

Лингвистические основы информатики (ЛОИ)

12.02.2019

Орг. вопросы

- годовой курс: зачёт+экзамен;
- Петрова Елена Александровна, elena.petrova@urfu.ru;
- консультации по понедельникам в 16:10 на кафедре алгебры и фундаментальной информатики.

Рекомендуемая литература

- [Языки, грамматики, распознаватели](#) (Шур, Замятин) - основной учебник (много багов!)
- Ахо, Лам, Сети, Ульман "Компиляторы. Принципы, технологии, инструменты" (Dragon book)
- Ахо, Ульман "Теория синтаксического анализа, перевода и компиляции"
- Cooper K. Engineering a Compiler.

[репозиторий с .djvu книгами](#)

Чем будем заниматься?

— Теорией компиляции. Узнаем:

- что такое язык;
- что такое компилятор;
- что делает компилятор с языком.

В итоге будем знать, как работают и как написать (*в теории*) компиляторы.

Немного комментариев и истории:

Даже разбор формулы в Экселе использует какие-то приёмы компиляции!

В 50-х годах людям надоело писать на ассемблере, и они начали думать. К 60-м придумали.

Дейкстра - двигатель прогресса, потому что придумал теорию, а не какое-то специфичное для задачи решение.

Что такое компилятор?

По-простому – переводчик с языка на язык. Можно рассматривать как чёрный ящик с каким-то входом, выходом и магией внутри.

Принято разделять его работу на 2 фазы:

↓ *исходный текст*

фронтенд: **анализ** исходного текста. Если есть ошибки, то останавливаемся.

↓ *промежуточное представление*

бэкенд: **синтез** - генерация программы, которая нам нужна вместе с какими-то оптимизациями.

↓ *целевой код*

Заниматься будем фронтендом!

Блок анализа

↓ *исходный текст*

лексический анализ: разбиваем текст на токены – знаки, переменные, идентификаторы.

↓ *токены*

синтаксический анализ (парсер): определяем, как можно получить такую терминальную строку

↓ *синтаксическое дерево*

семантический анализ: проверка типов.

↓ *промежуточное представление*

Язык

1. Лексика — слова
2. Синтаксис — правила построения предложений
3. Семантика — типы и подходящие им операции

Таблица символов - информация о переменных, константах, функциях. Используется на всех шагах анализа.

Заполнение:

- лексика (?): встречаем новый символ - записываем имя переменной и указываем место первого появления.
- семантика: тип, место хранения, время объявления

Написанию компилятора предшествует описание языка.

Рассмотрим язык с условным оператором. Что есть условный оператор с точки зрения синтаксиса? Опишем это с помощью **форм Бэкуса-Наура**¹.

```
1 <условный оператор> ::= if <логическое выражение> <список операторов> { else
2   <список операторов> }
3 <список операторов> ::= <оператор> | <оператор>; <список операторов>
4
5 ...
6
7 <идентификатор> ::= [a-zA-Z]\w*
```

Обозначения

| {} — альтернатива

<> — синтаксическая категория

::= — выводимость

Грамматика

[Порождающая] грамматика - объект математический. Основной способ описания синтаксиса и лексики (частный случай синтаксиса).

Опр. Грамматика $G = \langle \Sigma, \Gamma, P, S \rangle$, где

- Σ - терминальный алфавит (выходной);
- Γ - нетерминальный алфавит (вспомогательный);
- P - множество правил вывода;
- $S \in \Gamma$ - выделенный нетерминал - аксиома (одна).

Соглашения

- a, b, c, \dots - терминальные символы (if - терминал);
- x, y, z, \dots - терминальные слова (последовательности терминальных символов);
- A, B, C, \dots - нетерминальные символы;
- X, Y, Z, \dots - слова из любых символов;
- $\alpha, \beta, \gamma, \dots$ - совокупные слова, содержащие как терминальные, так и нетерминальные символы.
- λ - пустое слово.

Выводимость

Правило вывода: $\alpha \rightarrow \beta, \alpha, \beta \in (\Sigma \cup \Gamma)^*$, точнее $\alpha \in (\Sigma \cup \Gamma)^* \Gamma (\Sigma \cup \Gamma)^*$

↑ в альфе должен быть хотя бы 1 нетерминал!

Таким образом, терминальные символы стоит понимать как символы, из цепочек которых ничего нельзя вывести.

Основная функция этого правила — порождение языка.

Опр. Цепочка γ **непосредственно выводима** из цепочки σ , если:

- $\sigma = \delta_1 \alpha \delta_2$
- $\gamma = \delta_1 \beta \delta_2$
- $(\alpha \rightarrow \beta) \in P$

Обозначается как $\sigma \Rightarrow \gamma$ (или, при необходимости, $\gamma \Leftarrow \sigma$).

В цепочке сигма есть подпоследовательность альфа, которую можно заменить бетой

Выводимость - отношение на множестве цепочек. Рефлексивно-транзитивное замыкание $\sigma \Rightarrow^* \gamma$ — возможность вывести одну цепочку из другой за некоторое число шагов

Опр. γ **выводима** из σ если существует последовательность цепочек $\eta_0, \dots, \eta_n, n \geq 0$ такая, что $\eta_0 = \sigma, \eta_n = \gamma, \eta_{i-1} \Rightarrow \eta_i (\sigma \Rightarrow^* \gamma)$

Последовательность η_0, \dots, η_n — **вывод**

Получается, что грамматика для нас — просто набор правил вывода. Потому что всё остальное мы зафиксировали в обозначениях.

Опр. **Язык**, порождённый грамматикой $G = \langle \Sigma, \Gamma, P, S \rangle : \{w \in \Sigma^* | S \Rightarrow^* w\}$ — множество терминальных цепочек таких, что их можно вывести из аксиомы.

Опр. Дан вывод η_0, \dots, η_n :

- $\eta_0 = S$
- $\eta_n = w$
- $\eta_{i-1} \Rightarrow \eta_i$
- η_i — **форма** грамматики (шаг) — цепочка, принадлежащая выводу терминальной цепочки.

Если форма получена применением правила к самому левому нетерминалу в предыдущей форме, то она называется *левой*. Иначе — *правой*.

Пример

Убедимся в том, что язык $L = \{a^n b^n | n \in \mathbb{N}_0\}$ порождается грамматикой $G = \langle \{S\}, \{a, b\}, P, S \rangle$, в которой P состоит из следующих правил вывода:

- $S \rightarrow aSb$
- $S \rightarrow \lambda$

Рассмотрим вывод терминальной цепочки:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

ab - терминалы (см. соглашения)

Но тогда и слова $a^n b^n$ могут быть получены после n применений первого правила вывода к аксиоме S и затем однократным применением второго правила.

Ещё пример

$$S \rightarrow ABS | \lambda \quad S \rightarrow SS | a | b | \lambda$$

$$AB \rightarrow BA$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$S \Rightarrow ABS \Rightarrow ABABS \Rightarrow^* (AB)^n S \Rightarrow (AB)^n$$

Можем перейти к терминалам

$$S \Rightarrow^* ABABAB \Rightarrow ABBAAB \Rightarrow abbaab$$

Хотим загнать буквы A в конец, а B в начало. Будем менять местами буквы по второму правилу.

$$ABABAB \Rightarrow BA_AB_AB \Rightarrow B_AB_AAB \Rightarrow BBAA_AB_ \Rightarrow \dots$$

19.02.2019

```
1 pos = init + rate * 60;
2
3 // после лексического анализа превращается в...
4 id,15 <=> <id,2><+><id,3><*><const><;>
5 // синтаксическому анализу всё равно, как называется переменная
6
7 // после синтаксического анализа превращается в...
8 =
9 / \
10 id,1 +
11 / \
```

12	id,2 *
13	/ \
14	id,3 const
15	//после семантического добавятся какие-то атрибуты

На каждой стадии – новый язык. Значит, нужны новые способы порождения\описания. А этот способ порождает распознаватель.

Иерархия Хомского-Шютценберже

	Вид грамматики	Распознаватель	Класс языков
0	Грамматика обычного вида	МТ	Рекурсивно перечислимые
1	Контекстно-зависимые	МТ с линейно ограниченной памятью (LBA)	КЗЯ
2	Контекстно-свободные	Недетерминированный автомат с магазинной памятью (PDA)	КСЯ
3	Праволинейные	ДКА	Регулярные языки

Опр. **Контекстно-зависимая** грамматика — грамматика, все правила которой имеют вид $\alpha A \gamma \rightarrow \alpha \beta \gamma$ (у терминала имеется контекст, который сохраняется при его раскрытии).

Опр. Язык обладает свойством P , если \exists грамматика со свойством P , его порождающая.

Опр. **Контекстно-свободная** грамматика — грамматика, все правила которой имеют вид $A \rightarrow \beta$ (частный случай КЗГ, когда оба контекста пусты).

Опр. **Праволинейные** грамматики — грамматика, все правила которой имеют вид $A \rightarrow aB$ или $A \rightarrow \lambda$ (справа либо лямбда, либо терминал+нетерминал).

Вспомним пример. Кажется, что это грамматика обычного вида.

$$S \rightarrow ABS | \lambda \quad S \rightarrow SS | a | b | \lambda$$

$$AB \rightarrow BA$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Построим КСГ, которая породит язык выше. Порождаем цепочки, где букв B на одну больше, чем a .

Из A должны выводиться строчки, где на одну a больше

$$S \rightarrow aB | bA$$

$$A \rightarrow aS | bAA$$

$$B \rightarrow bS | aBB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$abba : S \rightarrow aB \rightarrow abS \rightarrow abbA \rightarrow abba$

Иерархия: регулярные \subset КСЯ \subset КЗЯ \subset Rec \subset RecEn².

Контекстно-свободные грамматики и языки

Деревья вывода

Опр. **Упорядоченное дерево** — дерево с заданным линейным порядком со следующими свойствами:

1. Если x — сын узла y , то $x \geq y$
2. Если $x \leq y$ и они братья, то для всех сыновей z узла x : $z \leq y$

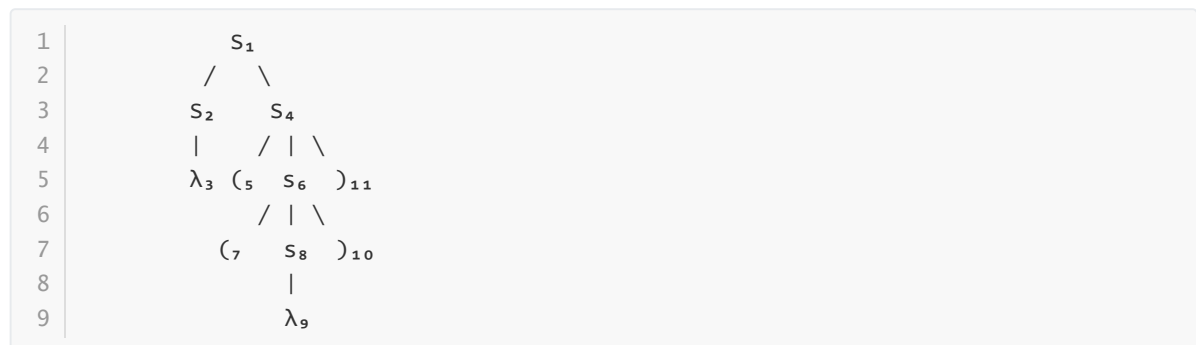
Порядок, возникающий при обходе в глубину слева направо

Пример:

$S \rightarrow SS|(s)|\lambda$

$(())$

$S \rightarrow SS \rightarrow S \rightarrow (S) \rightarrow ((S)) \rightarrow (())$



Опр. **Дерево вывода** цепочки ω в контекстно-свободной грамматике $G = \langle \Sigma, \Gamma, P, S \rangle$ — упорядоченное дерево со следующими свойствами:

1. Узлы — нетерминалы, корень — аксиома, листья — терминалы или λ , причём у листьев, помеченных пустым словом нет братьев.

Если у узла есть братья, то λa схлопывается до a

2. Если у узла x все сыновья это некоторый набор y_1, \dots, y_n , таких, что $y_1 \leq \dots \leq y_n$, и узлы x, y_1, \dots, y_n помечены символами X, Y_1, \dots, Y_n , то $(X \rightarrow Y_1, \dots, Y_n) \in P$.

Применили правило, в дереве появился куст вывода

3. Если все листья дерева имеют метки $a_1 \leq a_2 \leq \dots \leq a_n$, то $\omega = a_1 \dots a_n$.

Крона дерева задаёт цепочку ω

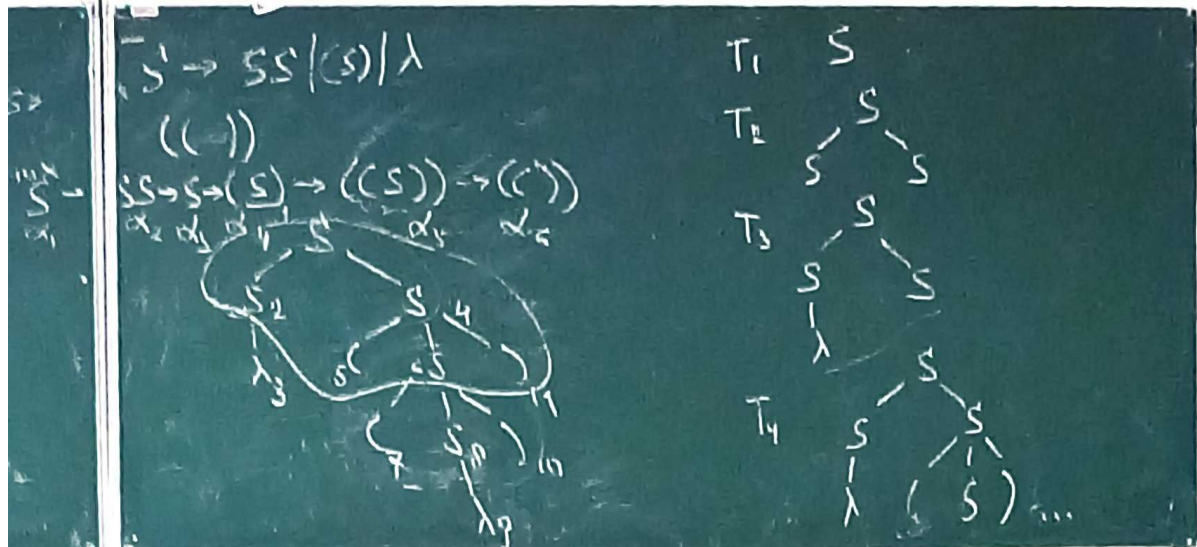
Опр. Вывод цепочки $\omega(S \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n = \omega)$ в $G = \langle \Sigma, \Gamma, P, S \rangle$ представлен деревом вывода T , если \exists набор стандартных поддеревьев T_1, \dots, T_n таких, что на упорядоченных листьях дерева T_i написана форма α_i .

Крона поддерева T_i задаёт форму α_i

Опр. Поддерево T' дерева T называется **стандартным**, если:

1. корень T' - корень T
2. Если узел x дерева $T \in T'$, то либо x — лист в этом поддереве, либо все сыновья x в $T \in T'$

Пример по последнему языку:



Наша любимая грамматика, которая порождает арифметику:

$$E \rightarrow E + E|E * E|(E)|x$$

$$x \vdash x * x$$

```

1      E
2      /\
3  E  +  E   - этот куст можно передвинуть влево, получится два разных дерева
4  |      /\   для одного и того же. Плохо.
5  x      E  * E
6         |    |
7         x    x

```

Опр. Грамматика **однозначна**, если $\forall \omega$, выводимой в грамматике, $\exists!$ дерево вывода.

Следующая грамматика однозначна и эквивалентна предыдущей

$$E \rightarrow E + T|T$$

$$T \rightarrow T * F|F$$

$$F \rightarrow (E)|_x$$

1. Правосторонний вывод и r-формы:

$$E \rightarrow E + T \rightarrow E + T * F \rightarrow E + T * x \rightarrow E + F * x \rightarrow E + x * x \rightarrow T + x * x \rightarrow F + x * x \rightarrow x + x * x$$

2. Левосторонний вывод и I-формы:

$$E \rightarrow E + T \rightarrow T + T \rightarrow F + T \rightarrow x + T \rightarrow x + T * F \rightarrow x + F * F \rightarrow x + x * F \rightarrow x + x * x$$

Плата за однозначность — увеличение длины вывода.

26.02.19

Теорема. Праволинейная грамматика порождает регулярный язык.

Д-во:

Суть: построим автомат и по теореме Клини³ — готово.

$$G = \langle \Sigma, \Gamma, P, S \rangle$$

Конечный автомат: $A = (\Sigma, \Gamma, \delta, S, F)$,

$F = \{A \in \Gamma \mid (A \rightarrow \lambda) \in P\}$ — терминальные состояния — такие нетерминалы, из которых выводится пустое слово

$\delta(A, a) = B \iff (A \rightarrow aB) \in P$ — переход возможен, если есть такое правило вывода

$$\omega = a_1 \dots a_n: \quad S \rightarrow a_1 A_1 \rightarrow a_1 a_2 A_2 \rightarrow \dots \rightarrow a_1 \dots a_n A_n \rightarrow a_1 \dots a_n$$

■

Пример

$a(b+cc)^*$ — чтобы построить грамматику, проще сначала нарисовать автомат, распознающий этот язык. Обозначим все состояния нетерминальными символами. А дальше — как в теореме выше, только в обратную сторону.

$$S \rightarrow aA$$

$$A \rightarrow bA \mid cB \mid \lambda$$

$$B \rightarrow cA$$

Преобразования грамматик

Хотим научиться удалять лишние вещи, которые не несут никакой пользы.

Приведённые грамматики

Опр. Нетерминал $A \in \Gamma$ называется **производящим**, если $A \Rightarrow_G^* \omega$.

== из него можно получить терминальную цепочку.

Опр. Нетерминал $A \in \Gamma$ называется **достижимым**, если $S \Rightarrow_G^* \alpha A \beta$

== его можно получить из аксиомы.

Опр. Грамматика **приведённая**, если все её нетерминалы достижимые и производящие.

Пример

$$S \rightarrow bAc \mid AcB$$

$$A \rightarrow abC$$

$$B \rightarrow Ea$$

$$C \rightarrow BD$$

$$D \rightarrow CCa$$

$$E \rightarrow Fbb$$

$$F \rightarrow a$$

Производящие (*producing*): $\Gamma_p = \{F, E, B\}$.

Если среди производящих нетерминалов нет аксиомы, то язык пустой.

Достижимые (*reachable*): $\Gamma_r = \{S, A, B, C, E, D, F\}$

Нахождение Γ_r :

- $\Gamma_r^1 \leftarrow S$;
- $\Gamma_r^n = \Gamma_r^{n-1} \cup \{A \mid (B \rightarrow \alpha A \beta) \in P, B \in \Gamma_r^{n-1}\}$.

смотрим, какие нетерминалы есть справа и добавляем те, которых ещё нет в Γ_r .

Нахождение Γ_p :

- $\Gamma_p^1 \leftarrow \{A \mid (A \rightarrow \omega) \in P\}$;
- $\Gamma_p^n = \Gamma_p^{n-1} \cup \{A \mid (A \rightarrow \gamma) \in P, \gamma \in (\Sigma \cup \Gamma_p^{n-1})\}$.

смотрим на достижимые из Γ_p^{n-1} нетерминалы;

Теорема. Для любой КСГ G существует эквивалентная ⁴ ей приведённая грамматика.

Д-во:

$$\text{КСГ } G = \langle \Sigma, \Gamma, P, S \rangle$$

Находим Γ_p :

- если $S \notin \Gamma_p$, то $G' = (\Sigma, \emptyset, \emptyset, \emptyset)$
- иначе $\tilde{G} = (\Sigma, \Gamma_p, \tilde{P}, S)$
 $\tilde{P} = \{(A \rightarrow \gamma) \in P \mid A, \gamma \in (\Sigma \cup \Gamma_p)^*\}$

Находим $(\Gamma_p)_r$

$(\Gamma_p)_r$ - достижимы в \tilde{G} , G' , производящие в \tilde{G} , G

$$A \in (\Gamma_p)_r: S \Rightarrow_{\tilde{G}}^* \alpha A \beta \Rightarrow_G^* uvv$$

Выкинули правила вывода, которые и так не могли участвовать в выводе терминальных цепочек.



Пример

$$S \rightarrow ab|bAc$$

$$A \rightarrow CB$$

$$B \rightarrow aSA$$

$$C \rightarrow bC|d$$

$$\Gamma_p = \{C, S\}$$

$$(\Gamma_p)_r = \{S\}$$

$$G' = \{S \rightarrow ab\}$$

Больше не будем рассматривать неприведённые грамматики

λ-свободные грамматики

Опр. $A \in \Gamma$ — **аннулирующий**, если $A \Rightarrow^* \lambda$.

Опр. $Ann(G)$ — множество аннулирующих нетерминалов.

Хотим, чтобы это множество было пустым. Ну или хотя бы только с аксиомой. Потому что тогда нам жить станет проще (почему? Сократим грамматику?).

Чтобы от аннулирующих нетерминалов избавиться, нужно их найти

Пример

$$S \rightarrow aBC|AE$$

$$A \rightarrow bC|\lambda$$

$$B \rightarrow ACA$$

$$C \rightarrow \lambda$$

$$E \rightarrow CA$$

$$D \rightarrow bE|c$$

$Ann(G)$:

1. $\{A, C\}$
2. $\{A, C, B, E\}$
3. $\{A, C, B, E, S\}$

Опр. λ-**свободная** грамматика — грамматика, которая либо не содержит аннулирующих правил вида $(A \rightarrow \lambda)$, либо содержит единственное такое правило $(S \rightarrow \lambda)$ и S не встречается в правых частях правил вывода.

Теорема. Любая КС-грамматика эквивалентна λ-свободной КС-грамматике

Построение

$$G = \langle \Sigma, \Gamma, P, S \rangle$$

0. Если $\lambda \in L(G)$, то $\Gamma' = \Gamma \cup S', P' = P \cup \{(S' \rightarrow \lambda), (S' \rightarrow S)\}$

$$\text{Иначе } \Gamma = \Gamma', S = S', P = P'$$

Смысл: добавим аксиому, которая справа встречаться нигде не будет???

1. Построим $Ann(G)$.
2. Рассмотрим бинарное отношение на множестве форм:

$\beta \preceq \gamma$, если β — подпоследовательность γ и все символы γ , которых нет в β , аннулирующие.

Условие $\beta \preceq \gamma$ влечёт $\gamma \Rightarrow_G^* \beta$

$$P' = \left\{ A \rightarrow \beta \left| \begin{array}{l} \beta \neq \lambda \\ \beta \preceq \gamma \\ (A \rightarrow \gamma) \in P \end{array} \right. \right\}$$

Взяли все исходные правила. В новую грамматику положили их "части"-подстроки, в которых либо присутствует, либо удалён каждый из аннулирующих нетерминалов.

3. Видно, что аннулирующие правила мы не взяли, поэтому она λ -свободная по определению.

Доказательство корректности

$$L(G) = L(G')?$$

1. $w \in L(G')$

$$S \Rightarrow_{G'} \alpha_1 \Rightarrow_{G'} \dots \Rightarrow_{G'} \alpha_n = w$$

Рассмотрим переход $\alpha_i \Rightarrow_{G'} \alpha_{i+1}$:

- $(A \rightarrow \beta) \in P'$, значит, в $G \exists (A \rightarrow \gamma) \in P$.
- $\beta \preceq \gamma$, значит, $\gamma = \eta_1 \beta \eta_2$. Из этого следует, что $(A \rightarrow \eta_1 \beta \eta_2) \in P$
- η_1 и η_2 — аннулирующие, значит, $\gamma \Rightarrow_G^* \beta$, значит, $A \Rightarrow_G^* \beta$.
- И это верно для любых цепочек, то есть если $\gamma \Rightarrow_{G'}^* \beta$ то и $\gamma \Rightarrow_G^* \beta$

как построить такую же цепочку, используя другие правила? Непонятно.

2. $w \in L(G)$

Нарисуем дерево вывода T цепочки w в грамматике G

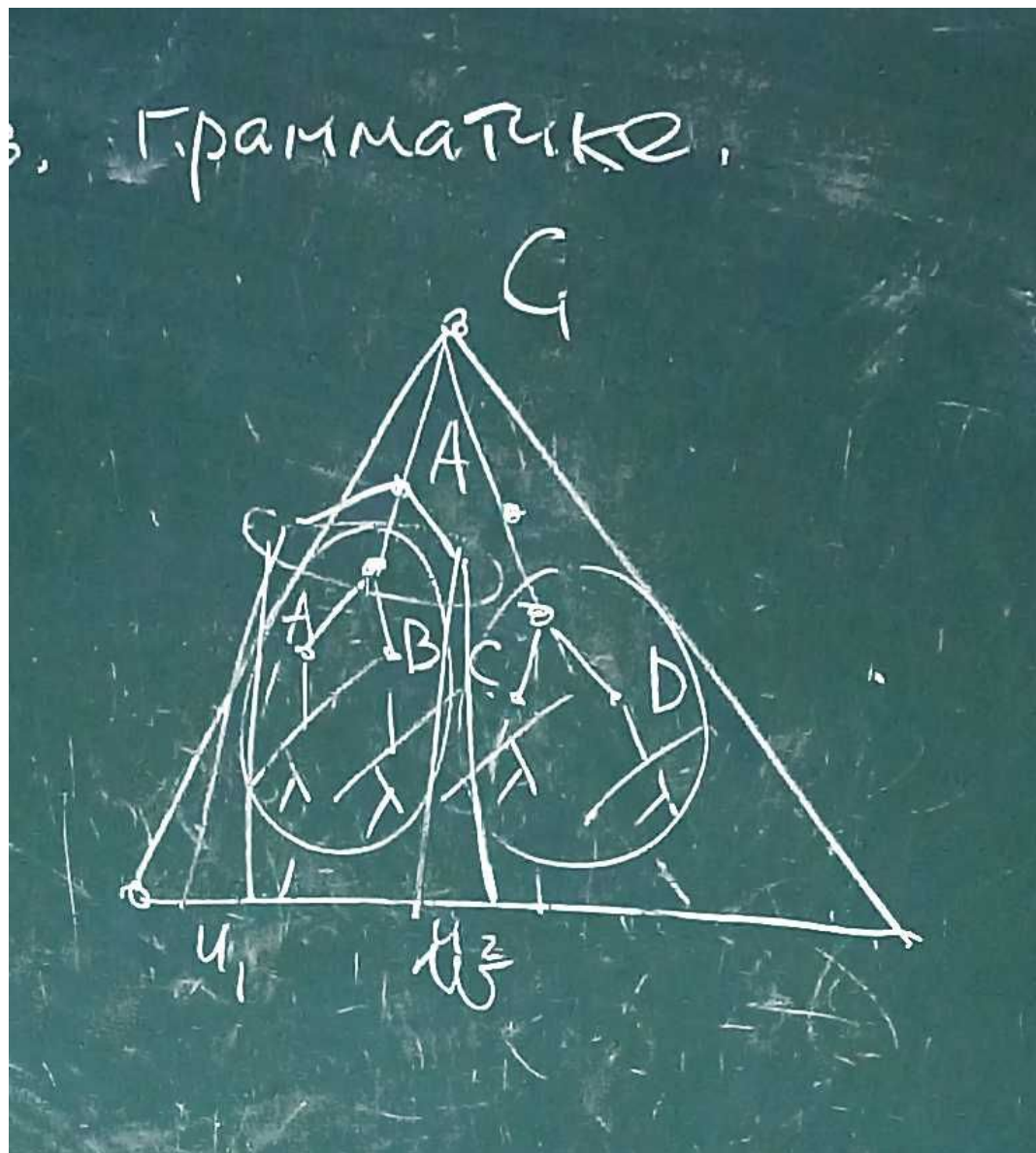
Обрежем все поддеревья с "пустой" кроной, получим *какое-то* дерево T'

Покажем, что T' — дерево вывода для w в G' :

- Посмотрим на какой-нибудь внутренний узел x в T'
- Метки его сыновей в T' образуют цепочку β
- Метки его сыновей, которые остались в T — аннулирующие. Назовём её η
- Если объединить эти цепочки, то получим $\gamma = \beta\eta$, то есть $\beta \preceq \gamma$.
- $\beta \neq \lambda$, по построению T' (все поддеревья с пустой кроной удалены)

Вот и всё, всё круто, в любом внутреннем узле дерева T' реализуется правило вывода грамматики G'

А ещё корень T' равен корню T (аксиоме), значит, **это дерево вывода w в G'**



05.03.2019

Нормальная форма Хомского

Нужна для доказательства важной теоремы, понадобится для алгоритма разбора.

Опр. Грамматика находится в ХНФ, если все её правила вывода имеют вид либо $A \rightarrow BC$ (справа ровно 2 нетерминала), либо $A \rightarrow a$. (справа один терминал), либо $S \rightarrow \lambda$.

Теорема. Любая КС грамматика эквивалентна некоторой грамматике в ХНФ.

Д-во. Конструктивное.

$G = \langle \Sigma, \Gamma, P, S \rangle$

Пусть G — исходная грамматика (λ -свободная)

1. Для всех правил грамматики, у которых в правой части хотя бы 2 символа сделаем следующее:

$\forall A \rightarrow X_1 \dots X_n, n \geq 2$

- если X_i — терминал, добавим новый нетерминал X'_i и правило $X'_i \rightarrow X_i$. Затем заменим вхождение терминала во всех правых частях на новый нетерминал.

Избавляемся от правил, где справа много терминалов.

2. $A \rightarrow B$ — **цепные** правила. Что делать с ними? Заменим правую часть на всё, что выводится из B . Но что, если есть цепочка $A \rightarrow B \rightarrow \dots \rightarrow A$ (цикл)? Сначала нужно от них избавиться.

Опр. Грамматика **циклическая**, если существует такой нетерминал A , что за какое-то ненулевое количество шагов из него выводится он сам. В противном случае — **ациклическая**.

Лемма. Любая грамматика эквивалентна некоторой ациклической.

Д-во. Пусть $A_1 \Rightarrow A_2 \Rightarrow \dots \Rightarrow A_n \Rightarrow A_1$

Мы рассматриваем только цепные правила, так как грамматика λ -свободная, то есть не возникнет ситуации $A \rightarrow BC \rightarrow AC \rightarrow A$ (если из C выводится λ)

Заменим все A_i на A и удалим правила $A \Rightarrow A$. Получилась G' . Готово.

Почему работает: $w \in L(G) \iff w \in L(G')$

\Rightarrow Вывод в G' получается стиранием индексов.

\Leftarrow Пусть A участвовал в выводе w . Пусть нетерминал A появлялся в какой-то правой части: $B \rightarrow \alpha A \beta$, $A \rightarrow \gamma$. Если такие правила были в G' , то в G существуют правила вывода $(B \rightarrow \alpha A_i \beta)$, $(A_j \rightarrow \gamma)$. Но мы знаем, что из $A_i \Rightarrow_G^* A_j$ (Если не совпадает с гаммой, то крутимся по циклу).

3. Пока в правых частях есть хотя бы 3 нетерминала, заменим два идущих подряд нетерминала на новый.

■

Пример

$S \rightarrow AB|aAb$

$A \rightarrow bB|aBC|\lambda$

$B \rightarrow AS|bA|a$

$C \rightarrow b$

Выведем λ -свободную грамматику. $Ann(G) = \{A\}$.

$S \rightarrow AB|B|_aAb|_ab|_a$

$A \rightarrow _bB|_aBC|_a$

$B \rightarrow AS|S|_bA|_b|_a$

$C \rightarrow b$

Приведём к ХНФ. Добавим $A' \rightarrow a$ и $B' \rightarrow b$:

$S \rightarrow AB|B|A'AB'|A'B'$

$A \rightarrow B'B|A'BC$

$B \rightarrow AS|S|B'A|b|a$

$C \rightarrow b$

Найдём цикл: $S \rightarrow B \rightarrow S$. Заменяем B на S , и подставляем в S всё, что выводится из B

$S \rightarrow AS|A'AB'|A'B'|B'A|b|a$

$A \rightarrow B'S|A'SC$

$C \rightarrow b$

Заменяем тройные нетерминалы на двойные, добавим $D \rightarrow AB'$ и $E \rightarrow SC$

$S \rightarrow AS|A'D|A'B'|B'A|b|a$

$A \rightarrow B'S|A'E$

$C \rightarrow b$

Свойства КСЯ

Лемма Огдена

Пусть есть L — КСЯ. Тогда $\exists m \in \mathbb{N} : \forall w \in L$ в которых помечено не менее m позиций, представимо в виде $w = uxzyv$, причём:

1. xy содержит хотя бы одну помеченную позицию;
2. xzy содержит не более m помеченных;
3. $ux^nzy^nv \in L \forall n \in \mathbb{N}_0$ (накачка).

Помечено - выбираем какие-то символы

Д-во. $G = \langle \Sigma, \Gamma, P, S \rangle, L = L(G)$

Пусть L порождается грамматикой в ХНФ, $m = 2^{|\Gamma|+1}$. Рассмотрим такое слово $w \in L$, что $|w| \geq m$ и пометим в нём не менее m позиций. Рассмотрим дерево вывода слова w (треугольник). Построим p_w — путь в этом дереве, построенный по следующим правилам:

- Корень (вершина треугольника) — аксиома. Принадлежит пути.
- Из двух (потому что ХНФ) потомков выберем того, из которого выводится больше выделенных позиций.

Точка ветвления — узел, у которого из *обоих* потомков выводится подслово w с помеченными позициями

Замечание: каждая следующая точка ветвления порождает не менее половины помеченных позиций w от тех, что порождает предыдущая точка.

Д-во:

БИ: $i = 0$. v_i — корень, порождает все m помеченных позиций

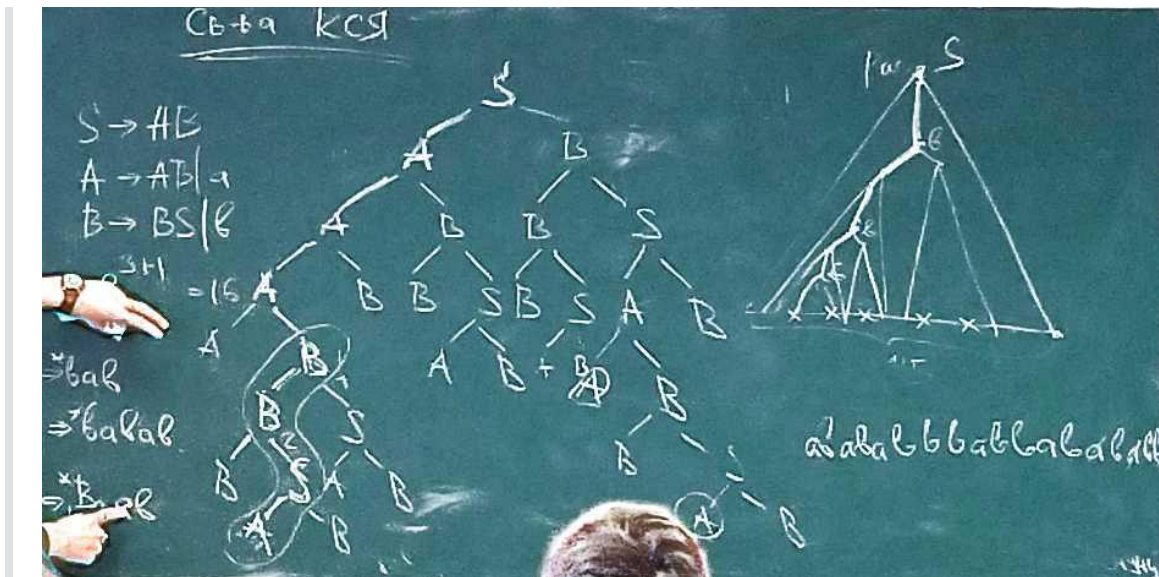
ШИ: рассмотрим v_i :

- не точка ветвления, тогда v_{i+1} порождает столько же точек ветвления, сколько и v_i
- точка ветвления, из двух сыновей выбираем потока с большим количеством выделенных позиций

в p_w не менее $|\Gamma| + 1$ точек ветвления (иначе количество помеченных позиций не совпадёт). Но у нас всего $|\Gamma|$ нетерминалов, значит, хотя бы 2 узла совпали — имеют одинаковую метку. Назовём её A . Рассмотрим две ближайшие к кроне вершины, чтобы потом можно было редуцировать поддереву, выводящееся из нижнего A

Тут надо нарисовать треугольник, и станет понятнее!

- Рандомный комментарий: для всех слов высота дерева вывода одинаковая! Для ХНФ.

$$B \rightarrow BS|b$$


Лемма о накачке

L — КСЯ $\Rightarrow \exists n, m \in \mathbb{N} : \forall w \in L : |w| \geq n : w$ представимо как $uxzyv$, причём:

- Суть: для любого КСЯ существуют натуральные константы такие, что любое слово определённой длины соответствует свойствам. Отсутствуют слова про выделенные позиции! То есть все символы выделены.

Приравнять n к m и все позиции сделать выделенными.

Следствия леммы о накачке

На экзамене будет вопрос про лемму о накачке и её следствия! Лемму доказывать не надо! Лемму Огдена надо. А следствия те, что ниже!

Сл. 1. Язык $\{a^n b^n c^n | n \in \mathbb{N}\}$ не КСЯ

Если не можем накачать слово, то это точно не КСЯ

Д-во: О.П. пусть язык L – контекстно свободный, следовательно выполняется лемма о накачке. Возьмём слово $a^l b^l c^l$, $l \geq m$, $3l \geq n$. Попробуем впихнуть туда xzy . Переберём все варианты. Если ни один не подойдёт – получим противоречие.

$|aa \dots aa|b \dots b|c \dots c|$

xzy расположены в одном блоке (1), либо на границе двух (2), т.к. длина блока — l больше либо равна максимально допустимой по лемме длине строки $xzy = m$

1. Накачиваться будет одна буква: $a^{l+r} b^l c^l \notin L$

2. Если x будет и в a , и в b , получится a после b , такое слово $\notin L$.

Значит, x лежит целиком в блоке a , y целиком в блоке b . Накачаем: $a^{l+r} b^{l+s} c^l \notin L$.

■

Сл. 2. Язык $L = \{ww | w \in \Sigma^*, |\Sigma| \geq 2\}$ — не КСЯ (язык квадратов)

Д-во: О.П. $\Sigma = \{a_1, \dots, a_n\}$. Должно накачиваться $a_1^l \dots a_k^l a_1^l \dots a_k^l$, где:

- $l > m$
- $2|w| \geq n$

Накачать можем одну из половинок или что-то посередине.

$|w|w|$

1. Накачаем вторую половину. Количество одинаковых букв l больше, чем m — максимальная длина накачиваемого слова. Значит, мы либо накачиваем одну букву a_k , либо a_k и a_{k+1} .

В левой половине (которую мы не накачивали) теперь $k * l$ букв, а в правой — $k * l + r$.

Если сейчас поделим слово пополам, то снова должно получиться "квадратное" слово. Но, левая половина теперь будет кончаться на a_1 , а вторая — на a_k .

$|w|a_1^l| \dots |w' \dots a_k|$

↑ новая граница

2. Качаем посерединке

$a_1^l \dots a_k^{l+r} a_1^{l+s} \dots a_k^l$, $r + s \leq m$ Б.О.О. $r \geq s$

- $r = s \Rightarrow$ в новом слове правая половина кончается на большее кол-во a_k
- $r > s$ первая половина начинается на a_1 , вторая — на a_k

■

Лемма о накачке не всеильна: $a^n b^n v^k$, $k \geq n$

См. доказательство [по ссылке](#)

Пример унарного языка: a^{n^r}

Теорема об унарных языках

Для языка $L \subseteq \{a\}^*$:

1. L — регулярный;

2. L — КСЯ;

3. мн-во длин слов из L — периодическое.

$M \subseteq \mathbb{N}$ — **периодическое**, если $\exists n_0, d \in \mathbb{N} : \forall n > n_0 (n \in M) \Rightarrow (n + d \in M)$

Д-во:

1. **2 \Rightarrow 3**

$\exists n, m : \forall a^n a^{n+r} (r \leq m):$

$a^n = [\text{по лемме о накачке}] = uxzyv = [\text{потому что язык унарный}] = uvzxy$

$uvz(xy)^k \in L$

Положим $n_0 = n$ (из леммы о накачке), $d = m!$

$m!$ делится на все $r \in \{1, \dots, m\}$, значит, $a^{n+lm!} \in L$.

2. **3 \Rightarrow 1**

построим автомат

M — периодическое множество длин слов.

$\forall i : 0 \leq i < d$ найдём минимальное $k_i : k_i \in M, k_i \equiv i \pmod d$. Если для какого-то i k_i не существует, положим его равным нулю.

M — бесконечное $\Rightarrow \exists i : k_i > 0$

Рисунок мухоловки с ручкой длины k , обода длины d

$\forall j \in \{0, \dots, k\}$ сост. q_j — заключительное $\iff a^j \in L$

Для остальных q_s — заключительное $\iff a^{s+rd} \in L$

3. **1 \Rightarrow 2** — очевидно.

■

Подстановки

Опр. Подстановка $\tau : 2^{\Sigma^*} \rightarrow 2^{\Delta^*}$

1. $\tau(\lambda) = \lambda;$

2. $\tau(a) \subseteq \Delta^*, a \in \Sigma;$ если a — слово, и выполняется пункт 3, то это гомоморфизм

3. $\tau(a_1 \dots a_n) = \tau(a_1) \cdot \dots \cdot \tau(a_n);$

4. $\tau(L) = \bigcup_{w \in L} \tau(w).$

Гомоморфизм — частный случай подстановки, при котором образ любой буквы — язык из одного слова ($\forall a \in \Sigma : \tau(a) = w \in \Delta^*$)

TODO: переписать

1559581398556 Что это была за картинка??? :(

Пример.

Который показывает, что операции объединения, произведения и итерации — частные случаи подстановки.

Пусть $\Sigma = \{a_1, a_2\}$

$\tau(a_1) = L_1 \subseteq \Delta^*$

$\tau(a_2) = L_2 \subseteq \Delta^*$

• $L = \{a_1, a_2\}$

$$\tau(L) = \tau(a_1) \cup \tau(a_2) = L_1 \cup L_2$$

- $L = \{a_1 a_2\}$

$$\tau(L) = \tau(a_1) \cdot \tau(a_2) = L_1 \cdot L_2$$

- $L = \{a_1\}^*$

$$\tau(L) = \tau(\{a_1\}^*) = \tau\left(\bigcup_{i=0}^{\infty} a_1^i\right) = \bigcup_{i=0}^{\infty} \tau(a_1^i) = \bigcup_{i=0}^{\infty} \tau(a_1)^i = \bigcup_{i=0}^{\infty} L_1^i = L_1^*$$

Теорема о подстановке.

Пусть $L \subseteq \Sigma^*$ — КСЯ, $\tau : 2^{\Sigma^*} \rightarrow 2^{\Delta^*}$ — подстановка: $\forall a \in \Sigma : \tau(a)$ — КСЯ.

Пусть τ — подстановка из одного конечного алфавита в другой, такая, что для любой буквы исходного алфавита, язык $\tau(a)$ — контекстно-свободный

Тогда $\tau(L)$ — КСЯ

Д-во:

L порождается $G = \langle \Sigma, \Gamma, P, S \rangle$, $\Sigma = \{a_1, \dots, a_n\}$

Каждому символу исходного алфавита сопоставим новую грамматику, которая будет задавать подстановку: $G_i = \langle \Delta, \Gamma_i, P_i, S_i \rangle$, $L(G_i) = \tau(a_i)$, $\forall i = 1..n$

Б.о.о. $\Gamma \cap \Gamma_i = \emptyset, \forall i$, $\Gamma_i \cup \Gamma_j = \emptyset, i \neq j$

Первое условие, для того, чтобы отловить переход из исходной грамматики в подстановку. Второе значит, что множества нетерминалов рассматриваемых грамматик попарно не пересекаются.

Грамматика $H = \langle \Delta, \bar{\Gamma}, \bar{P}, S \rangle$

$$\bar{\Gamma} = \Gamma \cup \bigcup_{i=0}^n \Gamma_i$$

$\bar{P} = P' \cup \bigcup_{i=0}^n P_i$, где P' получено из P заменой вхождений всех символов a_i в правой части на соответствующую аксиому S_i

То есть цепочку из исходных терминалов заменяем на подстановку, из которой можем получить что-то новенькое

$$L(H) = \tau(L)?$$

1. $w \in L(H)$

Построим дерево вывода T для w .

Так как S — корень дерева, принадлежит Γ , то $\exists T'$ — стандартное поддерево: все внутренние узлы из Γ , листья из $\bigcup_{i=1}^n \Gamma_i \cup \Delta$.

Γ_i , потому что мы могли не дойти до самого низа дерева, где появляются терминалы (см. [определение стандартного дерева](#))

Если $\exists (A \rightarrow \alpha B \beta) \in \bar{P} : A \in \Gamma, B \notin \Gamma$, то $\alpha, \beta = \lambda$ и $B = S_{i_j}$ по определению \bar{P}

Если метка какого-то внутреннего узла лежит в Γ , а метка его ребёнка — нет, то метка ребёнка — это какая-то из аксиом грамматик G_i . Больше никаких вариантов нет, так как B принадлежит либо Γ , либо Γ_i . Если это будет не S_i , то и A должно принадлежать Γ_i , что противоречит изначальному условию.

$B = S_{i_j}$, т.е. $S \Rightarrow_H^* S_{i_1} \dots S_{i_k} \Rightarrow_H^* w_{i_1} \dots w_{i_k} = w$, где

- $S_{i_j} \Rightarrow_{G_{i_j}}^* a_{i_j}$, т.е. $w_{i_j} \in \tau(a_{i_j})$

Аксиома S_{i_j} принадлежит грамматике G_{i_j} , в которой любое слово w_{i_j} — подстановка символа a_{i_j}

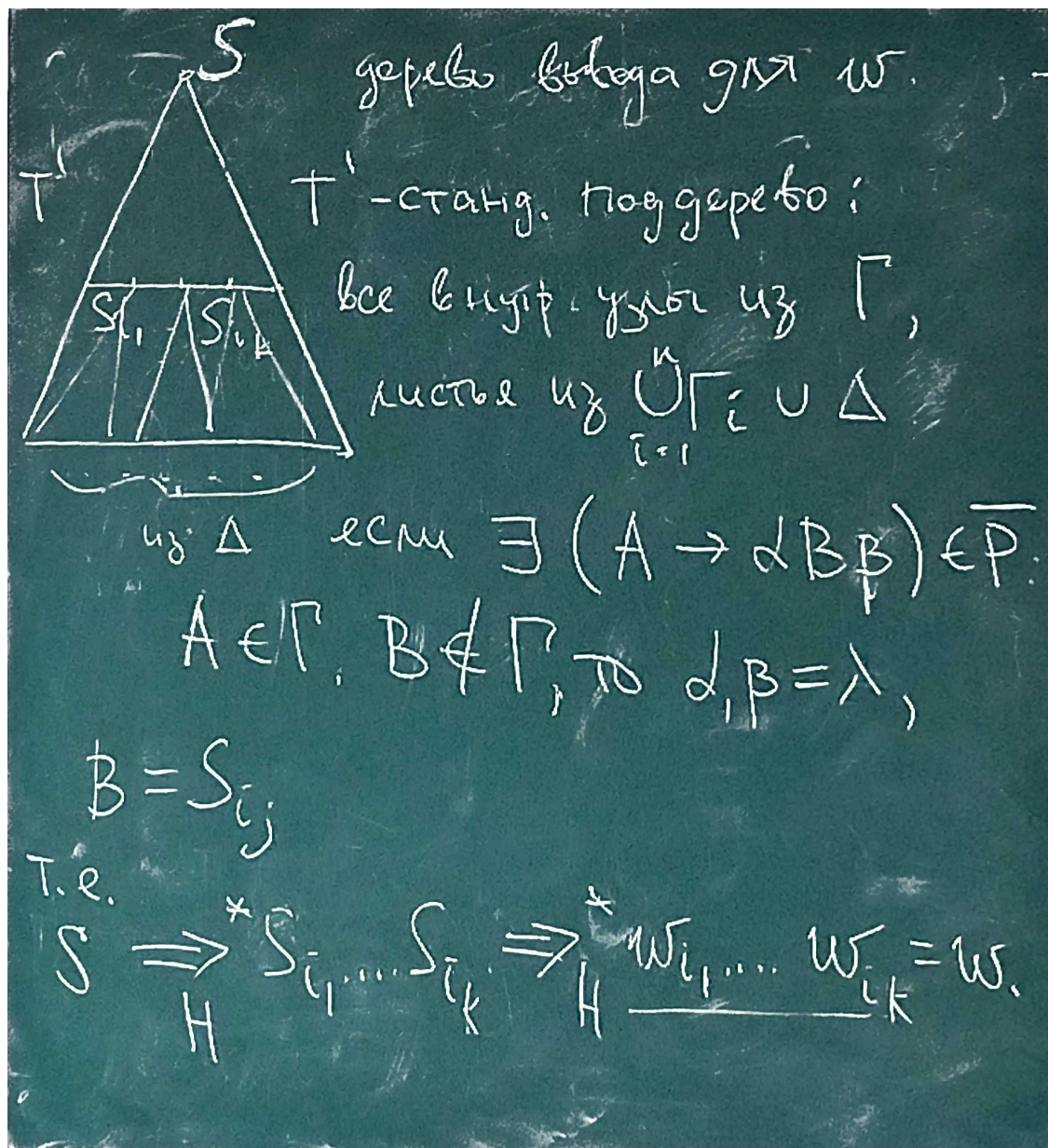
$w \in L(H)$, $w = w_1 \dots w_k$, потому что мы изначально рассматривали такое слово

$S \Rightarrow_G^* a_{i_1} \dots a_{i_k} = u \in L$ — по определению \bar{P} .

Значит, $w \in \tau(a_{i_1}) \dots \tau(a_{i_k}) = \tau(a_{i_1} \dots a_{i_k}) = \tau(u)$

" \in ", а не "=", потому что результат подстановки — множество.

Значит, $w \in \tau(L)$, потому что u — любое слово



2. $w \in \tau(L)$

$\exists u \in L : w \in \tau(u) \Rightarrow S \Rightarrow_G^+ u = a_{i_1} \dots a_{i_k} \iff S \Rightarrow_H^+ S_{i_1} \dots S_{i_k} \Rightarrow^+ w_{i_1} \dots w_{i_k}, w_{i_j} \in \tau(a_{i_j})$

$u = a_{i_1} \dots a_{i_k} \Rightarrow w \in \tau(a_{i_1} \dots a_{i_k}) = \tau(a_{i_1}) \dots \tau(a_{i_k})$

$w = w_{i_1} \dots w_{i_k}$

Следствия теоремы о подстановке

Сл. 1. Класс КСЯ замкнут относительно регулярных операций $(*, \cdot, \cup)$.

$$\{a_1, a_2\}$$

$$L_1 = \tau(a_1), L_2 = \tau(a_2) \text{ — КСЯ}$$

$$\tau(\{a_1, a_2\}_{\text{КСЯ}}) = L_1 \cup L_2$$

Сл. 2. Класс КСЯ замкнут относительно перехода к гомоморфным образам.

Гомоморфизм — частный случай подстановки. Применение подстановки к одному символу даёт язык из одного слова

$$\text{подстановка: } \tau(a) \subseteq \Sigma^*$$

$$\text{гомоморфизм: } \phi(a) \in \Sigma^*, \text{ т.е. } \phi(a) = L, |L| = 1$$

Предложение. Класс КСЯ не замкнут относительно пересечения и дополнения.

Д-во:

Пересечение:

$$L_1 = \{a^n b^n a^m | n, m \in \mathbb{N}_0\} \quad L_1 = \{a^n b^n | n \in \mathbb{N}_0\} \cdot \{a^*\}$$

$$L_2 = \{a^m b^n a^n | n, m \in \mathbb{N}_0\} \quad L_2 = \{a^*\} \cdot \{a^n b^n | n \in \mathbb{N}_0\}$$

Языки получены с помощью произведения КСЯ на КСЯ, значит, тоже КСЯ

$$L_1 \cap L_2 = \{a^n b^n a^n | n \in \mathbb{N}_0\}$$

$$L_1 \cap L_2 = \phi(L_3), L_3 = \{a^n b^n c^n | n \in \mathbb{N}_0\} \text{ — не КСЯ по лемме о накачке}$$

$$\phi(a) = a, \phi(b) = b, \phi(c) = a$$

Дополнение:

Выражаются через незамкнутое пересечение:

$$A \bar{\cap} B = \bar{A} \cup \bar{B}$$

$$A \cap B = \bar{\bar{A} \bar{\cap} \bar{B}}$$

Теорема о пересечении КСЯ с РЯ

Пересечение КСЯ с регулярным языком — КСЯ

Д-во:

Рассматриваем лямбда-свободные грамматики!

А что случится, если она будет не лямбда свободной? Дырки и лишние состояния?

$$L = L(G), G = \langle \Sigma, \Gamma, P, S \rangle \text{ — КСЯ}$$

$$A = (\Sigma, Q, \delta, q_0, F), M = L(A)$$

Что можно сказать про $L \cap M$?

Достаточно рассмотреть автоматы с единственным заключительным состоянием:

$$A = (\Sigma, \Gamma, \delta, q_0, f_i), f_i \in F$$

$$M = \bigcup_{f_i \in F} L(A_{f_i})$$

$$L \cap M = L \cap \bigcup_{f_i \in F} L(A_{f_i}) = \bigcup_{f_i \in F} L \cap L(A_{f_i})$$

Рассмотрим вспомогательную грамматику:

$$H = (\Sigma, \bar{\Gamma}, \bar{P}, \bar{S})$$

$$\bar{\Gamma} = Q \times (\Gamma \cup \Sigma) \times Q$$

$$\bar{S} = (q_0, S, f)$$

Правила вывода состоят из правил двух типов:

1. Те, что получаются из грамматики:

Если $A \rightarrow X_1 X_2 \dots X_n \in P$, то \forall набора состояний $p, q, r_1, \dots, r_{n-1} \in Q$:

$$(q, A, p) \rightarrow (q, X_1, r_1)(r_1, X_2, r_2) \dots (r_{n-1}, X_n, p) \in \bar{P}$$

2. Те, по которым есть правила перехода в автомате и $a \in \Sigma$:

$$\text{Если } \delta(q, a) = p, \text{ то } (q, a, p) \rightarrow a \in \bar{P}$$

$$L(H) = L \cap M?$$

$$w = a_1 \dots a_n$$

Поскольку правила второго вида порождают только листья дерева вывода, то можно считать, что при выводе терминальной цепочки из аксиомы грамматики H вначале выполняются правила первого вида, а затем правила второго вида:

$$\bar{S} \xRightarrow{(1)}_H^* (q_0, a_1, r_1)(r_1, a_2, r_2) \dots (r_{n-1}, a_n, f) \xRightarrow{(2)}_H^+ a_1 \dots a_n$$

По определению грамматики H первая из стрелок в этом выводе имеет место тогда и только тогда, когда $S \Rightarrow_G^* a_1 \dots a_n$, т.е. $a_1 \dots a_n \in L$

$$(q_0, S, f), q = q_0, p = f$$

Вторая из стрелок выполняема только тогда, когда $\delta(q_0, a_1 \dots a_n) = f$, т.е. $a_1 \dots a_n \in M$.

В итоге терминальная цепочка выводима в H тогда и только тогда, когда она принадлежит пересечению языков L и M . Получаем $L \cap M = L(H)$, что и требовалось.

■

Распознаватели КСЯ

Мы знаем, что регулярный язык можно распознать за линейное время. Про КСЯ пока ничего не знаем. Но, есть теорема, которая отвечает на этот вопрос. Попытаемся определить вхождение слова в КСЯ.

Алгоритм Кока-Янгера-Касами

$$G = \langle \Sigma, \Gamma, P, S \rangle \text{ — в ХНФ.}$$

Сначала нужно построить табличку. Пусть есть слово, которое мы проверяем:

$$w \in L(G) \Rightarrow \forall i, j, i \neq j : \exists A \in \Gamma : (A \rightarrow BC) \in P :$$

$$A \Rightarrow^* w[i..j],$$

$$B \Rightarrow^* w[i..k],$$

$$C \Rightarrow^* w[k+1..j], i \leq k \leq j$$

Для любой подстроки слова существует нетерминал, из которого эта самая подстрока выводится. При этом, так как справа стоят два нетерминала, из них тоже выводятся подстроки этой подстроки

Таблица — верхнетреугольная матрица размера $n \times n$, $|w| = n$

T_{ij}	Столбец - длина
Строка - позиция	Нетерминалы, из которых можно вывести подстроку из данной позиции с заданной длиной.

$T_{ij} = \{A | A \Rightarrow_G^+ w[i..i+j-1]\}$ — в ячейке храним нетерминалы, из которых выводится подстрока с позиции i длины j .

Первый столбец заполняется по правилам ХНФ (2):

$$T_{i1} = \{A | (A \rightarrow w[i]) \in P\}.$$

Остальные столбцы заполним, перебрав все возможные "распилы" строки на 2 части:

$$T_{ij} = \{A | \exists (A \rightarrow BC) \in P, B \in T_{ik}, C \in T_{i+k-1, j-k}, i \leq k < j-1\}$$

Если в $T_{1,n}$ есть S , то $w \in L(G)$.

Если в T_{ij} есть S , то в строке есть подстрока, принадлежащая $L(G)$ длины j с позиции i

Пример

$$S \rightarrow A' A | B B' | S S$$

$$A \rightarrow A' A | A' D | c$$

$$D \rightarrow C B'$$

$$B \rightarrow B B' | A' D | c$$

$$C \rightarrow A' D | c$$

$$A' \rightarrow a$$

$$B' \rightarrow b$$

$$w = aacbcbb$$

	1	2	3	4	5	6
1	A'	-	S, A	S, A	-	S
2	A'	S, A	A, B, C (acb)	-	-	
3	A,B,C	S, B, D	- (cbc)	S		
4	B'	- (с B' ничего не начинается)	- (bcb)			
5	A,B,C	S, B, D				
6	B'					

$w[1, 2] = w[1, 1]w[2, 2]$ — всего один способ поделить на 2 части

$w[1, 3] = w[1, 1]w[2, 3] = w[1, 2]w[3, 3]$ — можно поделить двумя способами:

- с позиции 1 длины 1 + с позиции 2 длины 2;
- с позиции 1 длины 2 + с позиции 3 длины 1.

Смысл: берём значение из ячейки слева (X), из ячейки справа (Y), и ищем нетерминал (Z), из которого выводится последовательность XY ($Z \rightarrow XY$). Если нашли такой терминал, то записываем.

Сложность: $n * n$ — таблица, n — распилы и поиск, итого $O(n^3)$

26.03.2019

$a^n b^n$ — не распознаётся ДКА.

$S \rightarrow aSb | \lambda$

МП-автоматы

Автоматы с магазинной памятью — стек, PDA — push-down automaton.

|с|л|о|в|о|...|—| — входная лента

 ↑ ↑
с ← |УУ| конец слова
т состояния

е

к

...

▽

Можем остаться на месте, или сдвинуться вправо

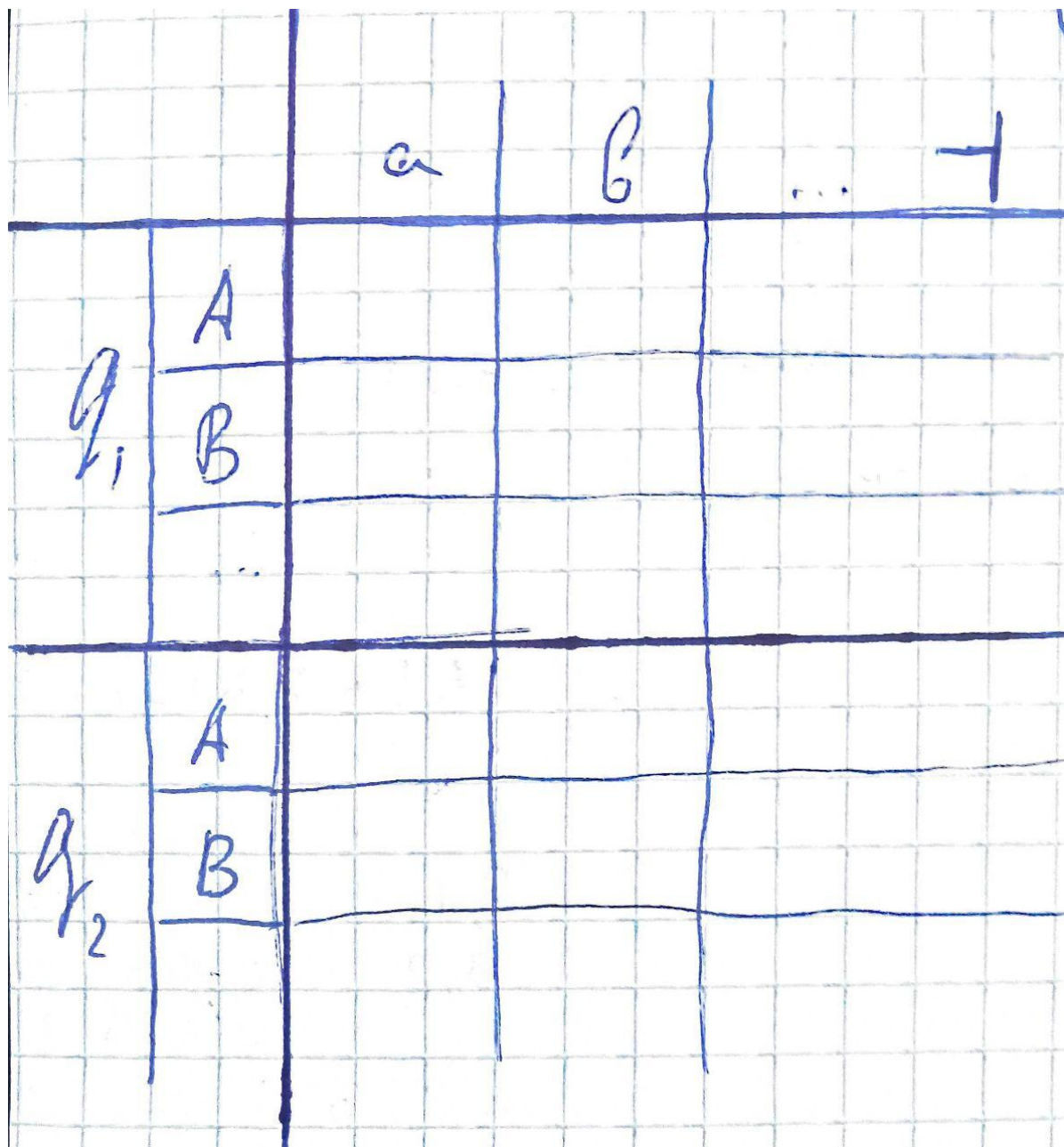
↓
 $(q, a, B) \rightarrow (q', \{ _, \rightarrow \}, \gamma) - (\cdot)$

Автомат закончит работу, когда дочитает строку и остановится в заключительном состоянии.

Опр. МП-автомат $M = (\Sigma, \Gamma, Q, \delta, i_0, F, \gamma_0)$

- Σ — входной алфавит;
- Γ — стековый алфавит;
- Q — множество состояний;
- δ — множество команд вида (\cdot) ;
- i_0 — начальное состояние;
- F — множество заключительных состояний;
- $\gamma_0 \in \Gamma^*$ — начальное состояние для стека.

Управляющая таблица для МП-автомата: по столбцам — символ, который читаем, по строкам — состояния и элемент на верхушке стека



Сколько мест в стеке, столько строк на каждое состояние

На каждом шаге обязательно надо читать из стека! Элемент при этом оттуда исчезает. Поэтому, если мы хотим, чтобы внизу всегда был символ ИКС, то в каждой команде нужно не забывать его туда класть.

Скобочный язык

в стеке будет лежать только открывающая скобка

При разборе — в левом префиксе открывающих скобок не меньше чем закрывающих

	()	⊙	⊢
q	(⊢ , →, //		
q	▽	→, (
⊢	(
⊢	▽			

Варианты распознавания МП-автомата:

- $(q, \vdash, B) \rightarrow \checkmark$ — команда допуска, слово читается.
- пустота стека (можно добавить переходов, которые просто очищают стек).

Опр. **Конфигурация** автомата — снимок его состояния $[q, w, \gamma]$

- q — текущее состояние;
- w — необработанная часть входной строки;
- γ — текущее содержимое стека.

Вершина стека пишется **слева!** (пока)

На множестве конфигураций можно построить отношение: возможность перехода из одной конфигурации в другую.

$[q, w, \gamma] \models [q', w', \gamma']$ — переход за 1 ход.

Опр. МПА **распознаёт** цепочку, если он дочитал её до конца и:

- оказался в заключительном состоянии
ИЛИ
- выполнил команду допуска
ИЛИ
- закончил работу с пустым стеком

Опр. МПА распознаёт w , если $[i_0 w, \gamma_0] \models^* [t, \lambda, \gamma], t \in F$.

Введение дополнительных стековых символов позволяет сократить количество состояний

$$L(M) = \{w \mid [i_0, w, \gamma_0] \models^* [t, \lambda, \gamma]\}$$

Пример

$$a^n b^n$$

Нужно следить, чтобы после b не появилось a . Для этого добавим 2 состояния: b ещё не было, b уже была. А можем выкинуть все состояния, и не заморачиваться, как на правой картинке.

		a	b	\neg			a	b	\neg
q	a	\rightarrow, aa	t			a	λ, \rightarrow		
	∇	\rightarrow, a				A	Aa, \rightarrow	λ	
t	a		\rightarrow			∇			
	∇								

Когда будем пошагово воспроизводить работу МП-автомата, стек будем писать в правой колонке!

02.04.2019

НМПА и ДМПА

ДМПА: $(q, a, B) \rightarrow (q', \{ _, \rightarrow \}, \gamma)$ — не более одной команды с такой левой частью

НМПА: $(q, a, B) \rightarrow 2^{(Q \times \{ _, \rightarrow \} \times \Gamma^*)}_{fin}$

fin — ограничиваем то, что складываем в стек, а то таблица станет бесконечной.

Теорема. Класс языков, распознаваемых НМПА, строго больше класса языков, распознаваемых ДМПА.

Д-во:

\overleftarrow{w} — слово w , развёрнутое задом наперёд.

$\{w\overleftarrow{w} \mid w \in \Sigma^*, |\Sigma| \geq 2\}$ — множество палиндромов.

$M = (\Sigma, \Gamma, \delta, X)$ — НМПА. X — символ, указывающий, что перехода к сравнению ещё не было.

$\Gamma = \Sigma \cup \{X\}$ — стековый алфавит.

$x, y \in \Sigma$

	x	y	...	\neg
X	Xx, \rightarrow $\lambda, -$	Xy, \rightarrow $\lambda, -$		
x	λ, \rightarrow			
y		λ, \rightarrow		
...				
∇				✓

НВ: вершина стека **слева**, запись Xx, \rightarrow означает, что на вершине стека будет X

Суть таблички: пока не дошли до середины слова, закидываем текущий символ в стек, а наверху оставляем маркер. Как только дошли — снимаем его со стека (команда $[\lambda, _]$). Затем просто ожидаем на входе те же символы, что и в стеке. Без проблем дошли до конца строки — это палиндром.

Что значит, что НМПА распознаёт символ?

Автомат с пустым стеком продолжать работу не может, так как на каждом шаге он что-нибудь берёт из стека.

О.П. \exists ДМПА, распознающий $\{w\overleftarrow{w} \mid w \in \Sigma^*, |\Sigma| \geq 2\}$. $x \in \Sigma, w \in \Sigma^*$

$wxx\overleftarrow{w}$ — распознаётся автоматом, потому что тоже палиндром. После прочтения wx автомат начнёт доставать элементы из стека и сравнивать.

Давайте подадим ему на вход $wxxxx\overleftarrow{w}$. Тут он к сравнению тоже перейдёт после wx и не распознает это слово.

■

МПА и КСЯ

Теорема. Любой КСЯ распознаётся НМПА с одним состоянием и одной командой допуска.

Д-во:

L — КСЯ, $G = \langle \Sigma, \Gamma, P, S \rangle$, G — КСГ, $L(G) = L$

Если состояние одно, то нигде про него говорить не будем. Поэтому тройки станут двойками.

$M = (\Sigma \cup \{\nabla\}, \Sigma \cup \Gamma \cup \{\nabla\}, \delta, S)$, S — начальное значение стека.

У нас остаётся входной и стековый алфавит

δ :

1. $(\nabla, \nabla) \rightarrow \checkmark$
2. $\forall a \in \Sigma : (a, a) \rightarrow (\lambda, \rightarrow)$
3. $\forall a \in \Sigma : (B, a) \rightarrow (\gamma, _)$
 $\forall (B \rightarrow \gamma) \in P$

Простыми словами:

1. Заканчиваем работу успешно, если стек пуст, а входная строка дочитана до конца
2. Читаем символ входной строки только тогда, когда на вершине стека — этот же символ
3. Для всех терминалов и для всех нетерминалов добавляем команду, которая будет “разворачивать” нетерминал по правилам из P .
Команд будет столько, сколько разных правых частей есть для этого нетерминала.

$L = L(M)?$

1. \Rightarrow

$$w \in L$$

\exists левосторонний вывод w в G

$$S \Rightarrow u_1 B_1 \gamma_1 \Rightarrow u_1 u_2 B_2 \gamma_2 \Rightarrow \dots \Rightarrow u_1 \dots u_{n-1} B_{n-1} \gamma_{n-1} \Rightarrow u_1 \dots u_n$$

Тогда в M реализуема следующая последовательность конфигурация

$$[w, S] = [u_1 \dots u_n, S] \models [u_1 \dots u_n, u_1 B_1 \gamma_1] \models^* [u_2 \dots u_n, B_1 \gamma_1] \models [u_2 \dots u_n, u_2 B_2 \gamma_2] \models^* \dots \models^* [u_n, B_{n-1} \gamma_{n-1}] \models [u_n, u_n] \models^* [\lambda, \lambda]$$

u_1 — цепочка из терминалов

γ_i — цепочка из терминалов и нетерминалов

Пока мы не дойдём до нетерминала мы продолжим чтение входной строки

B_{n-1} его правая часть — это какое-то правило, а левое — конец цепочки γ_n

2. \Leftarrow

$$w \in L(M)$$

Есть оракул, который говорит, что данная последовательность реализуема

$$[w, S] \models^* [\lambda, \lambda]$$

\uparrow — конечное число тактов m

$$w = a_1 \dots a_m, a_i \in \Sigma \cup \{\lambda\}$$

С помощью этой последовательности закодировали типы применяющихся команд. Если там лямбда, то мы применяли команду типа 2 и не сдвигались по входной строке.

$$[w, S] \models [a_1 \dots a_m, S] \models [a_1 \dots a_m, a_{i_1} B_1 \gamma_1] \models [a_{i_1+1} \dots a_m, B_1 \gamma_1] \models \dots \models [\lambda, \lambda]$$

Вспомогательная лемма. Произведение обработанной части входной строки на содержимое стека — левая форма G .

Левая форма — всё, что может возникнуть в процессе левостороннего вывода.

Д-во. По индукции.

БИ: в прочитанном — λ , в стеке — аксиома. $\beta_0 = \lambda \cdot S = S$. Аксиома — это левая и правая форма.

ПИ: обработанная часть: $a_1 \dots a_{n-1}$. В стеке γ_{n-1} . Произведение — левая форма.

ШИ:

- прочитали из строки символ

$$a_n \in \Sigma \Rightarrow \gamma_{n-1} = a_n \gamma_n.$$

$$\beta_{n-1} = \beta_n = a_1 \dots a_n \gamma_n$$

Прочитанное	Остаток строки	Стек
$a_1 \dots a_{n-1}$	a_n	γ_{n-1}
$a_1 \dots a_{n-1} a_n$	\vdash	γ_n

γ_{n-1} должно быть равно $a_n \gamma_n$, чтобы можно было прочитать последний символ.

Объединяем, получаем одинаковые формы на обоих шагах, а предыдущий является левой формой по ПИ.

- ничего из строки не прочитали

$$a_n = \lambda \Rightarrow \gamma_{n-1} = B_{n-1} \gamma'_{n-1}$$

$$\beta_{n-1} = a_1 \dots a_{n-1} B_{n-1} \gamma'_{n-1}$$

$$\beta_n = a_1 \dots a_{n-1} \gamma \gamma'_{n-1}$$

$$(\beta_{n-1} \rightarrow \gamma) \in P$$

Прочитанное	Остаток строки	Стек
$a_1 \dots a_{n-1}$	a_n	γ_{n-1}
$a_1 \dots a_{n-1}$	a_n	γ_n

Прочитанная строка не изменилась, значит, была применена команда вида (1) — мы раскрыли нетерминал с вершины стека по какому-то правилу из P . Значит, γ_{n-1} начинается с нетерминала. Посмотрим на предыдущую форму β_{n-1} и на новую β_n . Новая получена с помощью правила из P , которое применили к самому левому нетерминалу в левой форме β_{n-1} . Значит, и β_n — левая форма

■

Вернёмся к доказательству теоремы.

w — всё, что мы обработали, λ — то, что осталось в стеке. $w \cdot \lambda$ — левая форма G по лемме, то есть $w \in L(G)$

■

Что даёт теорема? Есть КСЯ, можем построить НМПА, его распознающий.

Теорема. Класс КСЯ и класс языков, распознающихся НМПА, совпадают.

Следствие. ДМПА распознаёт собственный подкласс КСЯ.

Пример

$$S \rightarrow (S)S | \lambda$$

Стековый алфавит — все терминалы, нетерминалы и дно стека. В начале в стеке аксиома

	()	\vdash
(λ, \rightarrow		
)		λ, \rightarrow	
S	$(S)S, _$ $\lambda, _$	$(S)S, _$ $\lambda, _$	$\lambda, _$
∇			✓

NB: не забываем, что вершина стека — слева!

Суть таблички: начинаем с аксиомы. Если во входной строке — скобочки, значит аксиому нужно “развернуть”. Если вложенные — то в нужный момент надо будет “развернуть” ещё раз. Если видим одинаковые скобки в стеке и во входной строке — считываем их. Если число закрывающих скобок в кусочке строки и в стеке одинаковое, то нужно убирать аксиомы с помощью правила с лямбдой.

Прочитаем $(()) \dashv$

Слева — прочитанное, справа — стек. Наша цель — получить дерево

Прочитанное	Остаток строки	Стек
λ	$(())$	S
λ	$(())$	$(S)S$
$($	$(())$	$S)S$
$($	$(())$	$(S)S)S$
$(($	$)$	$S)S)S$
$(($	$)$	$)S)S$
$(()$	$)$	$S)S$
$(()$	$)$	$)S$
$(())$	\dashv	S
$(())$	\dashv	∇

Сноски

- 1. Будем их использовать, чтобы не терять связь грамматики и компиляции.[↗](#)
- 2. [Recursively Enumerable](#)[↗](#)
- 3. Классы регулярных и автоматных языков совпадают[↗](#)
- 4. с таким же деревом вывода[↗](#)