Формальные языки и грамматики

Содержание

Цели и задачи курса

Вы уже изучили или получили представление о многих популярных языках программирования: C, C++, Python, Common Lisp, Java... А что нужно, чтобы создать свой язык программирования? Каким минимальным набором элементов необходимо обладать, чтобы получить полноценный язык программирования? Какое аппаратное устройство нужно использовать, чтобы реализовать этот язык программирования, как должно работать это устройство? Как строить наиболее оптимальный код на придуманном языке программирования, учитывая архитектуру аппаратного устройства,на котором происходит программирование? На все эти вопросы в идеале должен отвечать курс «Основы теории языков программирования и методы трансляции».

Нужно однако сказать, что практика пока не соответствует идеалу. Во-первых, необходимо отметить, что возможный язык программирования и техническая платформа (аппаратное устройство) очень тесно взаимосвязаны. Все, что на сегодняшний день хорошо разработано и детально исследовано, касается традиционной архитектуры - архитектуры фон-Неймана. Во-вторых, далеко не для всех формальных языков разработаны эффективные алгоритмы трансляции программ, написанных на этих языках, в инструкции, понятные конечному аппаратному устройству архитертуры фон-Неймана.

Исторические предпосылки

Анализируя принципы и возможности построения абстрактных вычислительных устройств (машин), ученые последовательно наращивали их возможности. В 1940-х и 1950-х годах немало исследователей занимались изучением простейших машин, которые сегодня называются “конечными автоматами”. Такие автоматы вначале были предложены в качестве модели функционирования человеческого мозга. Однако вскоре они оказались весьма полезными для множества других целей, решаемых современными трансляторами. В 1930-е годы, задолго до появления компьютеров, А. Тьюринг исследовал абстрактную машину, которая, по крайней мере в области вычислений, обладала всеми возможностями современных вычислительных машин. Целью Тьюринга было точно описать границу между тем, что вычислительная машина может делать, и тем, чего она не может. Полученные им результаты применимы не только к абстрактным машинам Тьюринга, но и к реальным современным компьютерам.

В конце 1950-х лингвист Н. Хомский занялся изучением формальных “грамматик”. Не будучи машинами в точном смысле слова, грамматики, тем не менее, тесно связаны с абстрактными автоматами и служат основой некоторых важнейших составляющих программного обеспечения, в частности, компонентов компиляторов-трансляторов.

Исторические предпосылки

В 1969 году С. Кук развил результаты Тьюринга о вычислимости и невычислимости. Ему удалось разделить задачи на те, которые могут быть эффективно решены вычислительной машиной, и те, которые, в принципе, могут быть решены, но требуют для этого так много машинного времени, что компьютер оказывается бесполезным для решения практически всех экземпляров задачи, за исключением небольших.

Задачи последнего класса называют “трудно разрешимыми” (“труднорешаемыми”) или “NP-трудными”. Даже при экспоненциальном росте быстродействия вычислительных машин (“закон Мура”) весьма маловероятно, что нам удастся достигнуть значительных успехов в решении задач этого класса.

Все эти теоретические построения непосредственно связаны с тем, чем занимаются ученые в области информатики сегодня. Некоторые из введенных понятий, такие, например, как конечные автоматы и некоторые типы формальных грамматик, используются при проектировании и создании важных компонентов программного обеспечения.

Другие понятия, например, машина Тьюринга, помогают нам уяснить принципиальные возможности программного обеспечения.

Определение транслятора

Предположим, что имеется некоторый язык A, понятный субъекту (устройству) S_A, требуется перевести его в язык B, понятный субъекту (устройству) S_B. Такой процесс называется трансляцией, а устройство («черный ящик»), выполняющее это преобразование называется транслятором.

Из данного определения видно, что возможна широкая трактовка процесса трансляции:

- Процесс перевода китайского текста на русский - транляция, переводчик - транслятор.
- Процесс перевода программы с языка программирования C на язык программирования Python - трансляция, программа, выполняющая такое преобразование - транслятор.
- Процесс перевода программы с языка программирования C на машинный язык - трансляция, программа, которая выполняет такую трансляцию, тоже транслятор, который чаще называют компилятором.

Входной язык A называется *исходным*, входная программа на этом языке - *исходным кодом*. Язык B, на который осуществляется трансляция, называется *объектным языком*, результат трансляции - *объектным кодом*. Устройство, на котором впоследствии оттранслированная программа будет запущена, называется *целевой машиной*.

Виды трансляторов

Как было показано, трансляторы могут выполнять разные задачи. При трансляции языков программирования это замечание также остается верным.
Если задачей трансляции является перевод в язык низкого уровня, например, машинный язык или язык ассемблера, то такой транслятор называется ***компилятором.***
Если задача трансляции состоит в непосредственном выполнении транслятором операторов исходного кода, то такой транслятор называется ***интерпретатором.***
В настоящее время часто используют **смешанные схемы**, когда код частично компилируется, частично интерпретируется: трансляция их выполняется как компиляция исходного кода на некоторый промежуточный код (пи-код, байт-код и т.д.) с последующей интерпретацией полученного промежуточного пи-кода. Фактически прогон такой программы есть интерпретация ее пи-кода. Типичными языками, для которых это оказывается справедливо - Python, Java.
В некоторых случаях стоит задача трансляции, при которой выходом является программа на языке высокого уровня, а входом - программа, написанная на некотором расширении этого языка. Такие трансляторы называются ***препроцессорами.*** *Например, препроцессор обработки макросов.*

Определение формальных языков

Наши определения всевозможных трансляторов остаются неполными, так как совершенно неясно, что понимается под исходным языком, объектным кодом - каким правилам и ограничениям они должны удовлетворять, как проверить, является ли некоторая комбинация символов предложением данного языка. Чтобы ответить на эти вопросы, необходимо строго формально ввести понятие языка, а также рассмотреть правила его задания.

Здесь мы сразу упростим себе жизнь и будем рассматривать очень простые языки, которые легко задать небольшим конечным перечнем простых правил. Ровно такие простые языки мы и будем называть формальными языками, подчеркивая их формальный характер, т.е., как правило, в обычном понимании такие языки не обладают никаким смыслом.

На самом деле вопрос смысла или *семантики* языка - это тоже очень сложный вопрос. Рассматривая формальные языки правильнее сказать, что у них очень простая семантика - примитивный смысл. Итак начнем с ряда формальных определений.

Определение формальных языков

Определение

Алфавитом называется конечное непустое множество. Его элементы называются символами.

Определение

Словом в алфавите Σ называется конечная последовательность элементов из Σ.

Пример

Рассмотрим алфавит $\Sigma = \{a, b, c\}$. Тогда baaa является словом в алфавите Σ.

Определение

Слово, не содержащее ни одного символа (то есть последовательность длины 0), называется пустым словом и обозначается ϵ.

Определение

Длина слова ω, обозначаемая $|\omega|$, есть число символов в ω, причем каждый символ считается столько раз, сколько он встречается в ω.

Пример

$|baaa| = 4, |\epsilon| = 0.$

Определение

Если x, y два слова в алфавите Σ, то слово xy (результат приписывания слова y в конец слова x) называется конкатенацией слов x, y. Если $y = x$, то $xy = xx$ обозначают x^2 и т.д. $x^n = xxx \ldots x (n \text{ раз})$, $x^0 = \epsilon$.

Пример

$ba^3 = baaa, (ba)^3 = bababa$

Определение формальных языков

Определение

Множество всех слов в алфавите Σ обозначается Σ^*. Множество всех непустых слов в алфавите Σ обозначается Σ^+.

Пример

Если $\Sigma = \{a\}$, то $\Sigma^* = \{\epsilon, a, aa, aaa, \ldots\}$, $\Sigma^+ = \{a, aa, aaa, \ldots\}$

Данные определения позволяют весьма просто ввести определение формального языка, как некоторого подмножества над некоторым алфавитом:

Определение

*Если $L \subseteq \Sigma^*$, то L называется **формальным языком** над алфавитом Σ.*

Поскольку формальный язык является множеством, можно рассматривать операции объединения, пересечения, разности языков, заданных над одним и тем же алфавитом.

Определение формальных языков

Над языками также вводят и ряд специфических операций.

Определение

Пусть $L_1, L_2 \subseteq \Sigma^*$, *тогда* $L_1 \cdot L_2 = \{xy : x \in L_1, y \in L_2\}$ *называется конкатенацией языков* L_1, L_2.

Пример

$L_1 = \{a, abb\}, L_2 = \{bbc, c\}$, *тогда* $L_1 \cdot L_2 = \{abbc, ac, abbbbc\}$

Определение

Пусть $L \subseteq \Sigma^*$. *Тогда* $L^0 = \{\epsilon\}$, $L^n = L \cdot L \cdot \ldots \cdot L$(n *раз*).

Пример

Пусть $L = \{a^k b a^l : 0 < k < l\}$, *тогда* $L^2 = \{a^k b a^l b a^m : 0 < k < l - 1, m > 1\}$

Определение

Итерацией *языка* L (*обозначается* L^*) *называется язык* $\cup_{n \in \mathbb{N}} L^n$ (*эта операция также называется звездочкой Клини*).

Примеры формальных языков

1. $\Sigma = \{a, b\}, L_1 = \{a^n b^n : n \geq 0\}$ - это бесконечное множество таких цепочек, которые начинаются символами a, заканчиваются символами b, причем количество символов a, b одинаково.
2. $\Sigma = \{a, b\}, L_2 = \{\alpha :$ в цепочке α количество вхождений a и b равны$\}$. Например, $abababb \in L_2, aab \notin L_2$.
3. $\Sigma = \{(,)\}, L_3 -$ множество правильных скобочных выражений. Например, $(())() \in L_3,))(() \notin L_3$.
4. $\Sigma = \{a, b, c\}, L_4 = \{\omega c \omega : \omega \in \{a, b\}^*\}$ - две идентичные цепочки из символов a, b слева и справа от символа c.
5. $\Sigma = \{$все словоформы русского языка$\}, L_5 -$ русский язык. Например, «Теория формальных языков проста»$\in L_5$, «от столом за по телеге»$\notin L_5$

Из приведенных примеров видно, что иногда язык удается довольно строго определить имеющимися у нас средствами, однако часто это не удается. Нужны специальные механизмы, описывающие смысл правильных выражений языка. Здесь под смыслом понимается факт принадлежности заданного предложения языку. Важную роль в этом направлении сыграл Ноам Хомский.

Идея синтаксически-ориентированного разбора Н.Хомского

Основные идеи Н.Хомского вообще относятся к области лингвистики. В конце 1950 годов лингвисты разрабатывали различные модели естественного языка для решения проблемы автоматического перевода одного естественного языка на другой. Н.Хомский предложил так называемую структурную модель естественного языка, которая рассматривает один узкий аспект языков - их синтаксис и их структуру. Свою модель он построил на основе гипотезы, что все естественные языки столь различны, но в основе их всех лежит некий общий метаязык. Функции порождения и распознавания смысла фраз естественного языка задаются именно правилами этого метаязыка.

Для обоснования своего подхода Хомский стал анализировать предложения с двойным смыслом: «Казнить (,) нельзя (,) помиловать». Ведь если смысл задается только структурой слов в самом предложении, то изменение этой структуры мгновенно меняет смысл. Однако в дальнейшем Хомский обнаружил, что при неизменности внешней структуры смысл предложений все равно может быть разный. Например,

«Порядок сменит хаос» - толи порядок придет на смену хаосу, толи наоборот?

«Он вернулся из командировки в Москву» - куда он вернулся из Москвы, или в Москву?

«Бытие определяет сознание» - кто-кого определяет?

Идея синтаксически-ориентированного разбора Н.Хомского

Представленные примеры показывают, что для извлечения смысла из этих предложений анализа одной лишь их структуры мало. Значит смысл определяется не внешней структурой этих предложений, а структурой выстроенного над этими предложениями метаязыка. Именно разная структура этих предложений на метаязыке придает им разный смысл.
В дальнейшем эта идея трансформировалась и нашла широкое применение в информатике при построении трансляторов, хотя в лингвистике модель Хомского признали не все лингвисты. Для информатики эта идея получила название **синтаксически-ориентированной трансляции**: процесс трансляции выполняется в два основных этапа: 1. модуль, который можно назвать распознавателем, строит структуру входной цепочки, 2. на втором этапе построенная структура используется для генерации выхода, выражающего в той или иной форме смысл входной цепочки.

Способы задания формальных грамматик

Определение языка как подможества множества всех возможных цепочек над конечным словарем является слишком общим и неконструктивным. В основном оно может быть удобно лишь для конечного языка. Для бесконечных языков нужны более выразительные средства - нужно суметь задать бесконечное множество цепочек с комощью какого-либо конечного механизма (алгоритма, устройства, исчисления, набора правил и т.д.). Такие механизмы и называются грамматиками.

Определение

Любой конечный механизм задания языка называется грамматикой.

Существуют два основных способа задания формальных языков.
Первый способ - это конечное множество правил порождения за конечное число шагов правильных (имеющих смысл) цепочек языка, причем эти правила не позволяют построить никакую цепочку, не принадлежащую языку. Такой способ имеет наименование ***порождающей грамматики.***
Второй способ - это задание механизма распознавания, который, получив в качестве аргумента любую конечную цепочку над словарем Σ, за конечное число шагов дает ответ, принадлежит ли эта цепочка определяемому языку или нет. Фактически второй способ - это некий алгоритм, устройство. Такой способ имеет наименование ***распознающей грамматики.***

Способы задания формальных грамматик

В качестве распознающей грамматики может выступать передача на радио «Говорим по-русски». На вход поступает вопрос, правильно ли говорить так: «Я скучаю по Вас». Конечное число экспертов дает ответ: да, нет.
Роль распознающей грамматики могут выполнять и различные технические устройства, например, автомат по продаже газировки должен распознать правильную комбинацию монет, чтобы продать Вам газировку.
В качестве порождающей грамматики для наглядности может выступать диктатор (вождь племени): все, что он говорит, и даже все, что может сказать, считается для племени правильным.
В целом порождающие и распознающие грамматики играют разные, но взаимодополняющие роли. Порождающая грамматика удобна для задания спецификации языка, однако она не является алгоритмом, приводящим к результату за конечное число шагов. Распознающие грамматики наоборот представляются алгоритмом, который оказывается важным для анализа, трансляции цепочек языка в некоторый выход.
Среди способов задания языка с помощью порождающих грамматик особую популярность получили порождающие грамматики Хомского.

Задание порождающих грамматик по Хомскому

В 1956 году американский лингвист Ноам Хомский предложил модель порождающей грамматики, которая весьма удобна для задания искусственных языков. Особенность этой модели состоит в том, что каждой порождаемой цепочке языка эта модель позволяет сопоставить ее структуру.

Определение

Порождающей грамматикой Хомского G называется кортеж $G =< T, N, S, R >$, где:

*T - конечное непустое мнжество (терминальный словарь). Элементы множества T называются **терминальными символами**;*

*N - конечное непустое множество (нетерминальный словарь). Элементы множества N называются **нетерминальными символами**, множества T и N не пересекаются;*

*S - выделенный(специальный) нетерминальный символ $S \in N$, так называемый **начальный символ**;*

R - конечное непустое множество правил (продукций), каждое из которых имеет вид $\alpha \to \beta$, где α, β - цепочки символов из $T \cup N$.

Как грамматика порождает язык?

Языком, порождаемым грамматикой G, называется множество цепочек, состоящих только из терминальных символов и выводимх из начального символа грамматики с помощью правил грамматики.
Рассмотрим, например, грамматику $G =< \{a, b\}, \{S\}, S, \{S \to aSb, S \to \epsilon\} >$. Рассмотрим, какие цепочки терминальных символов можно получать в этой грамматике:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

В целом, как легко догадаться, данная грамматика задает язык $L = \{a^n b^n : n \geq 0\}$.
Рассмотрим еще пример: $T = \{a, b, c\}$, $N = \{S, A, B, C\}$, множество R задано набором правил:

$$S \to ABC$$

$$A \to aA$$

$$A \to a$$

$$B \to b$$

$$C \to Cc$$

$$C \to c$$

Попробуйте догадаться, какой язык порождает эта грамматика?

Как грамматика порождает язык?

Часто правила записывают более компактно, перечисляя все альтернативы замены одного и того же нетерминала (разные правые части правил с одинаковой левой частью) с использованием символа $|$. Так предыдущая грамматика в этом случае может быть записана:

$$S \to ABC$$
$$A \to aA|a$$
$$B \to b$$
$$C \to Cc|c$$

Согласно Хомскому, нетерминалы грамматики представляют собой символы мета-языка, которые играют определенные роли в предложениях языка, а дерево вывода связывает эти роли и их смысловые нагрузки, что позволяет «вычислить» смысл всего предложения.

Нужно отметить, что грамматики могут иметь довольно сложный набор правил, например, $S \to aSbAc; aS \to bbAc; bAc \to B; SbA \to \epsilon; B \to b$. При этом грамматики разной сложности могут порождать один и тот же язык. Это все равно, что выразить одинаковый смысл в лаконичной форме, или в очень путанной и сложной. Соответственно возникает вопрос, как измерить сложность грамматики и сложность языка?

Классификация порождающих грамматик по Хомскому

Одна из важных идей Хомского состояла в том, что можно измерить сложность грамматики по сложности ее правил, а сложность языка по сложности наиболее простой грамматики, которая этот язык может задать. Исходя из этой посылки, Хомский выделил четыре класса грамматик, и соответственно 4 класса сложности языков. Для каждого из классов таких языков впоследствии были найдены (построены) распознающие грамматики (распознаватели). Итак, классификация Хомского включает следующий набор, представленный в порядке убывания сложности (в обозначениях ниже α, β, γ - цепочки из терминальных и нетерминальных символов; A, B - один нетерминальный символ; a - один терминальный символ).

- Грамматики типа 0. Данные грамматики называют **свободными** или неограниченными. Хотя бы одно правило должно быть вида $\alpha \to \beta$, распознаются только машинами Тьюринга.
- Грамматики типа 1. Данные грамматики называют **контекстно-зависимыми** или неукорачивающими. Хотя бы одно правило должно быть вида $\alpha A \beta \to \alpha \gamma \beta$, распознаются линейно-ограниченными автоматами.
- Грамматики типа 2. Контекстно-свободные грамматики. Все правила должны быть вида $A \to \alpha$, распознаются автоматами с магазинной памятью.
- Грамматики типа 3. Автоматные (регулярные) грамматики. Все правила должны быть вида $A \to aB|a|\epsilon$, распознаются автоматами с конечными автоматами.