Грамматика Хомского

Ниже описывается формализм порождающих грамматик Хомского. Методы задания языка с помощью порождающих грамматик сейчас довольно популярны, особенно для машинной обработки компьютерных языков. Но обычно изучение порождающих грамматик в теории трансляторов заканчивается на контекстно-свободных грамматиках. Последние являются довольно узким специальным классом порождающих грамматик Хомского и обычно используются как вид категориальных грамматик (как конкретно это делается, будет показано ниже) для задания синтаксических анализаторов. Последнее обстоятельство только затуманивает понимание подхода Хомского. Дальнейшее изложение предназначено тем, кому интересно понять, в чем состоит этот подход.

Определение порождающей грамматики

Грамматика представляет собой конечное описание формального языка. Формальный язык, в свою очередь, является произвольным множеством цепочек, составленных из символов некоторого конечного алфавита. Произвольность множества здесь понимается в том смысле, что оно может быть бесконечным, конечным или пустым.

Формализм порождающих грамматик Хомского был введен Ноамом Хомским в конце 50-х годов прошлого века. За короткое время этот формализм обрел необычайную популярность. Некоторое время порождающие грамматики рассматривались как панацея — универсальный подход для задания всевозможных языков, в том числе и естественных (т.е. языков, которые люди используют для повседневного общения между собой). Но время показало, что порождающие грамматики для описания естественных языков не очень удобны. Сейчас порождающие грамматики применяются, в основном, для описания синтаксиса формальных языков, подобных языкам программирования и другим компьютерным языкам.

Порождающая грамматика Хомского задается как множество «правил порождения» (продукций). Каждое правило является просто парой цепочек (`w'`, `w''`) и задает возможность замены левой цепочки на правую при генерации цепочек языка, задаваемого грамматикой. По этой причине, правила обычно записывают в виде `w' --> w''`, указывая конкретно, что на что можно заменять. Множество правил в грамматике должно быть непустым и конечным, и обычно обозначается латинской P.

Цепочки в правилах грамматики могут быть составлены из символов двух алфавитов: алфавита терминальных символов (терминалов) и алфавита

нетерминальных символов (нетерминалов). Алфавит терминалов обозначают через T. Этот алфавит на самом деле совпадает с алфавитом того формального языка, который задает данная грамматика. Смысл термина «терминальный» состоит в том, что в правилах грамматики в левой части не может быть цепочек, которые составлены только из терминальных символов. Поэтому, если такая цепочка получилась в результате подстановки, то следующая процесс порождения цепочки остановится (terminate). Нетерминальные символы используются в промежуточных порождениях цепочек. Смысл нетерминала в задании алгоритма порождения цепочки может быть самый разный и обычно зависит от типа грамматики, в которой этот символ используется. Различные примеры использования нетерминальных символов будут рассмотрены ниже.

Но один нетерминальный символ всегда имеет один и тот же смысл — он обозначает все цепочки языка. Называется этот нетерминал «начальным нетерминальным символов порождающей грамматики» и обычно обозначается посредством латинского S (start или sentence). В каждой порождающей грамматике обязательно должно быть правило, к которого левая часть состоит из единственного начального нетерминала, иначе в данной грамматике нельзя будет породить даже одной цепочки.

Итак, порождающая грамматика Хомского — это четверка `G = {N, T, P, S}`, где

- `N` — конечный алфавит нетерминальных символов.
- `T` — конечный алфавит терминальных символов (совпадает с алфавитом языка, задаваемого грамматикой).
- `P` — конечное множество правил порождения.
- `S` — начальный нетерминал грамматики `G`.

Язык порождающей грамматики

Порождающая грамматика Хомского задает язык посредством конечного числа подстановок цепочек из начального нетерминала грамматики на основе правил порождения. Опишем это чуть более конкретно.

Шаг порождения `w' alpha w'' => w' beta w''` состоит в замене подцепочки `alpha` на подцепочку `beta` в соответствии с правилом порождения `alpha --> beta`. При этом, очевидно, из цепочки `w' alpha w''` получается цепочка `w' beta w''`. Иначе говоря, если имеется некоторая цепочка и некоторая ее подцепочка является левой частью какого-то правила грамматики, то мы имеем полное право заменить эту левую часть правила на правую. Конечная последовательность шагов порождений называется порождением. Нуль или более порождений будет обозначать знаком `=>*`. Обозначение `alpha =>* beta` говорит о том, что цепочка `beta` получена из цепочки `alpha` конечным числом

подстановок на основе правил порождения. В этом обозначении может быть так, что подстановка (порождение) не была применена ни разу, в этом случае цепочка `alpha` совпадает с `beta`.

Итак, язык порождающей грамматики `G = {N, T, P, S}` — это множество цепочек, составленных из терминальных символов и порожденных из начального символа грамматики. Математическая формула такова: `L = {w | S =>* w}`.

Для иллюстрации приведем два простых примера.

Пример очень простого языка

Пусть язык `L` состоит из одной цепочки, которая состоит из единственного символа `a`. Иначе говоря, `L = {a}`. Для порождения цепочки `a` достаточно одного правила `S --> a`. Единственное порождение, которое может быть в данной грамматике — это `S => a`.

Следует заметить, что для этого языка можно было бы ввести еще один нетерминальный символ, скажем, символ `A`, а также правила `S --> A` и `A --> a`. Тогда единственным порождением было бы следующее: `S => A => a`. Так как алфавит нетерминалов грамматики мы выбираем произвольно, становится понятно, что даже для такого простого языка имеется бесконечное множество порождающих грамматик, задающих данный язык.

Язык простых арифметических выражений

Рассмотрим язык `A = {a+a, a+a+a, a+a+a+a, ...}`. Цепочки этого языка представляют собой последовательности символов `a`, разделенных символами `+`. Как задать правила порождения этого языка? Заметим, что каждая цепочка языка начинается с символа `a` за которым идет одна или более цепочек `+a`. Соответственно, возникает мысль сначала породить символ `a`, а затем каждая цепочка языка будет получаться присоединением к этому символу справа одной или более цепочек `+a`. Чтобы отделить эти две стадии порождения друг от друга, введем нетерминальный символ `A`. Тогда, получим грамматику со следующими правилами: `S --> aA, A --> +aA, A --> +a`.

Классификация языков

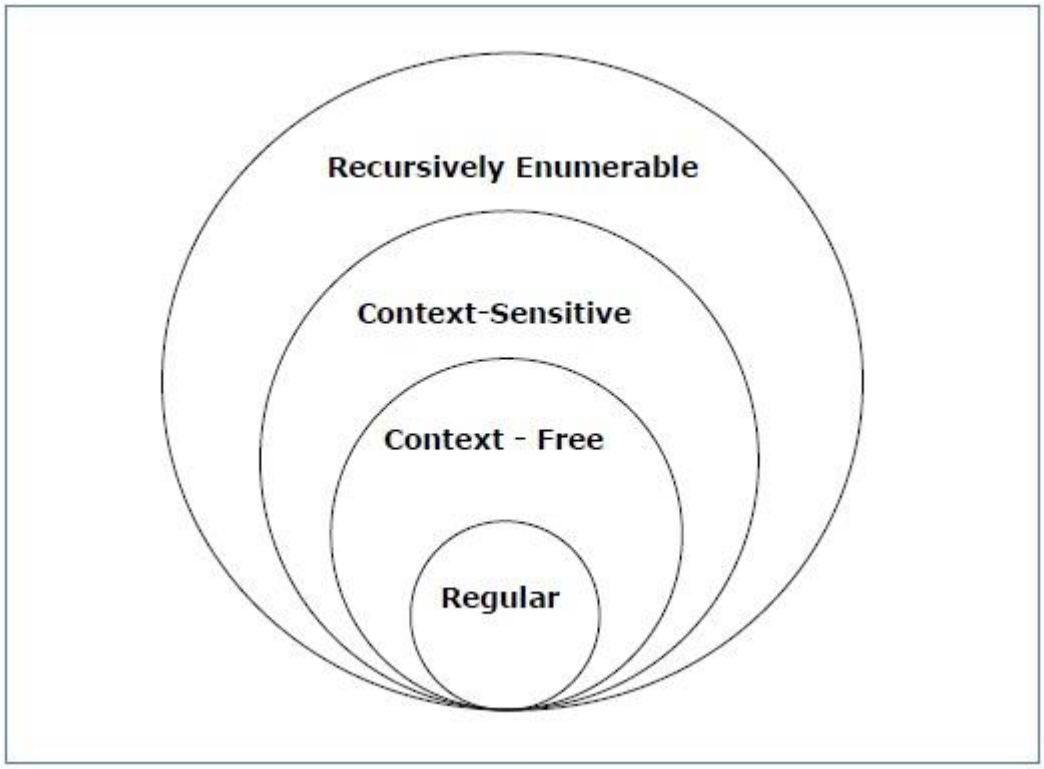
Формальные языки классифицируются в соответствии с типами грамматик, которыми они задаются. Однако, один и тот же язык может быть задан разными грамматиками, относящимися к разным типам. В таком случае, считается, что язык относится к наиболее простому из них. Так, язык, описанный грамматикой с фразовой структурой, контекстно-зависимой и контекстно-свободной грамматиками, будет контекстно-свободным.

Так же, как и для грамматик, сложность языка определяется его типом. Наиболее сложные — языки с фразовой структурой (сюда можно отнести естественные языки), далее — КЗ-языки, КС-языки и самые простые — регулярные языки.

В следующей таблице показано, как они отличаются друг от друга —

Тип грамматики	Грамматика принята	Язык принят	Автомат
Тип 0	Неограниченная грамматика	Рекурсивно перечислимый язык	Машина Тьюринга
Тип 1	Контекстно-зависимая грамматика	Контекстно-зависимый язык	Линейно-ограниченный автомат
Тип 2	Контекстная грамматика	Контекстно-свободный язык	Pushdown автомат
Тип 3	Обычная грамматика	Обычный язык	Конечный автомат

Посмотрите на следующую иллюстрацию. Он показывает объем каждого типа грамматики —



Тип — 3 грамматики

Грамматики типа 3 генерируют обычные языки. Грамматики типа 3 должны иметь один нетерминал с левой стороны и правую сторону, состоящую из одного терминала или одного терминала, за которым следует один нетерминал.

Произведения должны быть в форме **X → a или X → aY**

где **X, Y ∈ N** (нетерминал)

и **a ∈ T** (терминал)

Правило **S → ε** допустимо, если **S** не появляется справа от какого-либо правила.

пример

```
X → ε
X → a | aY
Y → b
```

Тип — 2 грамматики

Грамматики типа 2 генерируют языки без контекста.

Произведения должны быть в форме **A → γ**

где **A ∈ N** (нетерминал)

и **γ ∈ (T ∪ N) *** (строка терминалов и нетерминалов).

Эти языки, генерируемые этими грамматиками, распознаются недетерминированным автоматом.

пример

```
S → X a
X → a
X → aX
X → abc
X → ε
```

Тип — 1 грамматика

Грамматики типа 1 генерируют контекстно-зависимые языки. Продукция должна быть в форме

α A β → α γ β

где **A ∈ N** (нетерминал)

и **α, β, γ ∈ (T ∪ N) *** (строки терминалов и нетерминалов)

Строки **α** и **β** могут быть пустыми, но **γ** должна быть непустой.

Правило **S → ε** допустимо, если S не появляется справа от какого-либо правила. Языки, генерируемые этими грамматиками, распознаются линейным ограниченным автоматом.

пример

```
AB → AbBc
A → bcA
B → b
```

Тип — 0 Грамматика

Грамматики типа 0 генерируют рекурсивно перечислимые языки. Производства не имеют ограничений. Это грамматика любой фазовой структуры, включая все формальные грамматики.

Они генерируют языки, которые распознаются машиной Тьюринга.

Произведения могут быть в форме **α → β,** где **α** — строка терминалов и нетерминалов, по крайней мере, с одним нетерминалом, и **α** не может быть нулевым. **β** — цепочка терминалов и нетерминалов.