы заодно для каждого его префикса поняли, принимается ли он автоматом.

Def 1.1.4. A — ДКА, $L(A) = \{w : A \text{ принимает } w\}$ — язык, распознаваемый A.

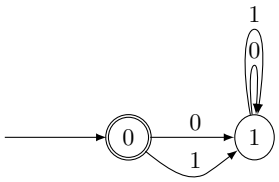
Пример 1.1.1. Рассмотрим, в качестве примера, автомат, принимающий только числа (записанные в двоичном виде, разумеется), делящиеся на 6.

Будем поддерживать следующий инвариант: пусть мы скормили какой-то префикс строки, стоим в вершине i, тогда остаток этого префикса при делении на 6 будет равен i. Тогда дописывание к слову справа нуля — умножение на 2, дописывание 1 — еще добавить 1. Терминальное состояние будет только одно – вершина с номером 0.



Def 1.1.5. Размер ДКА — количество состояний в нем

Пример 1.1.2. Еще один полезный пример — автомат, принимающий только пустое слово:



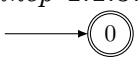
Def 1.1.6. Тупиковое состояние — состояние, из которого невозможно дойти до хоть какого-нибудь терминального.

Замечание 1.1.2. Иногда такие состояния называют дьявольскими. Это вполне себе принятое обозначение, но, все-таки, обычно используют термин “тупиковое состояние”.

В автомате для языка $\{\varepsilon\}$, нарисованном выше, вершина 1 как раз является тупиковой. Есть договоренность, что такие вершины при изображении автомата можно опускать, при этом, если из вершины нет перехода по какому-то символу, то он, на самом деле, ведет в тупиковое состояние.

Замечание 1.1.3. При этом, однако, надо помнить, что это состояние может присутствовать, например, когда считаем размер автомата.

Пример 1.1.3. Вот так вот будет выглядеть автомат, если убрать тупиковое состояние:



Не забываем, что в нем, на самом деле, две вершины.

Мы сопоставили каждому ДКА какой-то язык. В то же время, каждому языку может соответствовать несколько ДКА: например, можно просто добавить кучу вершин, недостижимых из начальной.

Def 1.1.7. Два автомата называются эквивалентными, если они распознают один и тот же язык.

Понятно, что мы сейчас разбили автоматы на классы эквивалентности, хочется взять и найти в каждом классе какого-то канонического представителя. Кажется вполне логичным, что чем меньше автомат, тем лучше. Например, с точки зрения скорости вычислений.

Для начала возьмем и построим алгоритм, который берет и строит по данному ДКА эквивалентный ему, но меньшего размера, а потом докажем, что этот алгоритм нам даст канонического представителя (в частности, надо будет показать, что мы получаем минимальный автомат, и при этом он ровно один).

1.1.1. Алгоритм минимизации ДКА

Def 1.1.8. Два состояния p, q называются различимыми, если $\exists w \in \Sigma^* : \hat{\delta}(p, w) \in T \oplus \hat{\delta}(q, w) \in T$ — найдется такое слово, что если по нему из этих двух состояний перейти, то ровно одно из двух итоговых состояний будет терминальным.

Соответственно, состояния неразличимы, если множества слов, которые они принимают, совпадают.

Замечание 1.1.4. Отношение неразличимости, как несложно убедиться, будет являться отношением эквивалентности.

Собственно, сам алгоритм:

0. Возьмем, обойдем автомат `dfs`-ом как орграф, некоторые вершины посетим, некоторые нет. Удалим непосещенные вершины, так как мы в них никогда не придем, вне зависимости от входа.

1. Хотим для каждой пары состояний узнать, различима она, или нет. Конечная цель — разбить все вершины на классы эквивалентности.

   Изначально мы знаем, что пара состояний $(t \in T, q \notin T)$ различима: в качестве w в определение можно подставить ε.

2. **Лемма 1.1.1.** Если (r, s) — пара различимых состояний, $\delta(p, x) = r, \delta(q, x) = s$, то (p, q) – тоже пара различимых состояний.

   ▶ Достаточно просто взять и к различающей (r, s) строке добавить в начало x. ◀

   Рассмотрим граф, вершины в котором — все возможные пары состояний. Между двумя вершинами $(p_1, q_1), (p_2, q_2)$ проведем ребро с символом c, если $\delta(p_1, c) = p_2, \delta(q_1, c) = q_2$. Обойдем этот граф по обратным ребрам, запуская `dfs` из всех пар вида (терминальная, нетерминальная), и наоборот.

3. Посещенные `dfs`-ом вершины это в точности пары различимых состояний.

   Очевидно, что если мы вершину посетили, то соответствующая ей пара состояний различима — просто посмотрим на путь из стартовой вершины обхода до данной, применим по индукции к каждой вершине этого пути лемму.

   То, что непосещенной вершине соответствует пара неразличимых состояний тоже понятно. Пусть они различимы, тогда, по определению, существовал бы путь до пары $(p, q) : p \in T, q \notin T$, или наоборот. То есть, вершина была бы достижима из исходных. Противоречие.

4. Итак, мы смогли взять и разбить вершины на классы эквивалентности. Возьмем и каждый класс стянем в одну вершину, если был переход из p в q по символу c, то теперь будет переход из $[p]$ в $[q]$ по тому же символу, где $[p], [q]$ – классы эквивалентности для p, q. От кратных ребер, понятное дело, избавились. Если $q \in T$, то скажем, что в полученном автомате $[q] \in T$. Соответственно, начальной вершиной будет $[q_0]$.

   Надо понять, что мы сейчас получили корректный автомат.

   Для начала заметим, что каждое терминальное состояние в новом автомате состоит только из терминальных состояний старого, так как любая нетерминальная вершина различима с терминальной.

   Теперь надо посмотреть на переходы и понять, что из одного состояния по каждому символу есть ровно один переход.

Легко заметить, что хотя бы один точно будет, так как он был в исходном автомате. Пусть их хотя бы два, при этом они ведут в разные вершины, то есть, в разные классы эквивалентности $[p], [q]$.

Пусть в p был переход из s, а в q из r, $[s] = [r]$. Понятно, что вершины p, q различимы, так как они находятся в разных компонентах. Ну а тогда по лемме вершины s, r тоже различимы, то есть, лежат в разных классах эквивалентности, противоречие.

5. Итак, у нас есть корректный автомат, в нем вершин не больше, чем в исходном.

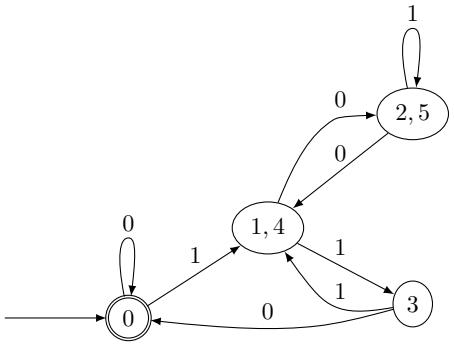
   А теперь заметим, что скармливанию строки исходному автомату можно сопоставить скармливание строки сжатому — просто взяли последовательность состояний $q_0, q_1, \dots, q_{|w|}$, отобразили ее в $[q_0], [q_1], \dots, [q_{|w|}]$, при этом результат (строка принята/не принята) не изменился: $q_{|w|} \in T \iff [q_{|w|]} \in T$.

6. Осталось еще оценить его асимптотику. Всего у нас $\mathcal{O}(n^2)$ состояний и $\mathcal{O}((n\Sigma)^2)$ ребер. Но, на самом деле, далеко не все ребра мы будем рассматривать: из каждой пары вершин есть всего Σ допустимых переходов, так как мы переходим одновременно по одинаковому символу. И итоговая асимтотика будет, на самом деле, $\mathcal{O}(n^2\Sigma)$

   Как правило, Σ в асимптотике опускают, так как это какая-то константа, которая нас не очень волнует. Тогда асимптотика просто $\mathcal{O}(n^2)$.

Замечание 1.1.5. Существует алгоритм Хопкрофта, работающий за $\mathcal{O}(n \log n)$, который даже вполне применим на практике. Рассказ занимает где-то одну лекцию, он реально сложный для понимания, поэтому рассматривать его не будем, желающие могут с ним ознакомиться.

Пример 1.1.4. Если применить этот алгоритм к автомату, принимающему числа, делящиеся на 6, получим следующее (при желании можно честно выписать пошагово что алгоритм делает и убедиться, что все будет действительно именно так):



1.1.2. Минимизация ДКА

Def 1.1.9. $C_L^R(x) = \{y \in \Sigma^*, xy \in L\}$ — правый контекст в языке L слова x.

Пример 1.1.5. $C^R_{RUSSIAN}(\text{“сту”}) = \{$л, лор, дент, к, ...$\}$ — множество всех возможных слов, которые можно приписать к “сту” так, чтобы в итоге получилось корректное слово из русского языка.

Замечание 1.1.6. Правые контексты могут пересекаться(например, правые контексты всех слов, принимаемых автоматом, содержат пустую строку)

Несложно понять, что $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) \Rightarrow C^R_{L(A)}(x) = C^R_{L(A)}(y)$: что бы мы не дописали к словам x, y, мы будем двигаться из одной и той же вершины, придем тоже в одну и ту же вершину.

Рассмотрим два слова x, y, посмотрим, что можно сказать в случае, когда их правые контексты не равны: $C^R_{L(A)}(x) \neq C^R_{L(A}(y) \Rightarrow \exists z : xz \in L(A), yz \notin L(A)$. Тогда $(\hat{\delta}(q_0, x), \hat{\delta}(q_0, y))$ — пара различимых вершин (просто по определению).

Утверждение 1.1.1. $\begin{cases} C^R_{L(A)}(x) = C^R_{L(A)}(y) \\ \hat{\delta}(q_0, x) \neq \hat{\delta}(q_0, y) \end{cases}$ $\Rightarrow$ состояния x, y можно объединить в одно

▶ Правые контексты равны следовательно, вершины являются неразличимыми. Тогда, по алгоритму минимизации автомата, их можно стянуть в одну вершину. ◀

Теорема 1.1.1. Количество вершин в минимальном ДКА будет в точности равно количеству различных правых контекстов в языке.

▶ $\geqslant$: различным правым контекстам должны сооветствовать разные состояния, так как они различимы

$\leqslant$: если есть два состояния с одинаковыми правыми контекстами, то их можно склеить.

◀

Теорема 1.1.2. Алгоритм минимизации ДКА строит минимальный автомат с точностью до изоморфизма.

▶ Понятно, что мы построили минимальный автомат по размеру. Осталось лишь понять, что все остальные минимальные ему изоморфны.

Берем два минимальных автомата. У начальных вершин правые контексты совпадают — это просто множества всех принимаемых слов.

Теперь, пусть мы перешли из состояния с правым контекстом $C^R_L(x)$ по символу c, обязательно пришли в состояние с правым контекстом $C^R_L(xc)$, значит, изоморфизм действительно есть. ◀

Теорема 1.1.3. У L конечное множество правых контекстов $\iff$ $\exists$ ДКА, принимающий L

▶ $\Leftarrow$ минимизируем автомат, получили все различные правые контексты.

$\Rightarrow$ построим ДКА A. $\forall C^R_L(x)$ становится состоянием.
Начальное состояние $C^R_L(\varepsilon)$
Состояние терминальное, если $\varepsilon \in C^R_L(x)$
Переходы: $C^R_L(x)$ по $a \to C^R_L(xa)$. Надо понять, что получили корректный автомат, который примет ровно то, что нужно.
Корректность — из каждой вершины по каждому символу ровно один переход. Казалось бы, может получиться так, что переходов несколько: стоит понимать, что для разных слов x, y может выполняться $C^R_L(x) = C^R_L(y)$, тогда вершины будут совпадать и мы как бы нарисуем два перехода по каждой букве. Но если вершины совпадают, то переходы по одной и той же букве приведут нас в одну и ту же вершину, поэтому, на самом деле, эти несколько переходов — кратное ребро, которое можно сократить в обычное.
Несложно понять, что, перейдя из стартового состояния по слову x, мы попадем в состояние, соответствующее $C^R_L(x)$. Тогда мы примем слово x только в том случае, если $\varepsilon \in C^R_L(x)$, что равносильно тому, что $x \in L$.

◀

Пример 1.1.6. Несложно построить пример языка с бесконечным количеством правых контекстов.

Можно взять язык ПСП. Рассмотрим $open_i$ = слово из i открывающих скобочек, а $close_i$ — слово из i закрывающих.

Рассмотрим тогда $C^R_L(open_i), i \in \{1, 2, ...\}$. Все эти правые контексты будут различными, так как $close_i$ будет принадлежать только $C^R_L(open_i)$.

1.1.3. Проверка эквивалентности автоматов

Def 1.1.10. Есть два ДКА A, B.

Рассмотрим пару состояний $(p \in A, q \in B)$. Будем говорить, что эта пара состояний различима, если $\exists w : (pw \in L(A) \oplus qw \in L(B))$ — найдется такое слово, что если по нему из этих двух состояний перейти, то ровно одно из двух итоговых состояний будет терминальным.

Замечание 1.1.7. По сути, это определение абсолютно аналогично определению различимости состояний внутри одного автомата.

Тогда сделаем следующее:

Рассмотрим все пары состояний вида $(p \in A, q \in B)$. Скажем, что пары вида $(p \in T(A), q \notin T(B))$ и $(p \notin T(A), q \in T(B))$, где $T(A)$ — терминальные состояния автомата A — пары различимых состояний (подставим ε в определение).

А дальше — действия будут абсолютно аналогичны тому, что мы делали в алгоритме минимизации автомата, разницы никакой нет.

Эквивалентность $\iff$ пара из $(q_{0,A}, q_{0,B})$ — пара неразличимых состояний.

1.2. Пересечение, объединение, разность автоматов

$L_1 = L(A), L_2 = L(B)$ Рассмотрим языки $L_1 \cap L_2, L_1 \cup L_2, L_1 \setminus L_2, \Sigma^* \setminus L_1$. Хочется понять, что если существовали автоматы для L_1, L_2, то и для всех этих языков автоматы тоже будут существовать.

По языку L можно построить автомат $\Rightarrow$ по его дополнению тоже можно (все терминальные вершины нетерминальными и наоборот). С точки зрения правых контекстов: $C^R_{\overline{L}}(x) = \overline{C^R_L(x)}$. Конечность количества правых контекстов не изменилась, следовательно, существование автомата тоже не изменилось.

Автомат для $L_1 \cap L_2$ существует, так как $C^R_{L_1 \cap L_2}(x) = C^R_{L_1}(x) \cap C^R_{L_2}(x)$, оно тоже конечно. Однако, количество состояний в этом автомате может достигать nm, где n, m — количества состояний в автоматах для L_1, L_2 соответственно.

Def 1.2.1. Пусть было два ДКА: $(\Sigma, Q, q_0, T, \delta)$ и $(\Sigma, Q', q'_0, T', \delta')$.

Их прямым произведением называют следующий автомат: $(\Sigma, Q \times Q', (q_0, q'_0), T \times T', \delta'')$, где $\delta''((p, q), a) = (\delta(p, a), \delta'(q, a))$.

Несложно понять, что построенный таким образом автомат будет принимать ровно $L_1 \cap L_2$

Существование автомата для объединения языков тоже можно выразить через правые контексты, надо только $\cap$ заменить на $\cup$. А для того, чтобы построить явную конструкцию, надо взять прямое произведение автоматов, и сказать, что терминальные состояния это $(T \times Q') \cup (Q \times T')$.

Абсолютно те же слова говорим и про разность. Конструкция такая же, терминальные состояния для $L_1 \setminus L_2$ — $T \times (Q' \setminus T')$

1.2.1. Динамическое программирование по ДКА

Замечание 1.2.1. Этой темы нет в книжке. Впрочем, нам все это и так должно быть знакомо, спасибо Серёже.

Общая идея следующая: у нас есть какой-то автомат, хочется на него внимательно посмотреть и понять какие-нибудь его свойства. Например, какие-нибудь характеристики слов, которые он принимает — слово минимальной длины, принимает ли он пустое слово, количество принимаемых слов длины i, и так далее. Задачи могут быть какимие-то более сложными, но мы не будем сильно в это все углубляться.

Минимальное слово, которое принимает — `bfs` по состояниям.

Хотим посчитать количество слов в $L(A)$ длины n. Для этого нам сейчас надо будет написать динамику.

Заведем табличку $a_{q,i}$ — количество слов $x : |x| = i, \hat{\delta}(q_0, x) = q$. Иными словами, сколько слов длины i приводит нас в состояние q.

Эту штуку надо как-то считать. Начальные состояния при $i = 0$ будут следующие:

$\begin{cases} a_{q,0} = 0, q \neq q_0 \\ a_{q,0} = 1, q = q_0 \end{cases}$

Переходы:

$a_{q,i} = \sum\limits_{p,c|\delta(p,c)=q} a_{p,i-1}$. Важно, что суммируем еще и по символу: если из p в q было несколько переходов по разным буквам, то их все надо учесть.

Теперь это дело можно честно посчитать за $\mathcal{O}(n\Sigma \cdot i)$. Ответом для длины i будет являться $\sum\limits_{q \in T} a_{q,i}$.

Однако, i можно захотеть взять достаточно большим, тогда все будет работать достаточно долго.

Запихнем все значения для $i-1$ в вектор-столбец A_{i-1}. Хотим через него посчитать A_i в терминах матриц. Для этого надо взять и выразить через умножение вектора на матрицу суммирование в переходе.

Несложно убедиться, что $A_i = MA_{i-1}$, где M — матрица смежности автомата (с учетом кратности). Тогда, $A_i = M^i A_0$, а в степень можно возводить достаточно быстро, получим итоговую асимптотику равную $\mathcal{O}(\log(i) \cdot n^3)$

Еще мы сейчас получили, что количество слов длины n либо линейно, либо зависит от n экспоненциально.

Замечание 1.2.2. Это должно нам казаться интуитивно понятным, возможно, мы должны были это пройти (но не прошли) в курсе линейной алгебры, Миша так сходу не умеет этот факт объяснять.

Анонс следующей лекции:

Рассмотрим язык, состоящий из всех слов таких, что 10 с конца символ равен нулю, остальные — не важны. Несложно понять, что в минимальном автомате для этого языка будет около 2^{10} вершин — надо честно помнить все последние 10 символов, и ничего с этим не поделаешь. Как-то многовато получается, а ведь это еще сравнительно простой язык, казалось бы.

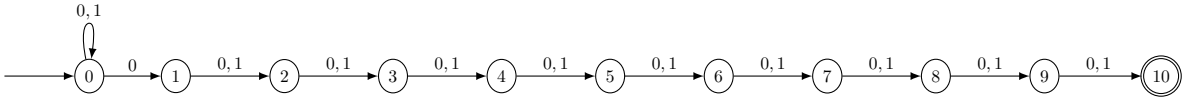
На следующей лекции будем пытаться бороться с этим с помощью недетерминированных конечных автоматов.

1.3. Недетерминироавнные конечные автоматы с ε-переходами (ε-НКА, NFA-ε)

Язык с прошлой лекции — все слова, на 10 месте с конца которых стоит нолик. Мы сказали, что можно построить большой ДКА из 2^{10} состояний, но это как-то много для такого, казалось бы, простого языка. Тут к нам на помощь приходят НКА.

Идея НКА в следующем: давайте скажем, что из состояния по символу может идти не один переход, а сразу несколько, и, когда мы скармливаем слово автомату, мы будем выбирать такой переход, который хотим. И будем говорить что слово автоматом принимается, если мы можем выбирать каждый раз переходы так, чтобы в итоге прийти в терминальное состояние.

Пример 1.3.1. Используя эту идею, для описанного выше языка можно построить следующий НКА:



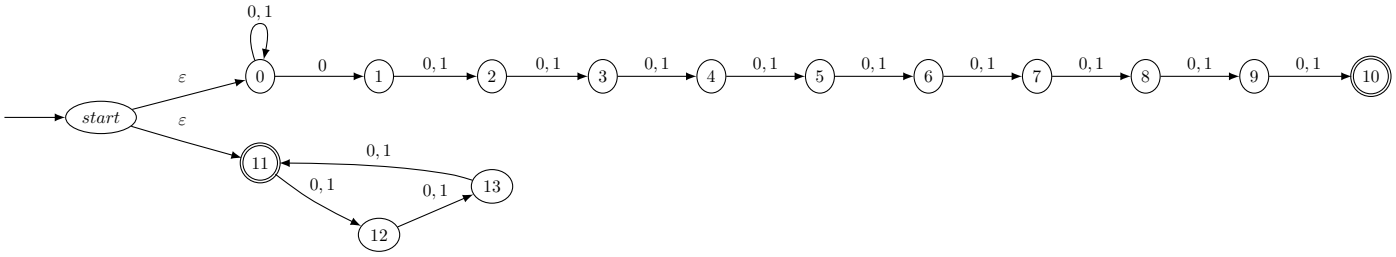
Посмотрим на него внимательно и покажем, что он принимает ровно этот язык. Пусть мы пропускали через автомат слово длины n из языка. Тогда скажем, что мы $n - 10$ раз пройдем по петле из нулевого состояния в нулевое, а потом оставшиеся символы пройдем по цепочке до терминального состояния. Очевидно, что если слово принадлежало языку, то мы так сможем сделать.

В другую сторону: чтобы принять какое-то слово, мы должны взять и последние 10 символов пропустить через "цепочку" до терминального состояния, а первый переход в ней возможен только по нулю. Поэтому ничего лишнего мы точно принять не сможем.

Итого, мы построили автомат из всего лишь 11 вершин, что достаточно круто.

Пример 1.3.2. Чуть более сложный пример: объединим предыдущий язык со словами, длина которых делится на 3.

Для того, чтобы построить автомат для этого языка, воспользуемся еще одной идеей — ε-переходами. Скажем, что на некоторых ребрах будет написан не символ из алфавита, а специальный символ $\varepsilon \notin \Sigma$. Переход по ε отличается от перехода по символу тем, что его можно сделать всегда, при этом мы текущий символ в строке не считываем.



Если мы хотим принять слово, длина которого делится на 3, то просто сделаем переход в состояние с номером 11, после чего будем бегать по циклу $11 - 12 - 13$. Иначе, пойдем в состояние с номером 0.

Несложно понять, что данный автомат примет ровно описанный язык.

Формализуем две идеи, описанные выше.

Def 1.3.1. ε-НКА $= (\Sigma, Q, q_0, T, \Delta)$.

Q — конечное множество состояний

$q_0 \in Q$ — стартовое состояние

$T \subset Q$ — множество терминальных вершин

$\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to (2^Q \setminus \emptyset)$ — функция переходов. Собственно, в этом и есть отличие ε-НКА от ДКА: если у ДКА функция сопоставляла паре из состояния и символа ровно одно состояние, куда мы должны перейти, то здесь у нас есть множество (непустое) допустимых для перехода состояний. А еще добавили специальный символ ε.

Def 1.3.2. $E(q)$ — ε-замыкание состояния q — множество состояний, достижимых из q только по ε-переходам.

Def 1.3.3. Слово $a_1 \dots a_n$ приниманется ε-НКА, если $\exists$ последовательность состояний $p_0, \dots, p_n$, такая что $p_0 \in E(q_0)$, $p_n \in T$, $\forall i \in [1, n] : p_i \in E(\Delta(p_{i-1}, a_i))$.

1.3.1. Алгоритм приема слова ε-НКА

Мы научились описывать некоторые языки достатчно компактными автоматами, чего не умели делать раньше, но пока что не очень понятно как для произвольного слова понимать, принимается ли оно автоматом, или нет, кроме как экспоненциального перебора всех возможных путей. Сейчас будет предложен хороший адекватный полиномиальный алгоритм, который обычно и используют на практике.

Будем прогонять по символу слово через автомат, в каждый момент времени поддерживаем R — множество состояний, в которых может находиться ε-НКА после принятия соответствующего префикса и, возможно, еще осуществления некоторого количества ε-переходов.

Изначально $R = E(q_0)$ (важно, что не просто $\{q_0\}$!)

При прочтении символа a: $R' = \bigcup\limits_{q \in R} E(\Delta(q, a))$.

Текущее слово принимается $\Leftrightarrow R \cap T \neq \emptyset$.

Это уже какой-то нормальный алгоритм, он работает за полином, оценим его время работы в худшем случае. $|R| \leqslant |Q|$ каждую итерацию (для каждого префикса). Надо на каждой итерации посчитать количество переходов, которые мы совершаем. Переходов по букве не более Q^2: из каждого состояния можно перейти в не более чем Q. Теперь, если еще реализовать алгоритм совсем наивно, то получится вообще Q^3 переходов на одну букву принимаемого слова: ε-переходов тоже может быть порядка Q, если их честно каждый раз все обходить (а это надо запускать `dfs` на графе ε-переходов), то Q^3 и получится. Получили что-то полиномиальное, но алгоритм можно улучшить.

Сейчас мы каждый раз зачем-то честно считаем переходы по символу, но их можно взять и один раз предподсчитать, запомнив для пары (состояние, буква) $E(\Delta(q, a))$ в виде битсета. Тем самым, мы улучшим асимптотику в Q раз, так как для каждой пары (состояние, буква) нам не надо каждый раз проходить явно по ε-переходим.

Итого, получили $\mathcal{O}(nQ^2)$. На самом деле, так как на практике внутри у нас будут битсеты, то получается $\mathcal{O}(nQ^2/w)$, где w — длина машинного слова, например, 64.

Можно вообще поделить на $\log |Q|$, применив четырех русских, но в реальной жизни на не совсем больших автоматах это не очень то и нужно...

Замечание 1.3.1. На практике, однако, алгоритм работает быстрее, так как маловероятно, что из каждого состояния по каждой букве будет по Q переходов.

Замечание 1.3.2. Заметим, что подобную штуку для недетерминированной МТ уже не прокрутить, так как там нам, помимо состояния, важно еще, что написано на ленте, это уже нам даст дофига возможных конфигураций.

1.3.2. Алгоритм детерминизации ε-НКА

Есть ε-НКА $A = (\Sigma, Q, q_0, T, \Delta)$. Хотим взять и конструктивно построить ДКА $B = (\Sigma, Q', q_0', T', \delta)$, который будет ему эквивалентен.

$Q' = 2^Q$, каждое состояние кодирует какое-то подмножество в Q — те состояния, в которых мы сейчас можем находиться (как R в предыдущем пункте).

$q_0' = E(q_0)$

$T' = \{p \subset Q : p \cap T \neq \emptyset\}$ — мы говорим, что состояние p терминальное, если хоть одно состояние из НКА, лежащее в множестве, соответствущему p, является терминальным. То есть, состояние в новом автомате терминальное, если в исходном недетерминированном автомате можно выбирать переходы так, чтобы попасть в терминальное состояние $\delta(R, a) = \bigcup\limits_{q \in R} E(\Delta(q, a))$

Смотрим внимательно на функцию переходов δ, она формально описывает ровно те переходы, что мы делали в предыдущем пункте. Если мы повторим те же самые рассуждения, получим доказательство корректности нового детерминированного автомата.

Замечание 1.3.3. Заметим, что далеко не все состояния нам на практике нужны, много чего можно выкинуть. Впрочем, никаких оценок на это дело давать не будут.

Можно просто явно не выписывать автомат, а просто обойти `dfs`-ом, получить все достижимое.

Можно еще взять и выбирать какой автомат использовать. НКА, конечно, хуже, чем ДКА, так как принимает слова гораздо медленнее. Но ДКА может быть большим. Поэтому попробуем детерминизировать НКА, если чувствуем, что слишком много вершин получается, то забиваем, и

будем использовать НКА. Иначе — получили какой-то не слишком большой ДКА, который будет за $O(|w|)$ принимать слово вместо $O(|Q|^2|w|)$, что очень хорошо.

1.4. Регулярные языки

1.4.1. Академические регулярные выражения (АРВ)

Они появились раньше, чем обычные регулярные выражения. В обычных начинается всякий ад, когда начинаем использовать что-то в духе $\backslash 1$ (это когда мы взяли, заматчили какое-то выражение, а потом хотим **его же** заматчить где-то в другом месте, в академической версии такого нет, там все очень просто).

Ниже табличка того, как строятся регулярные выражения, в этих обозначениях R, R_1, R_2 — произвольные АРВ.

R	$L(R)$ — язык, принимаемый регулярным выражением R
$\emptyset$	$\emptyset$
ε	$\{\varepsilon\}$
$a \in \Sigma$	$\{a\}$
(R)	$L(R)$
замыкание Клини: R^*	$\{a_1 \dots a_n : n \in \mathbb{N} \cup \{0\},\ a_i \in L(R)\} = (L(R))^*$ (то, что n может быть нулем — важно!)
$R_1 R_2$	$\{ab : a \in L(R_1) \wedge b \in L(R_2)\}$
$R_1 \mid R_2$	$L(R_1) \cup L(R_2)$

Приоритет убывает сверху вниз.

Замечание 1.4.1. Операций не очень много (по сравнению с привычными нам регулярными выражениями), но математики любят строить что-то поменьше и покомпактнее, а потом уже на основе этого достраивать остальную функциональность.

Например, чем проще синтаксис, меньше правил, тем проще будет автомат.

Пример 1.4.1. Рассмотрим пару простых примеров АРВ:

R	$L(R)$
01	$\{01\}$
$(0\mid1)^*$	все слова из ноликов и единиц
$(0\mid11)^*$	если есть группа подряд идущих единиц, то она четной длины

Def 1.4.1. Язык L называется регулярным, если $\exists$АРВ $R : L(R) = L$.

1.4.2. Теорема Клини

Теорема 1.4.1. Множество языков, принимаемых ДКА совпадает с множеством языком, принимаемых АРВ

▶ В обе стороны будут конструктивные доказательства.

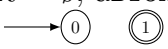
L — регулярный $\Rightarrow$ L — автоматный.

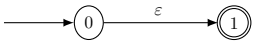
Будем сейчас строить соответствующий регулярному выражению ε-НКА.

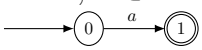
Будем брать и строить автомат “по индукции”: вот у нас как-то строится АРВ (можно, например, его представить в виде синтаксического дерева и сказать, что строим снизу вверх), будем так же строить и соответствующий ДКА. При этом будем поддерживать следующие инварианты:

1. В начальное состояние нет переходов.

2. Одно терминальное состояние, из него нет переходов.
3. Терминальное состояние не совпадает с начальным.

- $R = \emptyset$, автомат выглядит вот так:


- $R = \varepsilon$:


- $R = a$, $a \in \Sigma$


- $R = (R_1)$, автомат для такого выражения в точности равен автомату для R_1
- $R = R_1^*$. Пусть у автомата для R_1 было стартовое состояние s и терминальное t, у автомата для R будет стартовое состояние s' и терминальное t'.
  Возьмем целиком автомат для R_1, добавим следующие ε-переходы: $s' \to s$, $t \to s$, $s \to t'$.
  TODO тут должна быть красивая картинка, но у меня пока не получилось ее сделать
  Полученный автомат удовлетворяет инвариантам, описаным выше и, как несложно заметить, соответствует R: в t' мы попадаем только из s, в которое мы попадаем либо по пустой строке, либо несколько раз по строке, принимаемой R_1.
  Замечание 1.4.2. Инвариант «в начало ничего не входит» важен: иначе может быть так, что мы внутри R_1 примем слово лишь частично, а потом прыгнем в терминальное.
- $R_1 R_2$ — рисуем автоматы для R_1 и R_2, соединяем их по ε-переходу из терминального состояния у R_1 в стартовое для R_2, у терминального состояния R_1 убираем терминальность.
- $R_1 | R_2$ — берем стартовую вершину, делаем из нее по ε-переходу в автоматы для R_1 и R_2, из их терминальных вершин добавляем по ε-переходу в терминальную вершину нашего автомата, убираем у них терминальность.

В итоге получили автомат размера $\mathcal{O}(|\text{АРВ}|)$. Этот алгоритм применим на практике, если очень хочется построить автомат по АРВ самому, то вот это вот и рекомендуется писать.

L-автоматный $\Rightarrow$ L-регулярный.

Будем сейчас строить трехмерную таблицу регулярных выражений, R_{ijk} — АРВ, принимающее язык слов, которые переводят ДКА A из состояния i в состояние j с использованием промежуточных состояний с номерами $\le k$. Немного напоминает алгоритм Флойда, это не случайно, сейчас будет что-то похожее.

1. Слой $k = 0$:
   - $i = j$, $R_{ii0} = \varepsilon$. Это не совсем правильно, так как в автомате могли быть петли из i в i по символу c, в таком случае надо писать $R_{ii0} = \varepsilon | c_1 | c_2 \dots$.
     Внимание, формализм: по петле можно пройти не более одного раза, так как иначе будут промежуточные вершины, которые мы запретили в силу того, что $k = 0$, поэтому мы написали корректное АРВ.
   - $i \ne j$, $R_{ij0} = c_1 | c_2 | \dots$, c_i — символы, по которым можно перейти из i в j. Если таких символов нет, то $R_{ij0} = \emptyset$.
2. Слой $k > 0$:
   В алгоритме Флойда: мы хотим пройти из вершины i в вершину j. Либо идем по пути с промежуточными вершинами с номерами $\le k$, либо прошли по такому пути из i в k, а потом из k в j.
   Тут немножко не так — мы вершину k можем иметь несколько раз на пути из i в j, так как пройденный нами путь может быть произвольным, не обязательно кратчайшим. Надо это аккуратно записать в виде АРВ.
   $R_{i,j,k} = R_{i,j,k-1} | R_{i,k,k-1} R^*_{k,k,k-1} R_{k,j,k-1}$ — либо не использовали состояния с номерами k, либо наш путь делится на 3 части: часть из i до первого вхождения k, часть из последнего вхождения k в j, а еще несколько (возможно, ноль) раз в промежутке прошли из k в k.

Пусть $T = \{t_1, \dots, t_k\}$. Тогда АРВ, соответствующее ДКА, будет равно $R_{q_0,t_1,n}|R_{q_0,t_2,n}|\dots|R_{q_0,t_k,n}$, или же $\emptyset$, если $T = \emptyset$.

Время работы алгоритма будет $\mathcal{O}^*(4^{|Q|})$ — на каждом слое мы конкатенируем 4 АРВ, увеличивая длину в 4 раза.

Замечание 1.4.3. Здесь $\mathcal{O}^*(f(n))$ обозначает $\mathcal{O}(f(n) \cdot p(n))$ для некоторого полинома $p(n)$, т.е. «работает за $f(n)$ с точностью до полиномиального множителя».

◀

Замечание 1.4.4. Теперь часто будем употреблять термин “регулярный язык” как синоним “автоматный язык”

1.5. Лемма о накачке (Pumping lemma)

Теорема 1.5.1. Пусть L — регулярный язык. Тогда: $\exists n \in \mathbb{N}\colon \forall w \in L\colon |w| \geq n \Rightarrow \exists x, y, z\colon (w = xyz) \wedge (|x| + |y| \leq n) \wedge (|y| > 0) \wedge (\forall i \geq 0\colon xy^iz \in L)$

То же самое, но словами: для любого регулярного языка верно, что для любого достаточно длинного слова можно взять небольшую середину, повторить ее кучу раз и все равно получить слово из этого же языка.

Например, этой леммой удобно доказывать что какой-то язык не регулярен.

▶ Существует ДКА $A\colon L(A) = L$. $n = |Q|$. Рассмотрим произвольное слово $w \in L, |w| \geq n$. Будем скармливать автомату префиксы w, смотреть, куда пришли. Всего выписали $n + 1$ состояние (может и больше, рассмотрим тогда первые $n+1$), значит, по принципу Дирихле, для каких-то префиксов $x = a_1 \dots a_i$ и $xy = a_1 \dots a_i a_{i+1} \dots a_k$ состояния, им соответствующие, совпадут.

Обозначим $z = a_{k+1} \dots a_{|w|}$, тогда верно:
$\hat{\delta}(q_0, x) = q$,
$\hat{\delta}(q, y) = q$,
$\hat{\delta}(q, z) \in T$.

x, y, z подходят под условие леммы, победа.

◀

Пример 1.5.1. $L = \{0^n1^n\}$

Отрицание утверждения леммы: для любого n существует слово w длины n такое, что утверждение леммы не будет верным (то есть, нельзя разбить на три части).

Взяли произвольное n, в качестве w подставили $w = 0^n1^n$, лемму, как несложно заметить, поломали: нельзя вот так просто брать и копипастить середину.

Пример 1.5.2. $L = \{(0|1)^*\}$ (Количество 0 равно количеству 1).

Можно взять, например, $0^{2n}1^{2n}$. Так как $|xy| \leq n$ — префикс нашего слова, то y в любом случае будет состоять из одних нулей, поэтому если его мы повторим несколько раз, то все сломаем.

Глава 2

Формальные грамматики

2.1. Контекстно-свободные грамматики (context-free)

Всё, что вы сейчас увидите — это то, как люди пытались описать синтаксис построения предложения в естественный языках. Относительно получается: какие-то стандартные невычурные фразы (особенно в английском языке, где фиксирован порядок слов) даже получается описывать, однако кажется, что естественный язык намного богаче. Зато оказалось, что грамматиками можно задавать различные программистские вещи, в том числе языки программирования, регулярные выражения и так далее.

Замечание 2.1.1. Формальные грамматики — formal grammar.

Def 2.1.1. КС-грамматика (контекстно-свободная грамматика, context-free grammar, cf-grammar) G — это четвёрка (Σ, N, P, S) такая, что:

- Σ — конечный алфавит, его элементы мы назваем *терминалами*.
- N — некоторое конечное множество, причём $\Sigma \cap N = \emptyset$. Его элементы мы называем *нетерминалами/переменными/синтаксическими категориями* (это то, как его называли еще лингвисты: существительное, глагол, дополнение, предложение).
- $S \in N$ — *стартовый символ/стартовая переменная*.
- P — конечное множество продукций вида $A \to \alpha$, $A \in N$, $\alpha \in (N \cup \Sigma)^*$.
  P — конечное подмножество $2^{N \times (N \cup \Sigma)^*}$.

Замечание 2.1.2. Иногда будем употреблять слово *символ* для элементов $\Sigma \cup N$.

Замечание 2.1.3. Контекстно-свободная в том смысле, что слева в продукциях стоит только один символ. Например, правила $aAa \Rightarrow aABa$ недопустимы, а они, в каком-то смысле, зависят от контекста: A переходит в AB только если его окружают a.

Замечание 2.1.4. Довольно бессмысленно рассматривать бесконечные системы: наверняка бесконечной системой описать бесконечный язык несложно, а вот если можно описать конечной системой — это хорошо.

Замечание 2.1.5. Часто есть договоренность, заглавнеы буквы — нетерминалы, при этом, S — стартовый.

Замечание 2.1.6. P можно рассматривать, например, вот так: есть синтаксическая категория, например, <арифметическое выражение>. Мы его можем раскрыть как <число>, например, 239 — корректное арифметическое выражение, а можем, например, как <арифметическое выражение> <плюс> <арифметическое выражение>. Например, 239 + 30 + 179 будет принадлежать такой грамматике.

Замечание 2.1.7. Кстати, с контестно-свободными грамматиками мы должны быть знакомы, например, по форме Бэкуса-Наура (она же БНФ).

Сейчас у нас есть операция преобразования одного нетерминала в какую-то последовательность терминалов и нетерминалов. Но с последовательностями терминалов и нетерминалов мы пока ничего не умеем делать, надо все это дело формально определить.

Def 2.1.2. $\alpha B\gamma$, $\alpha, \gamma \in (\Sigma \cup N)^*, B \in N$.

Говорят, что $\alpha B\gamma \Rightarrow \alpha\beta\gamma$, если в P есть продукция $B \to \beta$, $\beta \in (\Sigma \cup N)^*$. Читается "$\alpha B\gamma$ выводит $\alpha\beta\gamma$"

Def 2.1.3. Если при этом $\alpha \in \Sigma^*$, то пишут $\alpha B\gamma \underset{lm}{\Longrightarrow} \alpha\beta\gamma$ (левосторонне выводит)

Аналогично, если $\gamma \in \Sigma^*$, то $\alpha B\gamma \underset{rm}{\Longrightarrow} \alpha\beta\gamma$ (правосторонне выводит)

Замечание 2.1.8. Левосторонне/правосторонне в том смысле что берем самый левый/правый нетерминал и его как-то раскрываем.

Def 2.1.4. $s \overset{*}{\Longrightarrow} t$, если $\exists s_0 \dots s_n \colon s_0 = s, s_n = t, \forall i \in [0, n] \colon s_i \Rightarrow s_{i+1}$.

Аналогично определяются $s \underset{lm}{\overset{*}{\Longrightarrow}}$ и $s \underset{rm}{\overset{*}{\Longrightarrow}}$

Def 2.1.5. $L(G) = \{w \in \Sigma^* : S \overset{*}{\Longrightarrow} w\}$ — КС-грамматика задает язык, состоящий из всех слов, которые можно вывести из стартового символа S.

Пример 2.1.1. $\Sigma = \{0, 1\}, N = \{S\}$
$P = \{S \to 0S0, S \to 1S1, S \to 0, S \to 1, S \to \varepsilon\}$

Еще одна договоренность: переходы, описанные выше, можно записать как $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$.

$S \Rightarrow 1S1 \Rightarrow 10S01 \Rightarrow 10101 \in \Sigma^*$

$L(G)$ — язык палиндромов над $\Sigma = \{0, 1\}$.

Формальными грамматиками, как видно, неплохо описываются всякие штуки, которые можно определить рекурсивно.

Замечание 2.1.9. Палиндромы не являются регулярным языком по лемме о накачке.

Def 2.1.6. Контекстно-свободным языком называется язык L такой, что $\exists$ КС-грамматика $G \colon L(G) = L$

Теорема 2.1.1. Регулярные языки $\subset$ контекстно-свободные языки.

▶ Рассмотрим регулярный язык L и ДКА $A \colon L(A) = L$

$A = (\Sigma, Q, q_0, T, \delta)$.

$G = (\Sigma, Q, q_0, P)$

Пусть были в состоянии A и перешли по символу c в состояние B. Тогда добавим продукцию $A \Rightarrow cB$.

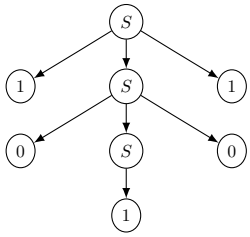
Теперь еще надо нетерминал иногда убивать. Для этого добавим еще продукции $A \Rightarrow \varepsilon \forall A \in T$.

Докажем, что это работает. Мы получаем терминал тогда и только тогда, когда у нас было слово из нескольких терминалов слева и символа, соответствующего терминальному состоянию справа. Это значит, что в ДКА мы прошли путь из стартового состояния в терминальное по полученному слову, значит, это слово принималось этим ДКА. ◀

Замечание 2.1.10. Из доказательства, в частности, следует, чтобы получить грамматики, равные регулярным языкам, надо сказать, что продукции могут в правой части иметь только выражения вида $aA \mid \varepsilon$

Пример 2.1.2. Вспомним пример с палиндромами. $S \Rightarrow 1S1 \Rightarrow 10S01 \Rightarrow 10101$.

Можно нарисовать дерево разбора этой штуки:



Для того, чтобы в результате разбора были только нетерминалы, необходимо и достаточно чтобы ими были все листья (крона дерева)

Def 2.1.7. Крона дерева с корнем в $v =$

- w, если v это лист с пометкой w.
- конкатенация крон $v_1 \dots v_n$, если v — не лист, есть дети $v_1 \dots v_n$.

Пример 2.1.3. Научимся парсить арифметические выражения, единиц и знаков «плюс»

$S \Rightarrow 1 \mid S + S$. Например, $S \Rightarrow S + S \Rightarrow S + S + S \Rightarrow 1 + S + S \Rightarrow 1 + S + 1 \Rightarrow 1 + 1 + 1$.

TODO картинки с деревьями разбора Заметим, что то же самое можно было получить другим порядком применения продукций. Например, $S \Rightarrow S + S \Rightarrow S + S + S \Rightarrow 1 + S + S \Rightarrow 1 + 1 + S \Rightarrow 1 + 1 + 1$

Def 2.1.8. КС-грамматика G неоднозначная, если $\exists w \in L(G)$: $\exists$ 2 различных вывода (дерева разбора).

Пример 2.1.4. «x * y» в языке C может означать и как «бинарный оператор умножения, примененный к x и y» и как «определение указателя с названием y на объект типа x».

Предыдущую грамматику можно поправить: $S \Rightarrow 1 \mid 1 + S$. Заодно сразу получим правую ассоциативность плюсика (до этого ее не было).

Замечание 2.1.11. Существуют существенно неоднозначнные КС-языки(то есть все грамматики неоднозначнные). Например, на практике доказывали, что язык $\{0^x 1^y 2^z\}$: $a = b \vee b = c$ существенно неоднозначен.

2.1.1. Преобразования КС-грамматик

Цель — как-то преобразовать грамматику так, чтобы потом можно было на нее удобно натравливать всякие алгоритмы.

1. Удаление ε-продукций: $A \to \varepsilon$. Хотим сделать так, чтобы ε вообще нельзя было получить.

   Def 2.1.9. Нетерминал A называется ε-порождающим, если $A \overset{*}{\Longrightarrow} \varepsilon$.

   Алгоритм определения ε-порождающих терминалов:

   - Если есть продукция $A \to \varepsilon$, то A — ε-порождающий. Все такие продукции просто выкинем из грамматики.
   - Если есть продукция $A \to A_1 \dots A_k$, где $\forall i \in [1; k] A_i$ — ε-порождающий, то A — ε-порождающий.

   Для каждой продукции $A \to A_1 \dots A_n$ для каждой S — подпоследовательности $[1; n]$, не совпадающей с $[1; n]$ такой, что A_{S_i} — ε-порождающие, сделаем новую продукцию: $A \to A_{t_1} \dots A_{t_k}$, где t_i пробегают все значения в $[1; n] \setminus S$.

Словами: в правой части продукции есть какие-то нетерминалы, посмотрим на те, из которых можно вывести в итоге ε. Если мы действительно выведем ε, то можно просто сказать, что у нас этот нетерминал просто и не появлялся. Таким образом, переберем те нетерминалы, которые в итоге просто исчезнут, выкинем их, допишем соответствующую продукцию без них.

Замечание 2.1.12. A_i здесь могут быть как и нетерминалами, так и терминалами

Замечание 2.1.13. Стоит понимать, что мы сейчас внезапно могли получить экспоненциальный рост количества продукций: если была продукция $A \to BCDE$, при этом B, C, E — ε-порождающие, то сейчас мы получили следующую продукцию:

$A \to BD \mid CD \mid D \mid BCD \mid DE \mid BDE \mid CDE \mid BCDE$. Чуть позже мы с этим поборемся.

Была грамматика G, получили грамматику G'. Тогда верно: $L(G') = L(G) \setminus \{\varepsilon\}$.

Просто аккуратно посмотрим на вывод, если мы какой-то нетерминал когда-то заменили на ε, то можно просто найти момент, когда мы его получили и сказать, что в тот момент мы использовали сокращенную продукцию без этого нетерминала. При этом могли получить где-то раньше в итоге ε, например, если все получившиеся из продукции нетерминалы превратились в ε. Тогда просто опять повторим эту же штуку, понятно, что если в итоге получали не пустое слово, то все будет хорошо.

Замечание 2.1.14. Если очень хочется получить тот же самый язык, что и был, то просто для S разрешают иметь продукцию $S \to \varepsilon$.

Еще можно ввести новый стартовый символ S', добавить продукции $S' \to \varepsilon \mid S$.

Оценка на размер G': $\mathcal{O}(|G|2^m)$, где m — максимально возможная длина правой части продукции. Выглядит страшно, но мы с этим в какой-то момент сможем побороться.

2. Удаление цепных продукций: $A \to B$, B — нетерминал. Рассмотрим транзитивное замыкание цепных продукций: добавим продукцию $A \to B$, если существуют $A_0 \dots A_n$: $A_0 = A, A_n = B, A_i \to A_{i+1} \in P, A_i \in N$.

   Теперь для каждой цепной продукции $A \to B$ из замыкания и для каждой продукции $B \to W$, не являющейся цепной (W может быть терминалом, или же последовательностью терминалов и нетерминалов), создадим продукцию $A \to W$. А все цепные продукции просто удалим.

   Пример 2.1.5. Были продукции
   $A \to B$
   $B \to C \mid b$
   $C \to B \mid AB \mid c$

   Тогда после удаления цепных продукций получим:
   $A \to b \mid AB \mid c$
   $B \to b \mid AB \mid c$
   $C \to AB \mid c \mid b$.

   Кстати, можно применить какую-нибудь эвристику вида “посмотрели теперь на продукции, увидели, что они для A и B совпадают, поэтому их можно склеить”.

   Размер G': $\mathcal{O}(|G||N|)$.

3. Удаление бесполезных символов (и терминалов и нетерминалов).

   Символ $T \in \Sigma \cup N$ называется полезным, если он учавствует в выводе какого-то слова, то есть, $S \overset{*}{\Longrightarrow} \alpha T \beta \overset{*}{\Longrightarrow} w \in \Sigma^*$. При этом, α, β могут быть пустыми.

Пример 2.1.6.
$S \to T$
$T \to t$
$S \to P$
$R \to r$

Символ T полезный, символы P, R — нет.

Соответственно, бесполезные символы, которые не встречаются в выводах, можно удалить.

Сначала для каждого нетерминала T проверим, что $\exists w \in \Sigma^* \colon T \overset{*}{\Longrightarrow} w$.

Если есть продукция $T \to w$, где $w \in \Sigma^*$, то говорим, что T — порождающий (то есть, можно вывести из него какое-то слово из терминалов) Терминальные символы формально тоже назовем порождающими.

$T \to W$, все нетерминалы в W порождающие, то T тоже порождающий.

Удалим теперь все непорождающие нетерминалы, и все продукции, где они встречаются: они нам никакой пользы, понятное дело, не дают, их не надо получать.

Теперь еще удалим символы, недостижимые из S.

Мы сейчас убрали все бесполезные символы. Оставшиеся символы полезные, так как каждый из них и достижим и порождающий, а это равносильно полезности. Все удаленные символы бесполезные, так как хотя бы одной из двух стрелок для них не существовало.

Размер $\mathcal{O}(|G|)$

2.1.2. Нормальная форма Хомского

Def 2.1.10. Про КС-грамматику G говорят, что она в нормальной форме Хомского, когда есть только продукции вида: $A \to BC$ (возможно, либо B либо C совпадает с A), при этом все символы — полезные и $A \to \alpha$, где α — терминал.

Замечание 2.1.15. В данном формализме нельзя получить пустое слово, можно опять просто сделать новый стартовый символ с продукциями $S' \to S \mid \varepsilon$

Алгоритм приведения грамматики в нормальную форму Хомского:

0. Удаление длинных правых частей. $A \to B_1 \dots B_k, k \geq 3$. ($B_i \in \Sigma \cup N$)

   Заменим это на
   $A \to B_1 C_1$
   $C_1 \to B_2 C_2$
   ...
   $C_{k-2} \to B_{k-1} B_k$

   C_i — какие-то новые нетерминалы, раньше их не было в грамматике.

1. Удаление ε-продукций. Заметим, кстати, что экспоненциального роста уже не будет — все продукции стали короткими.

2. Удаление цепных продукций.

3. Удаление бесполезных символов.

4. Остались только плохие продукции длины 2 вида $A \to aB, A \to ab$ (от длинных избавились, продукции вида $A \to B \in N$ исчезли на втором шаге).

   Продукцию $A \to aB$ просто превратим в
   $A \to XY$
   $X \to a$
   $Y \to B$

   X, Y — новые нетерминалы.

   Аналогично, $A \to ab$ заменяем на
   $A \to XY$
   $X \to a$
   $Y \to b$

   Можно еще просто сделать для каждого терминала соответствующий ему нетерминал, а не добавять каждый раз по два новых нетерминала.

Для нормальной формы Хомского уже можно написать какой-нибудь алгоритм, проверяющий слово на принадлежность грамматике

2.1.3. Проверка принадлежности слова КС-языку

Алгоритм Кока-Янгера-Касами (CYK)

У нас есть КС-грамматика G в нормальной форме Хомского, есть какое-то слово $w \in \Sigma^*$, хотим проверить, верно ли, что $w \in L(G)$. Пусть $|w| = n$. У нас будет трехмерная динамика d_{ijk}, $i \in [1; n], j \in [1; n], k \in N$ — верно ли, что $k \overset{*}{\Longrightarrow} w_{i\ldots j}$ (обратите внимание, k это не натуральное число, а нетерминал!)

1. $i = j$, мы хотим получить однобуквенное слово, $d_{ijk} = \text{true}$, если есть продукция $k \to w_i$.

2. $i \neq j$. Если реально получили из k $w_{i\ldots j}$, то мы k заменили на два нетерминала A, B При этом, A раскроется в префикс $w_{i\ldots j}$, а B — в суффикс.

   Тогда несложно посчитать d_{ijk}: надо просто перебрать m, A, B, где $k \to AB$, если $d_{i,m,A} \wedge d_{m+1,j,B}$, для какого-то m, то $d_{ijk} = \text{true}$.

   Время работы: $\mathcal{O}(n^3|G|)$ (цикл по A, B суммарно (суммируем по k) будет работать за $|G|$ для фиксированных i, j, m).

2.1.4. Лемма о накачке для КС-языков

Лемма о накачке для контекстно-свободных языков. Это будет снова сильное утверждение, которое позволит для некоторых языков доказать неконтекстно - свободность.

Лемма 2.1.1. Пусть L — контекстно-свободный язык.
Тогда есть такое натуральное $n \geqslant 1$, что для любого слова $w \in L$ длины хотя бы n существует его разбиение $w = uvxyz$ такое, что:

$$|v| + |x| + |y| \leqslant n \quad \text{«середина» достаточно короткая}$$
$$|vy| \neq \varepsilon \quad \text{«середина» отделена непустыми «скобочками»}$$
$$\forall i \geqslant 0 \colon uv^ixy^iz \in L \quad \text{«скобочки» можно повторять много раз или убрать}$$

Замечание 2.1.16. Бесконечность языка неважна: если язык конечен, просто возьмём $n = 1 + \max_{w \in L} |w|$ и получим квантор $\forall w$ по пустому множеству.

Замечание 2.1.17. Формулировка довольно похожа на формулировку леммы для регулярных языков, но отличаются детали.

▶ Раз L — контекстно-свободный, то есть контекстно-свободная грамматика G в нормальной форме Хомского, причём $L(G) = L \setminus \{\varepsilon\}$. Наличие или отсутствие пустого слова никак не влияет на лемму. Выберем в условии леммы $n = 2^{|N|}$, где N — множество нетерминальных символов в G. Теперь зафиксируем произвольное слово $w \in L$ такое, что $|w| \geqslant n$. Рассмотрим дерево разбора T слова w в грамматике G. Заметим, что так как $|N| \geqslant 1$, то $n \geqslant 2$, стало быть, корень T не может перейти в терминал сразу, он сначала перейдёт в два нетерминала.

Нам надо выяснить, как связана высота дерева разбора (число рёбер вдоль масимального пути в дереве от корня к листам) с числом листьев (т.е. с длиной слова).

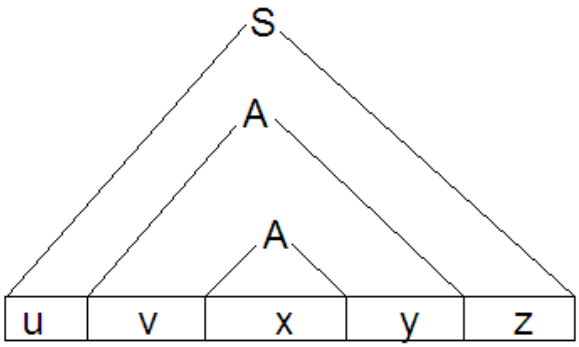
Утверждение 2.1.1. Если высота дерева равна h, то листьев не более 2^{h-1} (нам еще надо каждый нетерминал превратить в терминал).

▶ По индукции. База для $h = 1$ очевидна — мы сразу превращаем нетерминал в терминал, так как всего лишь одно ребро в дереве.

Теперь смотрим на корень. Его либо превратили в терминал сразу же, тогда крона состоит из одной вершины, либо его заменили на два нетерминала (потому что нормальная форма Хомского). При этом, этим нетерминалам соответствуют деревья высоты не более $h - 1$, применим предположение индукции, сложим, получим то, что надо. ◀

Так как у нас длина слова (то есть, крона), имеет размер $2^{|N|}$, то высота дерева хотя бы $|N|+1$. Рассмотрим путь до самой глубокой вершины, на нем есть хотя бы $|N| + 1$ нетерминальная. Значит, на этом пути есть хотя бы два одинаковых нетерминала (по принципу Дирихле), из всех пар повторяющихся выберем пару с наибольшей глубиной верхнего из них (то есть, самую близкую к листу).

Посмотрим на дерево разбора, оно естественным образом разбивается этими двумя вершинами на 5 частей, как на картинке:



Скажем, что эти 5 частей это в точности 5 частей из леммы и поймем что все сходится.

Мы знаем, что $A \overset{*}{\Longrightarrow} x$, $A \overset{*}{\Longrightarrow} vAy$, поэтому, можно записать $S \overset{*}{\Longrightarrow} uAz \overset{*}{\Longrightarrow} uvAyz \overset{*}{\Longrightarrow} ... \overset{*}{\Longrightarrow} uv^iAy^iz \overset{*}{\Longrightarrow} uv^ixy^iz$.

Так же заметим, что хоты бы одно из v и y не пусто. В верхнем A мы разделились на две части. В одной половине дошли до A, вторая половина это либо y, либо v и так как мы убрали ϵ-продукции, то эта строчка не пустая.

Осталось лишь понять, что $vxy \leq 2^{|N|}$. То есть, надо показать, что крона дерева с корнем в верхей вершине A имеют длину не более чем $2^{|N|}$.

Пусть это не так. Тогда есть путь от верхней A вниз длины хотя бы $|N| + 1$, там есть повтор, противоречие с тем, что мы взяли пару с самой глубокой верхней вершиной. ◀

Пример 2.1.7. $L = \{0^n1^n2^n\}$. Возьмем и с помощью леммы докажем, что он не КС-свободный.

Зафиксируем $n \in \mathbb{N}$, возьмем $w = 0^n1^n2^n$. Его как-то разбили на u, v, x, y, z. Заметим, что vy не может одновременно содержать и 0 и 2, так как иначе длина будет слишком большой. Возьмем $i = 2$. Не умаляя общности, будем считать, что 0 не вошел в vy. Тогда у нас нулей осталось n, а единичек и двоек у нас хотя бы $n + 2$, сломали.

Пример 2.1.8. Язык тандемов: $L = \{ww, w \in \Sigma^*\}$. Сломаем лемму.

Зафиксировали $n \in \mathbb{N}$, возьмем $w = 0^n1^n0^n1^n$. Нам это слово как-то разделили на u, v, x, y, z. Хотим предъявить какое-то i, которое все сломает. Есть два варианта: либо vxy внутри блока нулей/единиц, либо залез и туда и туда. В первом случае понятно, что можно несколько раз продублировать и просто количество нулей не совпадет с количеством единиц. Во втором случае тоже все хорошо: есть блок, который целиком не лежит в vxy, его длина не изменится, а длина остальных увеличится.

Следствие 2.1.1.1. $L_1 = \{0^n1^n2^m\}$, $L_2 = \{0^m1^n2^n\}$.

Несложно убедиться, что оба языка – КС-языки, просто написав для них соответствующие продукции. Для первого языка, например, можно записать:
$S \to A2$
$A \to A2$
$A \to B$
$B \to 0B1$
$B \to \varepsilon$

Но $L = L_1 \cap L_2 = \{0^n1^n2^n\}$ не является КС-языком.

То есть, пересекать КС-языки, в отличие от регулярных, не всегда получается. А вот объединять все равно можно.

2.2. Автоматы с магазинной памятью

2.2.1. Недетерминированные конечные автоматы с магазинной памятью

Замечание 2.2.1. Аббревиатура: МП-автомат (по умолчанию считаем, что недетерминированный). МПД-автомат — если детерминированный.

Замечание 2.2.2. Недетерминированный с ε-переходами.

Магазинная память — на самом деле стек. Магазин — имеется в виду магазин у огнестрельного оружия.

Def 2.2.1. МП-автомат: $A = (\Sigma, \Gamma, Q, q_0, T, z_0, \delta)$.

Σ — алфавит, Γ — стековый алфавит.

Q — множество состояний, $q_0 \in Q$ — стартовое состояние, $T \subset Q$ — множество терминальных состояний.

$z_0 \in \Gamma$ — нулевой символ

$\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ — функция перехода.

Замечание 2.2.3. Надо еще сказать, что у нас переходы в **конечное** подмножество $Q \times \Gamma^*$

По состоянию, символу в строке (или ε), а так же по вершине стека (ее мы при переходе сотрем) выдаем множество допустимых переходов. Переход — пара из нового состояния и строки, которую пишем на стек. Строка может быть пустой — просто достали символ со стека, если не хотим ничего со стека стирать — просто делаем строку длины 1 с символом, который только что стерли.

Замечание 2.2.4. Обычно стек записывают в виде строчки, при этом считают, что вершина стека находится слева.

Замечание 2.2.5. При этом, когда есть стек def и говорим, что кладем на него слово abc, в итоге мы получим стек $abcdef$, то есть, мы буквы кладем в таком порядке: c, b, a.

Def 2.2.2. Моментальное описание — тройка (q, w, s), где q – текущее состояние, w — непрочитанная часть входного слова, s — содержимое стека

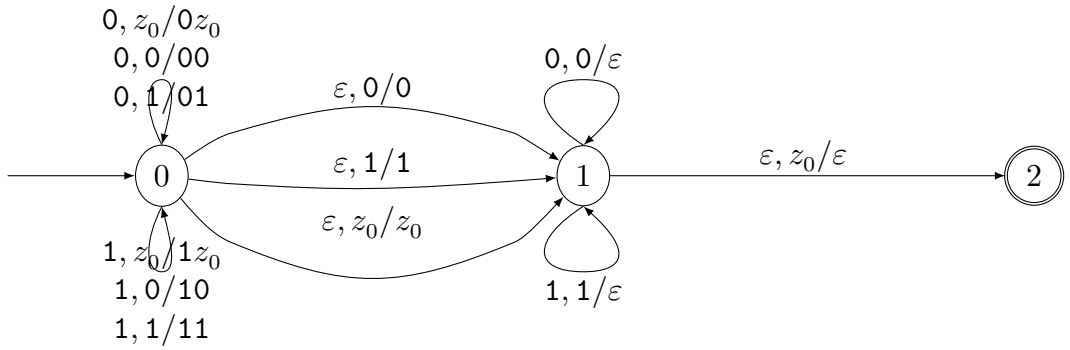
Def 2.2.3. Рассмотрим два МО, имеющих следующий вид: (q, aw, is) и $(p, w, \alpha s)$, $\alpha \in \Gamma^*, a \in (\Sigma \cup \{\varepsilon\})$. Говорят, что $(q, aw, is) \vdash (p, w, \alpha s)$ (читается «состояние переходит за один шаг в», если в δ есть элемент $(q, a, i) \to (p, \alpha)$.

Def 2.2.4. Аналогично, определяют переход за несколько шагов. Пусть есть два МО M_1, M_2, тогда говорят, что $M_1 \vdash^* M_2$ если есть последовательность МО $N_1 \dots N_k$ такая что : $M_1 \vdash N_1, N_1 \vdash N_2 \dots N_{k-1} \vdash N_k, N_k \vdash M_2$.

Def 2.2.5. МП-автомат A принимает слово w по терминальному состоянию, если $\exists q \in T, s \in \Gamma^*$: $(q_0, w, z_0) \vdash^* (q, \varepsilon, s)$

Когда рисуют такой автомат, на ребре пишут $a, i/\alpha$ — по символу a и символу i на вершине стека перешли куда-то, на стек записали α.

Пример 2.2.1. $L = \{ww^R : w \in \Sigma^*\}$.
$\Sigma = \{0, 1\}, \Gamma = \{0, 1, z_0\}$.



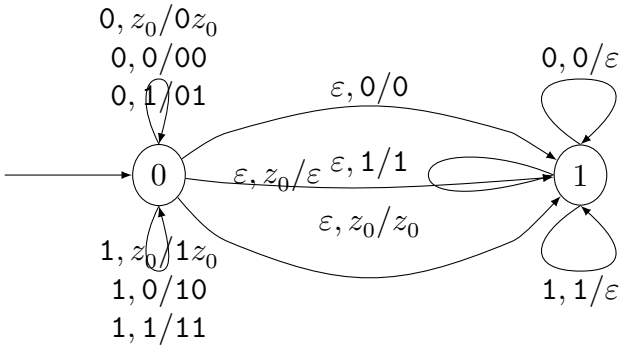
Выглядит ужасно и формально, ничего не поделаешь. Словами: мы хотим сначала все пихать на стек, потом доставать с него, при этом требование — чтобы то что достаем со стека совпадало с тем, что сейчас в строке.

Недетерминированность тут явно нужна, так как непонятно, в какой момент перестать класть на стек.

Def 2.2.6. МП-автомат принимает слово w по пустому магазину, если $\exists q \in Q$: $(q_0, w, z_0) \vdash^* (q, \varepsilon, \varepsilon)$.

Замечание 2.2.6. В таком способе принятия из определения исключают T, так как не нужно.

Пример 2.2.2. $L = \{ww^R : w \in \Sigma^*\}$.
$\Sigma = \{0, 1\}, \Gamma = \{0, 1, z_0\}$.



Вместо перехода в терминальное состояние добавили стирание нулевого символа со стека.

Утверждение 2.2.1. Принятие по терминальному состоянию позволяет принимать в точности те же языки, что и принятие по пустому магазину.

▶ Возьмем МП-автомат A, принимающий по терминальному состоянию.

В новом автомате введем новое стартовое состояние q_0' и еще один нулевой стековый символ z_1 и добавим переход из q_0', пишем на стек z_0z_1. Это нужно потому, что, вообще говоря, мы могли случайно в исходном автомате когда-то совсем-совсем затереть стек, а сейчас мы сделали так, чтобы мы при этом не завершались — у нас просто нет переходов, на которых написано, что снимаем со стека z_1.

А из терминальных состояний надо добавить переход $\varepsilon*/\varepsilon$ в новое состояние, переход ничего не будет добавлять, новое состояние зациклим на себя переходом $\varepsilon*/\varepsilon$, чтобы стек опустошить.

В другую сторону. Взяли МП-автомат, принимающий по пустому стеку. Хочется просто сказать, что если стек стал пустым, то надо перейти в терминальное состояние. Но так делать нельзя, так как для перехода нам нужно что-то со стека прочитать.

Для этого опять добавим новое стартовое состояние и символ z_1, а из каждого состояния добавим переход $\varepsilon, z_1/\varepsilon$ в терминальное состояние. ◀

Теорема 2.2.1. Языки МП-автоматов = КС-языки

▶ **МП-автомат $A \to$ КС-грамматика G** Будем считать, что автомат принимал по пустому стеку (мы уже показывали, что не важно, как он принимает, но так нам будет удобнее).

Для каждой буквы посмотрим на момент, когда мы ее положили на стек, и когда ее сняли с него. Рассмотрим слово, которое нам «помогло» ее снять, то есть, на подстроку w, которую мы успели принять между этими двумя состояниями. Получаем какой-то язык слов, которые могут перевести автомат из одного состояния в другое, сняв ровно одну букву со стека.

Зададим множество нетерминалов $N = Q \times \Gamma \times Q$. Будем обозначать нетерминал как $[qgp]$, это соответствует следующей тройке: (из какого состояния перешли, какой символ сняли со стека, в какое состояние перешли). Сделаем так, чтобы этот нетерминал выводил ровно те слова, которые могут перевести автомат A из q в p, снимая со стека g.

Для каждого состояния q рассмотрим все переходы из него вида $a, g/u$, где $u = u_1 \dots u_n$ — какое-то слово, которое мы кладем на стек, в состояние p. Для каждой последовательности $r_1 \dots r_n, r_i \in Q$ создадим продукцию $[qgr_n] \to a[pu_1r_1][r_1u_2r_2] \dots [r_{n-1}u_nr_n]$.

Осталось добавить стартовый символ S. Будут продукции вида $S \to [q_0z_0q_i] \forall q_i \in Q$ — S соответствует ровно всем тем словам, которые берут и снимают со стека z_0, оставляя его пустым.

Формально покажем, что принимаем ровно те слова, что надо. Просто индукция по построению не проходит — так как у нас продукции рекурсивные, могут циклиться. Но можно делать индукцию по количеству переходов в автомате при приеме слова, тогда все получится. Докажем, что любое слово, принимаемое автоматом, выводится из S. База понятна, если переход только один, то длина слова не больше единицы, при этом, каждый нетерминал гарантированно породит хотя бы один терминал. Поэтому, было только применение продукции $S \to a \in \Sigma$.

Переход. Смотрим на самый первый переход из стартового состояния в автомате. Мы перешли в строке по символу a, сняли z_0, добавили на стек какое-то слово $u = u_1 \dots u_k$. Каждый из этих символов мы в какой-то момент со стека снимем, есть последовательность состояний $r_1 \dots r_k$, в которых это случится. Тогда, по построению, есть продукция $S \to a[q_0z_0r_k] \to ac[q_0u_1r_1] \dots [r_{k-1}u_kr_k]$. Рассмотрим теперь переход из r_i в r_{i+1}, заметим, что его длина будет меньше количества переходов для приема всего слова, поэтому,

по предположению индукции, соответствующий ему нетерминал действительно выводит нужное слово.

Аналогично докажем, что не принимаем ничего лишнего — то есть, любое слово, которое можно вывести из S, можно принять и автоматом A. Только тут будет уже индукция по размеру вывода в КС-грамматике, делаем абсолютно тот же самый формализм.

Стоит заметить, что, вообще говоря, получили экспоненциальную грамматику (от размера u). Можно этого избежать, надо просто сказать, что у нас на стек будут добавляться короткие слова. Модифицируем исходный МП-автомат, вот у нас был переход из p в q, где мы на стек клали какое-то слово $w_1 \dots w_n$, снимая символ c. Уберем этот переход, добавим новые уникальные символы $c_1, \dots c_{n-1}$ в алфавит и новые состояния $q_1 \dots q_{n-1}$ со следующими переходами: из p идем в q_1, снимаем c, кладем $w_1 c_1$ (на вершине стека будет c_1). Теперь из q_i делаем переход в q_{i+1}, снимаем c_i, кладем $w_{i+1} c_{i+1}$. И из q_{n-1} переходим в q, снимаем c_{n-1}, кладем w_n.

КС-грамматика $G \to$ МП-автомат A Состояние у нас будет только одно, назовем его q.

Зададим стековый алфавит: $\Gamma = N \cup \Sigma \cup \{z_0\}$, а просто алфавит будет совпадать с множеством терминалов.

Функция перехода выглядит так: для каждой продукции $A \to \gamma \in (\Sigma \cup N)^*$ зададим следующий переход в автомате: $\delta(q, \varepsilon, A) = \{q, \gamma\}$ — просто взяли и положили на стек правую часть вывода (при этом его левый символ — вершина стека). И для терминалов еще добавим следующие переходы: $\delta(q, a, a) = \{q, \varepsilon\}$ — просто снимаем терминал со стека, если именно он сейчас есть в принимаемом слове. Чтобы у нас хоть что-то принялось (мы будем принимать по пустому стеку) надо, чтобы можно было его опустошить — добавим переход $\delta(q, \varepsilon, z_0) = \{q, \varepsilon\}$.

Интуитивное понимание того, что эта конструкция работает: есть какой-то левосторонний вывод (каждый раз берем и применяем продукцию к самому левому нетерминалу). Скажем, что у нас в каждый момент есть какая-то строка из терминалов и нетерминалов, мы по ней идем слева направо, терминалы выкидываем, запоминая, что они лежат в слове, как только доходим до нетерминала — применяем к нему какую-либо продукцию.

А это в точности то, что мы делаем со стеком — если мы применяем продукцию, то мы слева дописываем раскрытие нетерминала, а если мы выкидываем терминал, то мы проверяем, что этот символ действительно сейчас должен быть в слове.

Можно все это дело доказать абсолютно формально так же, как и в предыдущей части, индукцией по длине левостороннего вывода и индукцией по количеству переходов в автомате.

Пример 2.2.3. Пример для пояснения. Был такой вывод: $S \to 0A1 \to 00A1 \to 0011$ (соответственно, были продукции $S \to 0A1, A \to 0A|1$). В нашей конструкции происходит следующее.

- Сначала на стеке лежит только S, после этого кладем туда $0A1$, в слове переходим по ε.
- Снимаем со стека 0, переходя по этому же символу в строке (то есть, говорим, что первый символ слова, которое сейчас принимаем, равен нулю).
- Теперь снимаем со стека A, и кладем на стек $0A$, потому что мы применили такую продукцию, в слове переходим по ε, в итоге на стеке лежит $0A1$.
- Потом опять снимаем 0, в слове по нулю переходим.
- Снимаем A, переходим по ε, кладем на стек 1.
- На стеке лежит ровно две единички, снимаем их, в слове переходим по двум единичкам.
- В итоге стек опустел, закончили принимать. Приняли слово 0011, что и требовалось.

◀

Следствие 2.2.1.1. Получили, что любой МП-автомат можно переделать в МП-автомат с одним состоянием (принимающией по пустому стеку).

2.2.2. Детерминированные конечные автоматы с магазинной памятью (ДМП)

Когда у нас были ДКА и НКА, то мы умели детерминизировать НКА — там было не очень много допустимых конфигураций, как минимум, конечное число. А тут, к сожалению, такого нет — у нас конфигурация теперь представляет собой пару (состояние в автомате, строка на стеке). А на стеке у нас, в принципе, может быть строка абсолютно любого размера, то есть, конфигураций может быть бесконечное количество.

Поскольку ничего лучше какого-либо экспоненциального перебора не придумать, то имеет смысл рассматривать еще и детерминированные автоматы с магазинной памятью. К сожалению, как мы покажем, они принимают несколько меньший класс языков.

Def 2.2.7. МП-автомат называется детерминированным, если

- В δ для тройки $(q \in Q, a \in \Sigma \cup \{\varepsilon\}, g \in \Gamma)$ есть переход ровно в одну пару $(p \in Q, \gamma \in \Gamma^*)$. $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$.
- Если есть переход $(q, a \in \Sigma, g) \to ()$, то нет перехода $(q, \varepsilon, g) \to ()$ — либо есть переходы по обычным символам, или только по ε.

Def 2.2.8. L — префиксный язык, если $\nexists w_1, w_2 \in L$: w_1 — префикс w_2.

В отличие от МП-автоматов, у ДМП-автоматов прием по пустому стеку и по терминальному состоянию это не одно и то же, сейчас это покажем.

Теорема 2.2.2. Язык L принимается ДМП автоматом по пустому стеку $\Leftrightarrow$ L - префиксный и принимается ДМП-автоматом по терминальному состоянию

▶ $\Rightarrow$ Пусть есть слово $w \in L$, при этом еще существует непустое слово u такое, что $wu \in L$. Получаем противоречие, так как, когда мы примем префикс w слова wu, у нас будет пустой стек и мы закончим работу. Из всех состояний нашего автомата надо добавить переход $\varepsilon, z_1/z_1$ в новое, терминальное, состояние. Тут мы, как и на прошлой лекции, еще добавили новое стартовое состояние и новый символ z_1, из нового начального состояния в старое есть переход $\varepsilon, z_0/z_0z_1$.

$\Leftarrow$ A — ДМП-автомат, принимающий по терминальному состоянию префиксный язык $L(A)$. Это значит, что при приеме любого слова мы только один раз бываем в терминальном состоянии — иначе можно принять еще и его префикс. На самом деле, это не правда, можем прийти в терминальное состояние, потом сделать ε-переход, попасть снова в терминальное состояние. Но для дальнейшего перехода это совсем не важно.

Все переходы, которые были из терминальных состояний, удаляем (потому что они при приеме слова из языка все равно не использовались), добавляем петли $\varepsilon, \forall/\varepsilon$

Есть еще следующая проблема: могли в исходном автомате слово не принять, но попасть в какое-то состояние, имея пустой стек. Чтобы это слово не принялось в новом автомате, нужно, как обычно, чтобы было «двойное дно» — опять новое начальное состояние, символ z_1 в самом низу стека, который снимется в самом конце.

◀

Теорема 2.2.3. Языки, принимаемые ДМП-автоматами по терминальным состояниям, строго вложены в КС-языки, а регулярные строго вложены в ДМП-автоматы, принимающие по терминальным состояниям.

▶ Нестрогие вложенности очевидны: к ДКА для регулярного языка легко приделать стек — просто убираем z_0 и кладем его обратно. Вторая нестрогая вложенность тоже очевидна — ДМП это частный случай МП, которые в точности равны КС-языкам.

Надо теперь доказать, что классы не равны.

Регулярные $\neq$ ДМП-автоматам $L = \{0^n 10^n\}$ Этот язык не регулярный. Для доказательства используем лемму о накачке. Зафиксировали $n \in \mathbb{N}$, возьмем слово $w = 0^n 10^n$, для любого его подходящего разбиения на три части лемма о накачке ломается.
А вот ДМП-автомат для этого языка есть — просто кладем нолики на стек, пока не придет единичка, а потом достаем их обратно. Кончился стек — переходим в терминальное состояние.

ДМП-автоматы $\neq$ КС-языки Для неформального доказательства неравенства можно взять язык $ww^R, w \in \Sigma^*$. Говорим, что сначала мы должны класть на стек, а потом доставать, чтобы сравнивать с тем, что лежит в слове. Но при этом не очень понятно, в какой момент, собственно, прекращать все класть на стек. И как это формализовать — не очень понятно.
Для формального доказательства рассмотрим язык $L = \{0^n 1^n\} \cup \{0^n 1^{2n}\}$. Покажем, что этот язык не принадлежит ДМП-автоматам, принимающим по терминальному состоянию. Делать это будем от противного, предположим, что нашелся такой автомат.
У него есть терминальные состояния, принимающие $0^n 1^n$ и $0^n 1^{2n}$. Сделаем копию этого автомата (убирая начальность у начального состояния), добавим ε-переходы из терминальных состояний исходного в соответствующие терминальные состояния копии. Во всех тех местах, где в копии в переходах есть единички, заменим их на двойки. **TODO** картинка!
Докажем, что этот автомат будет принимать язык $\{0^n 1^n\} \cup \{0^n 1^{2n}\} \cup \{0^n 1^n 2^n\}$. Интуитивно это кажется более-менее понятным (как только появится картинка), но формально это надо объяснить.
Пусть w принимается автоматом. Если не было ε-перехода в копию, то оно принимается исходным автоматом, то есть, лежит в $\{0^n 1^n\} \cup \{0^n 1^{2n}\}\}$.
Если ε-переход был, то уберем его, перенесем все переходы из копии в исходный автомат (в смысле что если шли по копии какого-то ребра, то скажем, что пойдем по его оригиналу), получили какое-то слово u изноликов и единичек. Был какой-то префикс p слова, который мы приняли перед тем, как был ε-переход в копию. Этот префикс равен либо $0^n 1^n$, или $0^n 1^{2n}$ (потому что у нас переходы в копию только из терминальных состояний). В первом случае u это либо $0^n 1^n$, тогда $w = 0^n 1^n$, либо $u = 0^n 1^{2n}$, тогда $w = 0^n 1^n 2^n$. Во втором случае $u = 0^n 1^{2n}, w = 0^n 1^2 n$.
Это мы формально показали, что принимается не более чем этот язык. Несложно понять, что каждое слово из этого языка принимается (из построения).
Заметим, что $\{0^n 1^n 2^n\}$ не КС-язык (а, значит, и $\{0^n 1^n\} \cup \{0^n 1^{2n}\} \cup \{0^n 1^n 2^n\}$ тоже) Пользуемся леммой о накачке. Зафиксировали n, взяли слово $0^n 1^n 2^n$. Центральная часть разбиения не могла одновременно захватить все три цифры, поэтому, если мы ее повторим несколько раз, то получим слово не из языка, противоречие.
Итого, мы взяли ДМП-автомат, получили, что он принимает то, что не принимает КС-язык, при этом мы знаем, что есть вложенность. Противоречие

◀

2.3. Иерархия формальных грамматик Хомского

0. Неограниченныые грамматики

1. Контекстно-зависимые
2. Контекстно-свободные
3. Автоматные грамматики (регулярные)

Вложенность снизу вверх (нулевой класс самый большой).
Этим грамматикам соответствуют следующие вычислители:

0. Машина Тьюринга
1. Линейно ограниченная машина Тьюринга (памяти столько же, сколько длина входа)
2. МП-автоматы
3. ДКА

Def 2.3.1. Регулярные грамматики — леволинейные грамматики $\cup$ праволинейные грамматики.
Леволинейные — только продукции вида $A \to a, A \to Ba$. Праволинейные — $A \to a, A \to aB$.
(А еще если хотим, чтобы принималось пустое слово, сделаем, как делали в нормальной форме Хомского, продукцию $S \to \varepsilon | S'$.

Замечание 2.3.1. Если разрешить и леволинейные и праволинейные одновременно, то получим что-то между КС-грамматиками и регулярными.

Теорема 2.3.1. Языки регулярных грамматик равны регулярным языкам

▶ **ДКА** $A \to$ **регулярная праволинейная грамматика** G $N = Q$ $S = q_0$ Для перехода из p в q по символу a добавим продукцию $p \to aq$
Если q_0 — терминальное, то создаем стартовый символ S, из него делаем продукцию $S \to \varepsilon | q_0$
Если пришли из состояния p в состояние $t \in T$ по символу a, то делаем продукцию $p \to a$
$\hat{\delta}(q_0, w) = p \Leftrightarrow$ в G: $S \overset{*}{\Longrightarrow} wP, w \notin L(a), s \overset{*}{\Longrightarrow} w, w \in L(A)$

Регулярная грамматика $G \to$ **НКА** A НУО, будем считать, что грамматика леволинейная. Для праволинейной будет все то же самое.
$Q = N \cup \{Z\}$, Z — специальное состояние, означающее, что слово принято, что больше нет нетерминалов. $q_0 = S$ $T = \{Z\}$ Переходы: был переход $A \to aB$, в автомате рисуем переход из состояния A в состояние B по букве a. А для перехода $A \to a$ делаем переход из A по букве a в Z.
$\hat{\delta}(q_0, w) = p \Rightarrow$ в G: $S \overset{*}{\Longrightarrow} wP, p \neq Z, S \overset{*}{\Longrightarrow} w, p = Z$. доказываем простой индукцией.

◀

Def 2.3.2. Контекстно-зависимые грамматики (Context-sensitive)
Определение почти такое же, как у КС-грамматик, но только другие ограничения на продукции: $\alpha B \gamma \to \alpha \beta \gamma, \alpha, \gamma \in (\Sigma \cup N)^*, B \in N, \beta \in (\Sigma \cup N)^+$.
Тут мы опять не можем получить пустое слово (так как β содержит хотя бы один символ), можно добавить как обычно новое стартовое состояние и продукцию из него в ε.

Замечание 2.3.2. Контекстно-зависимые в смысле, что нетерминал можно преобразовывать только в случае, если он есть в определенном контексте — слева и справа от него стоит то, что надо.

Def 2.3.3. Неукорачивающие грамматики — если была продукция $\alpha \to \beta$, то $|\alpha| \leq |\beta|$

Теорема 2.3.2. Языки КЗ-грамматик равны языкам неукорачивающих грамматик

▶ **КЗ → неукорачивающие** .

Тривиально, так как все продукции в КЗ неукорачивающие.

неукорачивающие → КЗ .

Для каждого терминала a создадим новый соответствующий нетерминал A, во всех продукциях заменяем все вхождения a на A, добавляем продукцию $A \to a$.
Для продукции $x_1 \dots x_n \to y_1 \dots y_m, m \geq n, x_i, y_i \in (N \cup \Sigma)$ создадим уникальные для этой продукции нетерминалы $Z_1 \dots Z_n$, добавим следующие продукции:
$X_1X_2 \dots X_n \to Z_1X_2 \dots X_n$
$Z_1X_2 \dots X_n \to Z_1Z_2x_3 \dots X_n$
$Z_1 \dots Z_{n-1}X_n \to Z_1 \dots Z_n$
$Z_1 \dots Z_n \to Y_1Z_2 \dots Z_n$
...
$Y_1 \dots Y_{n-1}Z_n \to Y_1 \dots Y_{n-1}Y_n \dots Y_m$
После применения первой продукции единственный способ избавиться от Z_i, в силу их уникальности, только применить все остальные продукции, тем самым переведя $X_1 \dots X_n$ в $Y_1 \dots Y_m$

◀

Замечание 2.3.3. Без первого шага сведение некорректное: если все x_i — терминалы, то продукция $x_1x_2 \dots x_n \to Z_1x_2 \dots x_n$ не является корректной продукцией для КЗ-грамматики.

𝔇𝔢𝔣 2.3.4. Нормальная форма Куроды (для КЗ-грамматик)

Грамматика G в нормальной формы Куроды, если все продукции имеют следующий вид: $AB \to CD, A \to CD, A \to a$. Иногда разрешают еще $A \to B$, но от этого можно избавиться (удаление цепных продукций как в нормальной форме Хомского)

Замечание 2.3.4. Важный факт на будущее: можно разрешить продукции $A \to \varepsilon$, получим нормальную форму Куроды для произвольных грамматик.

Замечание 2.3.5. Есть небольшая путаница в обозначениях, будьте внимательны. Лучше уточнять, какая именно форма имеется в виду.

Теорема 2.3.3. Любая КЗ-грамматика G представима в виде НФ Куроды

▶ Будем разрешать продукции $A \to B$.

1. Из правых частей исключим терминалы. Для любого символа $a \in \Sigma$ создадим нетерминал A и продукцию $A \to a$ Во всех продукциях $A \to \alpha$, кроме созданных, в α заменяем любой терминал на соответствующий нетерминал
2. Остались только продукции $A \to a$ и $\alpha \to \beta \in N^+, |\alpha| \leq |\beta|$.
   Продукции $A \to a$ не трогаем, они и так подходят под нормальную форму. Рассмотрим остальные продукции $\alpha \to \beta$. Разберем случаи
   - $|\beta| \leq 2$. Уже в нормальной форме Куроды (с цепными продукциями)
   - $|\beta| \geq 3, |\alpha| = 1$. Тогда это продукция $A \to B_1 \dots B_n$. Создадим для этой продукции уникальные нетерминалы $A_1 \dots A_{n-2}$, заменим исходную продукцию на следующие:
     $A \to B_1A_1$
     $A_1 \to B_2A_2$
     ...
     $A_{n-2} \to B_{n-1}B_n$
   - $|\beta| \geq 3, |\alpha| \geq 2$. Эта продукция имеет вид $A_1A_2 \dots A_n \to B_1B_2B_3 \dots B_m$. Запишем
     $A_1A_2 \to B_1C$
     $CA_3 \dots A_n \to B_2 \dots B_n$. Второе правило можно раскрывать далее либо по второму, либо по третьему пункту (длина и левой и правой части уменьшилась на 1).

◀

Def 2.3.5. Произвольная грамматика — грамматика, у которой единственное ограничение на продукции — слева стоит хотя бы один нетерминал.

Теорема 2.3.4. Любая формальная грамматика представима в виде НФ Куроды для произвольной грамматики

▶ Пропало правило про то, что продукции могли быть неукорачивающими.

Дана произвольная грамматика G. Добавим нетерминал Z и единственную продукцию $Z \to \varepsilon$. Для любой продукции $\alpha \to \beta, |\alpha| = n, |\beta| = m$. В случае, когда $n > m$ заменим ее на $\alpha \to \beta Z ... Z$ (повторим Z $n - m$ раз) ◀

Теорема 2.3.5. Языки неограниченных грамматик совпадают со всеми перечислимыми языками.

▶ Не будет строгого доказательства, все «на пальцах»

Произвольная грамматика $G \to$ НМТ T Процесс такой: на ленте записана текущая строка из терминалов и нетерминалов. Мы можем выбрать какую-то подстроку, применить к ней какую-то продукцию. Для этого надо проверить, что подстрока соответствует левой частми продукции, после чего сделать замену, подвинув правую часть ленты вправо/влево, если правая часть продукции короче/длиннее.

Поскольку машина будет недетерминированной, можно будет применить нужную продукцию, чтобы получить в итоге нужное слово.

МТ $\to$ произвольная грамматика Пускай есть МТ, принимающая на вход слово, и говорящая YES/NO.

Есть конфигурация МТ: слово на ленте, положение головки, текущее состояние. Храним мы ее так: у нас в каждый момент времени записана какая-то строка из нетерминалов, каждый нетерминал это либо пара (символ на ленте, -1), либо (символ на ленте, состояние). Пара второго типа будет встречаться в единственном экземпляре и будет означать, что головка сейчас находится над данным символом, при этом МТ имеет ровно такое вот состояние.

Соответственно, переходы у МТ описываются легко: мы просто берем, и делаем продукции из пары (символ на ленте, состояние) плюс еще по символу справа/слева в новую конфигурацию этих трех символов — возможно, подвинулась головка, то есть, пара второго типа оказалась правее/левее, еще могло измениться состояние и измениться символ под головкой.

Научились МТ эмулировать, хотим, чтобы, если мы пришли в YES, то в итоге было выписано изначальное слово, которое было записано на ленте.

Построим грамматику для следующего языка: $\{\wedge w \# w\}$, вот она:

$S \to \wedge S_1 \#$

$S_1 \# \to AS_1 \# a \mid BS_1 \# b \mid \varepsilon$

$aA \to Aa$

$bA \to Ab$

$aB \to Ba$

$bB \to Bb$

$\wedge A \to \wedge a$

$\wedge B \to \wedge b.$

Почему работает: берем, генерим какой-то нетерминал, во вторую копию w в начало пихаем соответствующую букву, а сам нетерминал мы умеем двигать влево. При этом, мы не умеем менять местами два нетерминала, а превратить его в терминал сможем только тогда, когда достигнем начала слова.

Вернемся к МТ. Возьмем, изначальную конфигурацию продублируем, получим $\wedge w\#w$. Над исходной строкой (первая копия), описывающей состояние, проведем эмуляцию МТ, в случае, если пришли в YES, затрем все что слева от #, получим исходное состояние (справа от # ничего не трогали), переведем его в строчку из терминалов. Если перешли в NO, то ничего не делаем, вывод не заканчиваем и все хорошо. Затирать можно, например, так
$\gamma A\# \to A\#, \gamma \in (N \cup \Sigma)$
$\wedge\# \to \varepsilon$
Хотим показать, что для МТ есть грамматика, которая порождает ровно тот же язык, который принимает МТ. Для этого пишем сначала грамматику, генерирующую удвоенные слова (только терминалы заменим на нетерминалы, чтобы с ними точно можно было дальше работать). При этом, еще скажем, что последний символ, который эта грамматика сгенерирует (самый левый), будет еще говорить, что на него указывает головка (мы говорили выше про то как выглядят состояния МТ).
После этого добавим продукции, описывающие работу МТ, и продукции, которые по приходу в YES/NO делают то, что надо.

◀

Замечание 2.3.6. Кстати, это почти то же самое, что и задача ассоциативного исчисления, которая была в прошлом семестре на логике. Она неразрешима, и тут мы тоже не можем сказать, принадлежит ли слово произвольной грамматике: никак нельзя понять, МТ еще работает, но не готова нам выдать ответ, или она зависла.

2.4. Лемма Огдена

Теорема 2.4.1 (Лемма Огдена (обобщенная лемма о накачке для КС-языков))**.** Пусть Γ — КС-грамматика. Тогда $\exists n \in \mathbb{N}$ такое, что $\forall w \in L(\Gamma)$: $|w| \geq n$, при этом, если выделить хотя бы n позиций в w, будет верно, что $\exists u, v, x, y, z$ такие, что x содержит выделенную позицию и одно из двух верно: u, v содержат выделенную позицию или y, z содержат ее vxy содержит не более чем n выделенных позиций, и при этом $uv^ixy^iz \in L(\Gamma)$

▶ Рассмотрим все продукции, посмотрим на их правые части, длину самой большой из них обозначим за l. Крайний случай: если $l = 1$, скажем на всякий случай, что $l = 2$.

Возьмем $m = |N|, n = l^{2m+3}$.

Берем произвольное слово w, можем построить дерево разбора, соответствующее тому, как мы из S получили w.

Рассмотрим следующий путь из корня в этом дереве: начинаем с u_1 — корень, а дальше по индукции для любого $i \geq 2$ выбираем u_i так: смотрим на детей u_{i-1}, выбираем того, у которого в поддереве наибольшее количество выделенных позиций. В итоге жадно получили какой-то путь $u_1 \dots u_p$ (u_p — лист).

Детей у каждой позиции не более чем l, смотрим на количество выделенных позиций у u_{i-1}, смотрим на количество выделенных позиций, у u_i, понимаем, что это количество уменьшилось не более чем в l раз (мы шли в сына с наибольшим числом выделенных вершин, если их количество уменьшилось более чем в l раз, то просто не сойдется сумма и будет противоречие). У корня было хотя бы l^{2m+3} выделенных позиций, тогда у i-й вершины их хотя бы l^{2m+4-i}. Откуда получаем, что $p > 2m + 3$: u_p — лист, у него в поддереве не более одной выделенной вершины (он сам), если p маленькое, то их должно быть больше.

Замечание 2.4.1. Тут таки важно, что $l \geq 2$, как мы потребовали в начале.

Рассмотрим все вершины на нашем пути, посмотрели на u_i и его ребенка u_{i+1}. Посмотрим на всех детей u_i, которые левее, чем u_{i+1}. Если для какого-то такого ребенка верно, что у него в

поддереве хотя бы одна выделенная позиция, то назовем u_i левоветвящейся. Если то же самое верно для правых детей — назовем правоветвящейся. Ветвящаяся вершина — либо лево-, либо правоветвящаяся (либо одновременно и то и то)

Рассмотрим последние $2m+3$ ветвящиеся вершины, тогда верно (принцип Дирихле), что среди них либо количество левоветвящихся хотя бы $m+2$, либо правоветвящихся хотя бы $m+2$. Ветвящихся вершин хотя бы столько, так как количество выделенных вершин в поддереве среди u_i уменьшалось хотя бы $m+3$ раза, как было показано выше. То есть, какие-то вершины оказывались левее/правее u_{i+1}.

НУО, левоветвящихся хотя бы $m+2$. Обозначим $v_1 \dots v_{m+2}$ — последние левоветвящиеся вершины из пути $u_1 \dots u_p$. По принципу Дирихле есть $i < j$ такие, что нетерминалы, записанные в вершинах v_i, v_j совпадают. **TODO** Картинка (похожа на просто лемму о накачке).

Среди всех таких пар (i, j) выбрали пару с максимальным i. Обозначили за x поддерево v_j, v, y — левая и правая части поддерева v_i (выкинули поддерево v_j). u, z — остальное. Заметим, что у x хотя бы одна выделенная позиция есть (по построению, u_p лежит в x).

Так как у нас v_i, v_j — левоветвящиеся, то у v_i есть ребенок слева от пути, у которого есть выделенные позиции. Тогда в v есть выделенные позиции.

Так как в пути хотя бы $m+2$ вершины, то $i > 1$ (выкидываем v_1 и принцип Дирихле все равно работает). Тогда над v_i тоже есть хотя бы одна левоветвящаяся вершина, значит, в u тоже есть выделенная позиция.

Аналогично, если бы у нас было бы много правоветвящихся вершин, то было бы верно, что в y, z есть выделенные позиции.

Теперь хотим понять, почему vxy содержит не более чем n выделенных позиций. Заметим, что после v_i у нас было не более чем $2m+3$ ветвящихся вершины (потому что мы брали только последние $2m+3$ ветвящихся)

Поскольку каждый раз у нас количеств выделенных позиций уменьшалось не более чем в l раз, при этом, оно могло уменьшаться только в ветвящихся вершинах (если вершина не ветвящаяся, то все выделенные позиции у одного ребенка), то верно, что в поддереве v_i не более $n = l^{2m+3}$ выделенных позиции.

То, что можно сколько угодно раз повторять v, y доказывается абслютно так же, как и для обычной леммы о накачке. Просто говорим, что мы нетерминал, стоящий в v_i, превратили в его же (v_j), а еще слева и справа выписали v, y. А теперь можно v_j опять вот так вот разложить, и так сколько угодно раз. А можно еще вообще сразу сказать, что мы ничего дублировать не будем и вместо v_i нарисуем сразу v_j со своим поддеревом, просто v, y вырежем. ◀

Замечание 2.4.2. Можно было привести в нормальную форму и получить $l = 2$, доказательство бы от этого не изменилось.

Следствие 2.4.1.1. Обычная лемма о накачке, когда выделяем все позиции в лемме Огдена.

2.5. Нормальная форма Грейбах

Замечание 2.5.1. Грейбах — женщина, поэтому именно так, а не Грейбаха. Как алгоритм Ахо-Корасик.

Это когда есть только продукции вида

$A \to a$

$A \to aB$

$A \to aBC$

Ну и обычный формализм — если хотим пустое слово, то делаем новый стартовый символ.

На практике это соответствует очень удобным автоматам с магазинной памятью.

Еще есть ослабленная НФ Грейбах — это только продукции $A \to a\beta, a \in \Sigma, \beta \in N^*$.

Теорема 2.5.1. Любая КС-грамматика представима в виде ослабленной НФ Грейбах.

Замечание 2.5.2. Можно и обычную НФ получить, но на лекции не получилось.

▶ G — КС-грамматика

1. Удаление ε-продукций.
2. Удаление непосредственных левых рекурсий. Это когда $A \to A\beta$. Есть еще косвенная левая рекурсия, это когда мы в итоге получим слева A, но не сразу, а через несколько продукций (например, $A \to B \to Aa$). Аналогично бывают и правые рекурсии.
   Непосредственная левая рекурсия на самом деле просто говорит, что мы порождаем что-то, где нет A в качестве первого символа, а потом кучу раз дописываем в конец β.
   Пусть была продукция $A \to A\beta_1 \mid A\beta_2 \mid ... \mid A\beta_k \mid \gamma_1 \mid ... \mid \gamma_m$, γ_i не начинаются с A. Тогда заменим ее на
   $A' \to \beta_1 A' \mid ... \mid \beta_k A' \mid \beta_1 \mid \beta_k$
   $A \to \gamma_1 A' \mid ... \mid \gamma_m A' \mid \gamma_1 \mid ... \mid \gamma_m$
   Понятно, что получили то же самое, но теперь нету непосредственной левой рекурсии.
3. А теперь удалим еще и косвенную.

```
for i = 1 .. n
    for j = 1 .. i - 1
        for продукция $A_i \to A_j \gamma$
            for продукция $A_j \to \beta$
                создадим $A_i \to \beta \gamma$
            Убираем продукцию $A_i \to A_j \gamma$
    Устраним непосредственные левые рекурсии для $A_i$
```

   Здесь $\beta, \gamma \in (\Sigma \cup N)^*$.
   Докажем, что после i-го шага из A_i есть только продукции $A_i \to A_{>i} ... \mid a$. Как обычно, делаем это по индукции, база $i = 1$ верна (это просто отсутствие непосредственных левых рекурсий).
   Переход: есть продукции $A_i \to A_i ...$, это непосредственные левые рекурсии, мы их убиваем. Есть еще $A_i \to B = A_{<i}$. Мы знаем по предположению индукции, что B раскрывается в $A_{\geq i} ...$, равенство убьем, получим корректный переход.
4. Удаление нетерминалов в началах правых частей

```
for i = n .. 1
    for j = i + 1 .. n
        for продукция $A_i \to A_j \gamma$
            удаляем ее
            for продукция $A_j \to \beta$
                создаем $A_i \to \beta \gamma$
```

   Будет поддерживаться (на каждой итерации внешнего цикла) инвариант что все продукции для $A_{\geq i}$ имеют вид $A_{\geq i} \to a\gamma$, доказывается как и прошлый пункт.
5. Теперь в начале всегда терминалы, но их может быть несколько. Формализм — введем соответствующие нетерминалы для каждого терминала, заменим лишние терминалы на нетерминалы.

◀

Замечание 2.5.3. Вообще говоря, приведенное доказательство неправильное. Проблема в том, что, когда мы убираем непосредственные левые рекурсии каждую итерацию, у нас появляются какие-то новые терминалы. Во-первых, у нас из-за этого появляются новые нетерминалы (почему внешний цикл не зациклится?), а еще есть проблема, что у нас может сломаться инвариант $A_i \to A_{>i} ...$.

Эти проблемы решить очень просто: скажем, что у нового, штрихованного терминала номер будет минимальным среди всех. Тогда все продукции, где он в левой части, будут удовлетворять

инвариант (и итерацию внешнего цикла по нему делать не надо), а сам по себе он нигде не является самым левым нетерминалом в правой части, поэтому второй проблемы тоже нет.

На этом курс закончился. Лекций больше не планируется, конец.