**Определение1:** Грамматикой в **нормальной форме Хомского** (англ. *Chomsky normal form*) называется КС- грамматика, в которой могут содержаться правила только следующего вида:

*A*→*BC*,

*A*→*a*,

*S*→*ε*,

где *a* — терминал, *A*,*B*,*C* — нетерминалы, *S* — стартовая вершина, *ε* — пустая строка, стартовая вершина не содержится в правых частях правил. Иными словами, **грамматика имеет нормальную форму Хомского,** если правые части состоят из двух нетерминалов или одного терминала.

**Теорема.** Любую контекстно-свободную грамматику можно привести к нормальной форме Хомского.

Рассмотрим контекстно-свободную грамматику G. Для приведения ее к нормальной форме Хомского необходимо выполнить пять шагов. На каждом шаге мы строим новую G*i*, которая допускает тот же язык, что и G.

**Пример грамматики, которая является грамматикой в нормальной форме Хомского.**

\begin{align*}
S \; & {\to} \; RR , & A \; & {\to} \; a , \\
S \; & {\to} \; AB , & B \; & {\to} \; RB , \\
R \; & {\to} \; RR , & B \; & {\to} \; b \\
R \; & {\to} \; AB , 
\end{align*}

**АЛГОРИТМ ПРИВЕДЕНИЯ ГРАММАТИКИ К НОРМАЛЬНОЙ ФОРМЕ ХОМСКОГО**

**ШАГ 1.** УБИРАЕМ длинные правила. Получаем грамматику G1, эквивалентную исходной, содержащую правила длины 0,1 и 2.

**ШАГ 2 .** УДАЛЯЕМ *ε*-правила. Получим грамматику G2, эквивалентную исходной, но в которой нет *ε*- правил.

**ШАГ 3.** УДАЛЯЕМ цепные правила. Алгоритм удаления цепных правил работает таким образом, что новые *ε*-правила не образуются. Получим грамматику G3, эквивалентную G.

**ШАГ 4.** УДАЛЯЕМ бесполезные символы. Получим грамматику G4.

**ШАГ 5.** УБИРАЕМ ситуации, когда в правиле встречаются несколько терминалов.

Для всех правил вида *A*→*u*1*u*2 (где *ui* — терминал или нетерминал) заменим все терминалы *ui* на новые нетерминалы *Ui* и добавим правила *Ui*→*ui* . Итог преобразования - правила содержат либо одиночный терминал, либо строку из двух нетерминалов.

**Таким образом, мы получили грамматику в нормальной форме Хомского, которая допускает тот же язык, что и G.**

Стоит заметить, что порядок выполнения операций важен. Первое правило должно быть выполнено перед вторым, иначе время нормализации ухудшится до *O*(2|Γ|). Третье правило идет после второго, потому что после удаления *ε*-правил, могут образоваться новые цепные правила. Также четвертое правило должно быть выполнено позже третьего и второго, так как они могут порождать бесполезные символы.

**При таком порядке действий (преобразований) размеры грамматики возрастают полиномиально.**

После удалении длинных правил из каждого правила длины *k*⩾3 могло появиться *k*−1

новых правил, причем их длина не превышает двух. На этом шаге размер грамматики возрастает не более, чем вдвое.

При удалении *ε*-правил из грамматики, содержащей правила длины 0,1 и 2, размеры грамматики могли вырасти не больше, чем в 3 раза.

Всего цепных правил в грамматике не больше, чем *n*2 , где *n* — число нетерминалов. При удалении цепных правил мы берем каждую из цепных пар и производим добавление нецепных правил, выводимых из второго нетерминала в паре. Если максимальная суммарная длина всех правил, выводимых из какого-либо нетерминала, равна *k*, то размер грамматики возрастет не больше, чем на *k*⋅*n*2. Наконец, на последнем шаге может произойти добавление не более, чем |Σ| (Σ — алфавит грамматики) новых правил, причем все они будут длины 1

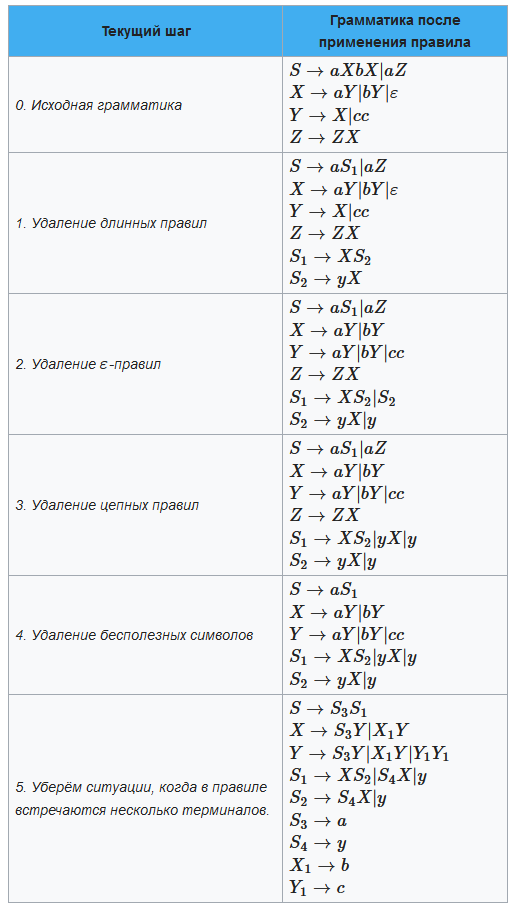


Рис.1 Применение алгоритма преобразования КС-грамматики к нормальной форме Хомского

**Определение 2.** Грамматика имеет нормальную форму Грейбах, если правые части всех правил начинаются с **терминала**, за которым следуют **нетерминалы**.

**Определение 2”**. Грамматикой в **нормальной форме Грейбах** (англ. *Greibach normal form*) называется [контекстно-свободная грамматика](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BD%D0%BE-%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D1%8B%D0%B5_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B8,_%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4,_%D0%BB%D0%B5%D0%B2%D0%BE-_%D0%B8_%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4,_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D1%80%D0%B0%D0%B7%D0%B1%D0%BE%D1%80%D0%B0), в которой могут содержаться только правила одного из следующих типов:

*A*→*aγ*

*S*→*ε*

где *a* — терминал, *A* — нетерминал (возможно, стартовый), *S* — стартовый нетерминал (причём он не должен встречаться в правых частях правил), *ε* — пустая строка, *γ* — строка не более, чем из двух нетерминалов.

**Определение 3**. Грамматикой в **ослабленной нормальной форме Грейбах** (англ. *Greibach weak normal form*) называется [контекстно-свободная грамматика](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BD%D0%BE-%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D1%8B%D0%B5_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B8,_%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4,_%D0%BB%D0%B5%D0%B2%D0%BE-_%D0%B8_%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4,_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D1%80%D0%B0%D0%B7%D0%B1%D0%BE%D1%80%D0%B0), в которой могут содержаться только правила одного из следующих типов:

*A*→*aγ*

*S*→*ε*

где *a* — терминал, *A* — нетерминал (возможно, стартовый), *S* — стартовый нетерминал (причём он не должен встречаться в правых частях правил), *ε* — пустая строка, *γ* — строка из произвольного числа терминалов и нетерминалов.

**Пример грамматики в нормальной форме Грейбах.**

\begin{align*}
S \; & {\to} \; aST , \\
S \; & {\to} \; aT , \\
T \; & {\to} \; bS , \\
T \; & {\to} \; b 
\end{align*}

**АЛГОРИТМ ПРИВЕДЕНИЯ ГРАММАТИКИ К ОСЛАБЛЕННОЙ НОРМАЛЬНОЙ ФОРМЕ ГРЕЙБАХА.**

**ШАГ 1** . Избавимся от *ε* -правил.

**ШАГ 2.** Применить алгоритм устранения левой рекурсии. Воспользуемся [алгоритмом устранения левой рекурсии](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A3%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D0%B8). Получим грамматику, все правила которой будут иметь следующий вид:

* *Ai*→*aγ*
* *Ai*→*Ajγ*, где *Ai*, *Aj* — нетерминалы, *a* — терминал, *γ* — произвольная последовательность из терминалов и нетерминалов, *i*<*j*.

**ШАГ 3.** Воспользоваться следующей функцией для придания грамматике нужного вида:

**function** greibah(правила *A*1…*An*

из контекстно-свободной грамматики G):

**for** i = n .. 1

**for** j = i + 1 .. n

Для каждого правила вывода из *Aj*

вида *Aj*→*δ*1|…|*δk* заменить каждое правило *Ai*→*Ajγ* на *Ai*→*δ*1*γ*|…|*δkγ* .

После каждой итерации главного цикла все правила для *Ak* (где *k*⩾*i*) будут иметь вид *Ak*→*aγ*. Значит, после применения процедуры все правила грамматики будут иметь вид *A*→*aγ*.



Рис.2 Применение алгоритма преобразования КС-грамматики к ослабленной нормальной форме Грейбаха.

**Применение грамматик в нормальных формах**

**Использование нормальных форм существенно упрощает доказательство теорем.**

Например, использование нормальной формы Грейбах позволяет доказать, что для каждого контекстно-свободного языка (не содержащего *ε*) существует автомат с магазинной памятью без переходов по *ε*.

**Разбор грамматики**

Нормальная форма Хомского позволяет производить разбор грамматики, например, с помощью [алгоритма Кока-Янгера-Касами](https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%BE%D0%BA%D0%B0-%D0%AF%D0%BD%D0%B3%D0%B5%D1%80%D0%B0-%D0%9A%D0%B0%D1%81%D0%B0%D0%BC%D0%B8_%D1%80%D0%B0%D0%B7%D0%B1%D0%BE%D1%80%D0%B0_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B8_%D0%B2_%D0%9D%D0%A4%D0%A5). **Алгоритм** [**Кока**](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BA,_%D0%94%D0%B6%D0%BE%D0%BD)**— Янгера — Касами** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Cocke — Younger — Kasami algorithm*), **алгоритм CYK** либо **CKY** — [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), позволяющий установить, можно ли в заданной [контекстно-свободной грамматике](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BD%D0%BE-%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) вывести заданную строку, и если это так, то предоставить её вывод. Другими словами, это [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [синтаксического анализа](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) строки. Алгоритм относится к классу табличных методов: для анализа цепочки строится треугольная таблица, элементы которой получают значения подмножеств множества нетерминалов. Временная сложность метода n3 – где n – длина входной цепочки. Алгоритм реализует [синтаксический анализ](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) снизу-вверх и основывается на методе [динамического программирования](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Его открыватели: [Джон Кок](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BA,_%D0%94%D0%B6%D0%BE%D0%BD), Дэниел Янгер, Тадао Касами и Джейкоб Т. Шварц. Они использовали восходящий анализ и [динамическое программирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Стандартная версия CYK работает только с контекстно-свободными грамматиками, заданными в нормальной форме (CNF). Однако любая контекстно-свободная грамматика может быть преобразована (после конвертирования) в грамматику CNF, выражающую тот же язык.

В свою очередь, нормальная форма Грейбах позволяет использовать метод рекурсивного спуска, сложность которого является линейной, несмотря на возвраты. **Метод рекурсивного спуска** — алгоритм **нисходящего синтаксического анализа**, реализуемый путём взаимного вызова процедур парсинга, где каждая процедура соответствует одному из правил контекстно-свободной грамматики. Применения правил последовательно, слева-направо поглощают токены, полученные от лексического анализатора. Это один из самых простых алгоритмов парсинга, подходящий для полностью ручной реализации.

Парсер — это программа, анализирующая входное арифметическое выражение. Программы подобного класса, иногда называют так же «распознавателями». Парсинг — **процесс разбора входного арифметического выражения на более простые составляющие**. Результатом работы парсера является сформированное дерево лексем.