Глава 2. Формальные языки

Не грусти. Рано или поздно всё станет понятно, всё станет на свои места и выстроится в единую красивую схему, как кружева. Станет понятно, зачем всё было нужно, потому что всё будет правильно.

Льюис Кэрролл. «Алиса в стране чудес»

Когда мы говорили «запись алгоритма», то предполагали, что эта запись сделана на некотором *языке*. Далее, все знают, что алгоритмы записывают на *алгоритмических языках* (а ещё есть и языки программирования, это одно и то же, или нет?). И что же такое вообще *язык*?

Язык относится к очень сложным понятиям, он принадлежит к так называемым знаковым системам (их изучает наука *семиотика*). Мы не будем вдаваться в сложные теории, и постараемся разобраться с этим важным понятием «по-простому». Ну, во-первых, языки бывают разные. Одним из языков у людей является *язык жестов*. Сейчас это и общение глухонемых, и «разговор» размахивающих флагами матросов-сигнальщиков на кораблях, да и привычные всем светофоры на дорогах.

Как Вы думаете, к какому языку отнести «звуковые» жесты, например, стрельбу из пушки в полдень? Или звуки, издаваемые некоторыми светофорами для плохо видящих людей?

Далее в разных областях планеты у людей появилось множество привычных нам *устных* языков. И, наконец, возникли многочисленные *письменные* языки. Ясно, что программистов в первую очередь будут интересовать письменные языки, так как ввод, а особенно редактирование текстов нецелесообразно делать с помощью речевого общения с компьютером. Действительно, попробуйте представить, как бы Вы писали и редактировали свою программу с помощью какой-нибудь системы речевого общения типа Siri, Алисы или Маруси. Впрочем, с появлением современных сложный нейросетей типа LaMDA, которые спокойно проходят тест Тьюринга на разумность ⚠, всё может и измениться. Далее мы, однако, будем рассматривать только письменные языки.

Как Вы знаете, в мире существует много естественных языков (русский, английский, китайский и т.д.). Естественные языки возникли в процессе эволюционного развития человеческого общества, они непрерывно изменяются. Например, в них постоянно возникают новые слова, скажем, на слова «удалёнка» и «санитайзер» всё ещё ругается система проверки правописания даже последних версий редактора WORD. Так, считается, что в английский язык добавляется примерно 1000 новых слов в год (по три слова в день).[1] Другие слова, наоборот, устаревают, постепенно выходят из употребления и, наконец, становятся непонятными большинству людей, говорящих на этом языке. Даже у Пушкина многие фразы сейчас либо непонятны, либо имеют совсем другое значение.[2]

Далее, многие из Вас знают, что существуют и специально придуманные *искусственные* языки. Одни из них относительно просты, типа Эсперанто, или выдуманного Атлайского языка жителей мифической Атлантиды. А вот в эльфийском языке квенья (quenya) из трилогии «Властелина колец» Дж. Р. Толкина существительные изменяются по десяти падежам и четырём числам (единственное, множественное, двойственное и дробное 🙄). Этот язык преподают в некоторых колледжах Англии.

Другие искусственные языки совсем сложны для изучения и применения. В качестве примера можно привести созданный Джоном Кихадой в 2004 году язык Ифку́иль (Ithkuil), в котором 58 букв, 72 падежа и порядка 90 различных звуков ⚠. В этом языке очень развитая семантика, короткой фразой можно *точно* выразить то, на что у «обычного» языка потребует нескольких длинных предложений. Например, один из суффиксов uî, вставленный между согласными корня, задаёт одно из 36 значений, заданных в так называемой матрице корня. На этом языке, в частности, можно излагать сложные философские понятия. Например, показанная слева фраза переводится примерно как «Напротив, я думаю, может оказаться, что эта неровная цепь гор заканчивается

---

[1] Правда, большинство из них это так называемые «портмонто» (portmanteau), слова, образованные путём слияния двух и более слов в одно, типичный пример babysitter.

[2] «Мой дядя самых честных правил, когда не в шутку занемог, то уважать себя заставил, и лучше выдумать не мог». Во времена Пушкина эта фраза понималось так: «Лучшее, что мог сделать не любимый мной и не совсем порядочный дядя, это заболеть и умереть (оставив мне наследство)».

где-то там».

Язык имеет и линейное написание, например, само слово iţkuîl означает «гипотетическая композиция разнообразных высказываний, сосуществующих в кооперативном единстве». В качестве иллюстрации семантической мощи языка обычно приводится слово Onxeizvakcispourboi, которое можно перевести как «Эмоциональное и психологическое состояние, возникающее у человека, когда он слишком поздно осознаёт, что поступки, совершённые им, и вытекающие из них последствия постепенно убили надежды его возлюбленного или возлюбленной на то, что их отношения будут долгими и моногамными». Ифкуи́ль считается самым сложным из искусственных языков, сейчас нет людей, говорящих на нём (включая автора), хотя есть поющая на этом языке рок-группа 🙂).

И, наконец, Вы наверняка уже слышали, что существуют и *формальные* языки, к которым относятся и все языки программирования. Сейчас нам предстоит понять, что же такое формальный язык, и чем он отличается от естественных и искусственных языков.

Сначала вспомним, что мы знаем о (письменных) языках. Каждый язык имеет алфавит – набор различимых друг от друга значков-букв (символов).

Символы, не входящие в какие-либо алфавиты, обычно называют знаками. Это иероглифы, дорожные знаки, рожици эмодзи и т.д. Знаковые системы используют знаки для хранения и передачи информации. Сейчас большинство статических знаков (тех, которые неподвижны и их можно компактно изобразить на бумаге или экране) вместе с «обычными» символами собраны в «большой» алфавит Unicode. В версии Unicode 15.0 2021 года содержится около 149 тыс. символов.

Как и раньше, будем рассматривать только конечные алфавиты, с фиксированным набором символов $A=\{a_1,a_2,\ldots,a_n\}$. Конечную (непустую) цепочку $w=\alpha_1\alpha_2\ldots\alpha_k$ символов алфавита будем привычно называть словом, слова как обычно, читаются слева направо. В заданном алфавите A можно составить бесконечное (но счётное) множество слов, его принято обозначать как A^* (это так называемое рефлексивно-транзитивное замыкание 😊). Итак, каждое слово $w \in A^*$. Как уже говорилось, будем понимать термин «слово» достаточно широко. Ясно, что, если пробел и знаки препинания считать буквами, то привычные нам предложения в книге становятся словами. Если переход на следующую строку текста задать в виде буквы, то вся страница книги будет словом и т.д. Например, привычный нам текстовый файл можно тоже считать словом.

Далее, в языке есть синтаксис. В общем случае это набор правил, позволяющий строить из символов алфавита правильные слова. Здесь мы приходим к важному понятию: некоторые слова из множества всех слов A^* мы считаем *правильными* (принадлежащими нашему языку), а другие – *неправильными* (не принадлежащими языку). Например, в русском языке слово «арбуз» явно правильное, а вот слово «марбуз» – явно не правильное. Другие слова могут вызвать у нас сомнение, например, «трямочка» – правильное слово или нет?

В русском языке синтаксис входит в состав *грамматики*, которая включает в себя синтаксис, морфологию, пунктуацию, орфографию и т.д. В языках программирования от всего этого «богатства» остаётся только синтаксис. Немного отдельно в синтаксисе существуют правила построения так называемых лексем, вскоре мы их изучим. Есть, однако, и такая наука, как теория формальных грамматик, она, в основном, используется при разработке компиляторов.

Можно сказать, что все правильные слова имеют для читающего их какой-то *смысл*, а неправильные мы считаем бессмысленными. Обычно, вкладывая в прочитанное слово какой-то смысл, человек, как принято говорить, *интерпретирует* его, то есть *отображает* это слово на некоторое *понятие* в своем сознании. Однако попытка более строго определить, что такое смысл, сразу наталкивается на серьёзные трудности. Ведь, если для одного человека слово имеет какой-то смысл, то для другого (который, например, его не знает) оно будет бессмысленным. Синтаксис естественного языка не предоставляет нам никаких правил, позволяющих определить, имеет ли произвольно взятое слово смысл (принадлежит ли оно нашему языку), или нет.

Здесь полезно вспомнить один занимательный случай, который произошел на заре развития программ, способных общаться на естественном языке. В 70-е года прошлого века на факультете ВМК МГУ велись работы по созданию программ, «понимающих» (тогда ещё только письменный) русский язык. В частности, эта программа должна была отвечать на вопросы и, следовательно, уметь правильно спрягать глаголы. Например, ей вводят слово «идти», а она в ответ выводит «я иду», «он идёт», «они идут» и т.д. Авторы очень гордились этой своей программой, но тут видят, что у компь-

ютера роятся студенты, и у них нездоровое оживление. Вот студенты вводят глагол «ковать» и получают в ответ «я кую», «он куёт», «они куют». А потом они вводят «глагол» "кровать", и программа начинает его спрягать… 🙂 Понятно, что проблема здесь в том, что правила русского языка не позволяют однозначно определить, является ли некоторое слово глаголом, или же нет.

А вот теперь главное. Формальные языки имеют настолько «строгий» синтаксис, что по нему можно построить распознающий алгоритм, который для любого слова $w \in A^*$ однозначно определяет, принадлежит ли это слово языку (т.е. является правильным), или же нет. Таким образом, некоторый формальный язык L – это просто *непустое*, как-то чётко определённое подмножество всех слов над заданным алфавитом, т.е. $L \subseteq A^*$. Теперь становится понятным, что все языки программирования должны быть формальными языками. Действительно, если компилятор начнёт «сомневаться», является ли предъявленная ему программа синтаксически правильной, или нет, то и программировать (в привычном нам смысле) станет невозможным.

Как уже говорилось, кроме синтаксиса, в языке есть и семантика, это тот смысл, который читающий вкладывает в слово языка (в общем случае в некоторый текст). Мы видели, что для человека семантика является весьма «туманным» понятием, что для одного имеет смысл, то для другого может быть бессмысленным. С этим приходится мириться, ведь, как говорится, все люди разные…

Начиная с середины прошлого века тексты стали читать не только люди, но и программы ЭВМ. Нас, конечно, в первую очередь интересуют тексты на формальных языках, и главное – тексты программ. С синтаксисом программ всё понятно, скажем, компилятор может однозначно сказать, правильная ли (синтаксически) программа, или нет. А как быть с семантикой? Вкладывает ли компилятор в прочитанную программу какой-то смысл? Интуитивно ясно, что, если компилятор и вкладывает в прочитанную им программу смысл, то это какой-то свой, «не человеческий», *формальный* смысл. Сейчас существуют несколько способов задать формальную семантику.[1] При изучении языка программирования высокого уровня мы рассмотрим один такой способ задания так называемой *аксиоматической* формальной семантики.

И, наконец, есть ещё одно интересное для нас свойство языка – прагматика. Проще всего определить прагматику как набор правил, определяющих, в каких ситуациях следует использовать конкретные конструкции языка. Хорошим примером в естественных языках является дипломатический протокол. Например, там чётко определено, что к этому дипломату надо обращаться «Ваше превосходительство», а к другому только «Ваше высокопревосходительство», кому положена красная ковровая дорожка к трапу самолёта и почётный караул, а кому и нет…

В качестве примера прагматики в языках программировании можно привести использование операторов цикла. Во многих языках есть три разных оператора цикла, и в каждом конкретном случае один из них использовать предпочтительнее (прагматичнее), чем другие.

2.1. Описание формального языка

Весь математический формализм является как бы забором, следуя вдоль которого, слепой может уверенно двигаться в намеченном направлении.

Станíслав Лем. «Сумма технологии»

Итак, мы уже определили, что для заданного алфавита A формальный язык L есть просто подмножество всех слов $L \subseteq A^*$. Это подмножество задано настолько строго, что можно построить распознающий алгоритм, т.е. множество L является *разрешимым* (см. разд. 1.7). Как же описать это множество?

---

[1] Более строго, это так называемая *динамическая семантика*, мы в эту тему вдаваться не будем.

	a	b	Λ
q_1	ΛΠq_2	ΛΠq_3	N!
q_2	N!	N!	Y!
q_3	ΛΠq_4	ΛΠq_4	N!
q_4	N!	N!	Y!

В простейших случая, когда язык содержит *конечное* число правильных слов, всё просто. Например, для алфавита `A={a,b}` язык `L={a,ba,bb}` содержит всего три слова. Слева показана распознающая этот язык машина Тьюринга. Машина стирает входное слово, и для правильных слов оставляет в качестве ответа букву `Y`, а для неправильных оставляет слово, начинающееся с буквы `N`.[1]

Понятно, что любые «человеческие» языки содержат конечное число слов. Так, словарь В. Шекспира составляет около 12000 слов, словарь какого-нибудь первобытного племени мог содержать всего около сотни слов. А вот, например, язык героини известного романа Ильфа и Петрова «12 стульев» Эллочки Щукиной состоял всего из 30 фраз (слов): «Хами́те», «Хо-хо!», «Знамени́то», «Парниша», «Поедем на такси́?» и некоторых других, что не мешало ей свободно общаться.

Нам, однако, нужны формальные языки, состоящие из бесконечного числа правильных слов. Действительно, если в языке программирования конечное число правильных программ, то это как-то неправильно 🙂.

Во-вторых, для описания *подмножеств* можно обратиться за помощью к математике, где это часто встречается. Например, язык $L=\{aX \mid X \in A^*\}$ состоит из всех слов, начинающихся с буквы `a`, за которой следует любое (в частности и пустое) слово `X`. Надеюсь, что Вы легко напишите для этого языка распознающий алгоритм в машине Тьюринга или НАМ. В качестве ещё одного примера опишем язык $L=\{a^n b^n \mid n \geq 0\}$. Возведение в степень здесь понимается как «символьное», то есть запись a^4 означает `aaaa`, а запись a^0 означает не единицу, как для числовых значений, а *пустое слово*. Таким образом, во всех правильных словах сначала стоит сколько-то букв `a` (может быть и ни одной), а затем ровно столько же букв `b`.

Напишите распознающий алгоритм этого языка (машину Тьюринга и/или НАМ).

В последних примерах мы пользовались тем, что описываемый язык имеет некоторую *регулярность* в структуре своих слов. Соответственно, пользуясь принятыми в математике обозначениями, множество всех слов языка мы записывали с помощью таких регулярных выражений.

Регулярные выражения (regular expressions), по сути, являются механизмом манипулирования строками символов, этот механизм встроен во многие современные языки программирования, например в PHP, Python, C++ (с 2011 года) и т.д. Регулярные выражения широко используются и в текстовых редакторах, скажем, для поиска. Например, в текстовом редакторе WORD вот такой поисковый запрос `<[A-Z]{3;}>[0-9]{2}` будет искать все слова, которые начинаются не менее, чем с трёх заглавных латинских букв, и заканчиваются ровно на две цифры. Поймите, что так мы описали формальный язык, в который входят только такие слова.

С помощью регулярных выражений, однако, можно описать только простейшие языки. Описать широко распространённые языки программирования таким способом, к сожалению, не получится, в них слишком мало регулярности…🙂

В средине прошлого века, когда стали появляться первые языки программирования, вопрос их строгого описания встал особенно остро. Одним из таких способов стало описание формального языка с помощью так называемых металингвистических формул Бэкуса-Наура (Backus-Naur Form).

2.2. Металингвистические формулы Бэкуса-Наура

Весь мир пронизывает пустота, а она подчиняется формулам – это даёт нам безграничные возможности.

Григорий Перельман

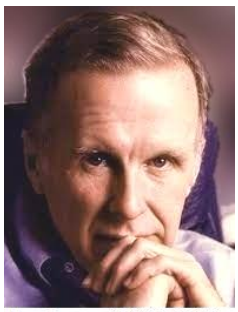
Сначала разберёмся, что значит *описать*, некоторый язык. Ясно, что это можно сделать только тоже на каком-то языке, который называется метаязыком (metalanguage). Конечно, когда нас учили в

---

[1] Обратите внимание, что в *выходном* алфавите ($A_{вых}$) этой машины Тьюринга есть символы `Y` и `N`, которых нет во *внутреннем* алфавите ($A_{вн}$), все символы которого должны быть над столбцами. Естественно, если случайно считать эти символы с ленты, то будет аварийный останов алгоритма.

школе русскому языку, то использовали для его описания тоже русский язык, который был метаязыком для самого себя ⚠. Можно обосновать, что таким свойством обладают все естественные (и искусственные) языки, они могут выступать метаязыками друг для друга. Так, для описания и английского языка, и эсперанто, вполне подойдёт русский язык и наоборот. А вот для описания формального языка это уже не так, его метаязык должен быть *богаче*, чем сам описываемый язык. «Богаче» означает, что в метаязыке есть понятия, которые отсутствуют в описываемом формальном языке. Например, во многих языках программирования можно описывать (конкретные) переменные, но описать, что такое переменная в нём уже нельзя.

Итак, для строгого описания формального языка нам нужен некоторый метаязык. Здесь правда, возникает следующая трудность. Чтобы строго описать формальный язык нам нужен метаязык, который тоже должен быть описан строго, то есть он, в свою очередь, должен быть формальным и описываться на метаметаязыке 🙁, а метаметаязык тоже должен описываться строго и т.д. Как здесь быть? Учёные предложили такой выход. Надо сделать метаязык настолько *простым*, чтобы его описание можно было выполнить на *естественном* языке, и это не вызывало бы у читающих никакого неоднозначного понимания и толкования.


Дж. Бэкус (1924-2007)


Петер Наур (1928-2016)

Металингвистические формулы Бэкуса-Наура (БНФ) являются широко используемым метаязыком, предназначенным для описания широкого круга формальных языков. БНФ была предложена Джоном Бэкусом, он был создателем языка Фортран, одного из первых языков программирования высокого уровня. Ряд улучшений этого метаязыка предложил Петер Наур, он применил БНФ для описания языка Алгол-60, в разработке которого он принимал участие. В дальнейшем язык БНФ был усовершенствован и расширен знаменитым Никлаусом Виртом, автором языка Паскаль, этот метаязык часто называется РБНФ (расширенный БНФ). Именно эту модификацию БНФ мы и будем использовать, для неё существует международный стандарт [7].

> Сначала описание формальных языков с помощью БНФ даже многим учёным (не говоря уже о студентах 🙂) казалось странным и сложным, однако сейчас БНФ рассматривается как простое и элегантное средства описание синтаксиса и широко используется в научных публикациях, поэтому каждый программист должен это знать.

Как и у всякого языка, у БНФ есть свой алфавит. Он включает все символы *описываемого* языка, а также свои собственные символы, среди которых различают *служебные* символы и метасимволы. В качестве служебных используются символ ::= (читается *является* или по *определению есть*),[1] символ вертикальной черты | (читается *или*), квадратные ([ ]), угловые (<>) и фигурные ({ }) скобки, а также многоточие (…) (правда, только после фигурных скобок).

Метасимволы чаще называют метапеременными или *нетерминальными* символами, смысл этих названий вскоре станет понятным, мы чаще всего будем называть их метапеременными. Как и все переменные, метапеременные тоже могут принимать различные *значения*, с точки зрения программиста это *строковые* переменные, их *значениями* являются цепочки символов. В отличие от слов описываемого языка, внутрь цепочки могут входить как символы описываемого языка, так и метасимволы (немного сложно, надо напряься, чтобы понять 🙂).

Каждая метапеременная записывается как произвольный текст, заключённый в служебные угловые скобки, например:

<имя> <правильная программа> <number> <x+y>

Описание формального языка задаётся в виде одной или нескольких метаформул. Каждая метаформула имеет такой вид:

<метапеременная> ::= цепочка$_1$ | цепочка$_2$ | … | цепочка$_n$

Среди всех метаформул особое значение имеет основная (главная) формула, она должна стоять в описании языка первой, порядок следования остальных формул не имеет значения. Метапеременная

---

[1] В некоторых книгах вместо служебного символа ::= используется символ →, но это может быть неудобно, так как его обычно нет на клавиатуре.

основной формулы и должна в конечном итоге принимать значения всех правильных слов описываемого формального языка. Чтобы понять, как всё это работает, начнём разбирать примеры.

В качестве первого примера опишем формальный язык $L=\{aX \mid X \in A^*\}$ всех слов, начинающихся с буквы a:

```
<правильное слово> ::= a<любое слово>
<любое слово> ::= a<любое слово> | b<любое слово> | <пусто>
<пусто> ::=
```

А теперь синтаксис метаязыка БНФ описанный уже на естественном (русском) языке: все метапеременные, входящие в цепочки из правых частей формул должны встретиться в левой части, и ровно по одному разу. Так и есть в нашем примере. Очень просто, не правда ли?

Итак, наш язык описывается тремя метаформулами. Для того, чтобы понять, *в каком смысле* эти формулы описывают этот язык, введём важное понятие механизма вывода. Механизм вывода встроен в БНФ и работает следующим образом. Сначала берётся главная (первая) метаформула, и её метапеременная, стоящая в левой части, заменяется на цепочку, стоящую в правой части. В том случае, когда в правой части стоят несколько цепочек, разделённых спецсимволом |, то случайным образом выбирается одна из этих цепочек. Далее возможны два случая:

1. В полученной цепочке есть метапеременные. В этом случае случайным образом выбирается одна из них, и берётся металингвистическая формула, в которой эта метапеременная стоит в левой части. По синтаксису БНФ такая формула есть и только одна.[1] Используя найденную формулу, метапеременная в цепочке заменяется на одно (случайно выбранное) значение в этой формуле. После этого опять выполняем пункт 1.
2. В цепочке больше нет метапеременных, т.е. это слово описываемого языка. В этом случае механизм вывода останавливается и объявляет полученное слово правильным (принадлежащим описываемому формальному языку).

Теперь понятно, почему метапеременные называются *нетерминальными* символами: пока они есть в цепочке, механизм вывода продолжает работать. А теперь главное – по определению, формальный язык, описываемый заданными метаформулами, состоит из множества слов, которые получаются при всех запусках механизма вывода. Говорится, что мы определяем язык путём порождения (generation) всех правильных слов этого языка.

Приведём пример работы механизма вывода.

```
<правильное слово> ⇒ a<любое слово> ⇒ ab<любое слово> ⇒
   abb<любое слово> ⇒ abba<любое слово> ⇒  abba<пусто> ⇒ abba
```

Можно заметить, что работа механизма вывода в чём-то похожа на работу исполнителя Нормального Алгоритма Маркова, в обоих случаях выполняются операции *поиска* и *замены* слов. Есть, однако, и существенное отличие: механизм вывода не является алгоритмом, т.к. из-за случайного выбора метапеременной и подставляемого значения не выполняется свойство детерминированности. Отметим также, что длина выводимой строки с каждым шагом не уменьшается (за исключение последнего шага с метапеременной <пусто>).

Итак, механизма вывода, используя описание языка, получает из главной метапеременной правильное слово. Эта работа в чём то похожа на развитие живого организма из одной зародышевой клетки по информации, записанной в ДНК. Сам «механизм вывода» при этом находится в живых клетках, это специфический набор белков и рибонуклеиновых кислот (мы ещё мало что знаем о работе этого механизма). Каждый ген при этом является аналогом метапеременной, которая, постепенно «развиваясь», превращается в какую-то конструкцию формального языка. Получается, что составление описания языка в виде метаформул в чём-то похоже на работу генетика, кодирующего в ДНК структуру будущего организма 🙂.

---

[1] Здесь можно отказаться от требования, чтобы метапераменная встречалась в левых частях формул строго *по одному разу*. Просто, если таких формул несколько, случайным образом выбирается любая из них. При этом можно отказаться и от требования, чтобы в правой части формул было *несколько* цепочек, просто описание языка становиться более громоздким.

Любопытно, что нечто, похожее на БНФ и механизм вывода впервые был описан очень давно. По-видимому, это был древнеиндийский ученый Панини (примерно V век до н.э.), он использовал похожий метод для описания языка санскрит.

Итак, набор металингвистических формул определяет некоторый язык $L \subseteq A^*$, т.е. подмножество всех слов в заданном алфавите. Осталось показать, что это именно формальный язык, т.е. для него существует распознающий алгоритм.

В качестве распознающего нам подойдёт так называемый алгоритм рекурсивного спуска (другое его название это *алгоритм полного перебора с возвратом при неудаче*). Принцип работы этого алгоритма легко понять, он позволяет гарантированно определить, является ли какая-либо шахматная позиция выигрышной (при безупречной игре) или нет. Для этого достаточно делать любые (допустимые правилами) ходы, а когда Вам поставят мат или будет ничья, то сказать: «Ошибся, беру ход назад», после чего сделать другой ход. Когда окажется, что из данной позиции все ходы не ведут к выигрышу, надо взять два хода назад, и т.д. Ясно, что при таком алгоритме Вы либо выиграете, либо определите, что все пути ведут к проигрышу или ничьей. Правда, на это может потребоваться время, большее, чем уже существует наша Вселенная 🙂.

Аналогично ходам шахматной партии, в описании формального языка используется конечное число формул, и в каждой из них конечное число метапеременных и способов их замены. Будут две ситуации, когда надо будет «взять ход назад». Во-первых, когда выведенное слово не совпадает с проверяемым, и, во-вторых, когда в процессе вывода получено слово, длина которой *больше*, чем у проверяемого слова (потом эта длина уже не может уменьшаться).

Это свойство неуменьшение длины очень существенно, так у нас не может быть бесконечного применения правил. Для шахматной партии нам нужно похожее правило «отсечения» ветвей, не приводящих в нашему поражению или ничьей, а приводящих к бесконечной игре. Для шахмат это правило, что ничью наступает после троекратного повторения позиции на доске. Так как число фигур на доске не может увеличиваться, то этого достаточно, рано или поздно будет ничья или наше поражение, либо троекратное повторение позиций. Заметим, что, например, для такой игры, как го, где число фигур может увеличиваться, этот алгоритм уже не работает.

Программисты знают, что обычно алгоритм полного перебора с возвратом при неудаче весьма просто реализуется рекурсивной программой, но он очень не эффективный, поэтому для проверки правильности программ компиляторы используют другие, более быстрые алгоритмы.

Заметим теперь, что синтаксически правильное описание некоторого формального языка с помощью БНФ совсем не гарантирует, что этот язык *существует*. По аналогии, программисты хорошо знают, что успешно прошедшая компиляцию (т.е. *синтаксически* правильная программа) не всегда будет *семантически* правильной, т.е. давать требуемый результат, а не, скажем, зацикливаться или делить на ноль. Это же верно и для нашего метаязыка, синтаксически правильный набор метаправил не обязательно определяет какой-то формальный язык. Например, правила

```
<слово1> ::= a< слово2>
<слово2> ::= b< слово1>
```

не определяют никакого формального языка, так как механизм вывода не сможет получить ни одного правильного слова (всегда зацикливается). Можно было бы сказать, что это описание порождает *пустой* формальный язык, но мы специально определили язык как *непустое* множество правильных слов.

Отметим также, что любая метаформула (а не только первая) тоже определяет формальный (под) язык, для этого надо считать её первой (главной) формулой.

В качестве следующего примера опишем через БНФ язык $L=\{a^n b^n \mid n \geq 0\}$. Главную метапеременную назовём $\langle a^n b^n \rangle$ (чтобы было легче читать описание):

```
<anbn> ::= a<anbn>b | <пусто>
<пусто> ::=
```

Убедитесь, что выводятся все правильные слова нашего формального языка (включая пустое слово, которое тоже правильное), и не выводится ни одного неправильного слова.

Как видно, при определении формулы $\langle a^n b^n \rangle$ в её левой части в свою очередь встречается метапеременная с этим же именем. Такое описание языка называется рекурсивным, можно показать,

что при отсутствии в диаграммах такой рекурсии (прямой или косвенной) порождается только *конечное* число правильных слов.

А теперь опишем в алфавите `A={a,b}` язык симметричных слов (палиндромов) которые одинаково читаются слева направо и справа налево:

```
<симметричное> ::= a<симметричное>a | b<симметричное>b |
                   a | b | <пусто>
<пусто> ::=
```

Пример работы механизма вывода:

```
<симметричное> ⇒ b<симметричное>b ⇒ ba<симметричное>ab
               ⇒ baa< симметричное>aab ⇒ baabaab
```

Как и для машины Тьюринга, для БНФ существуют условные обозначения, призванные сделать описание более кратким и понятным. Во-первых, это строка в квадратных скобках, встретив такие скобки, механизм вывода бросает монетку «на удачу» 🙂. При удачном броске текст из скобок вставляется в результат, а при неудачном – просто выбрасывается.

Во-вторых, это текст в фигурных скобках, за которыми может следовать многоточие. Когда многоточие опущено, то содержимое фигурных скобок берётся ровно один раз. При наличие многоточия текст из фигурных скобок вставляется в результат ноль или большее количество раз. Можно считать, что при этом механизм вывода обращается к генератору случайных чисел и получает от него целое неотрицательное число, а потом повторяет содержимое фигурных скобок нужное число раз. Например, если для записи `{ab}…` у генератора случайных чисел выпало значение 4, то берётся текст `abababab`. [1]

Текст внутри фигурных скобок может содержать несколько строк, разделённых вертикальной чертой. В этом случае для работы берётся случайная из этих строк. Возможна комбинация двух последних обозначений. Например, если у записи `{ab | ba | b}…` для многоточия выпало значение три, то сначала фигурные скобки повторяются три раза:

`{ab | ba | b}{ab | ba | b}{ab | ba | b},`

а потом, независимо друг от друга, в каждой скобке выбирается одна из строк. Например, в итоге может получиться строка `abbb`.

С помощью этих условных обозначений язык слов, начинающихся с буквы a $L=\{aX \mid X \in A^*\}$ будет описан одной формулой

<слова языка> ::= a{a|b}…

а язык $L=\{a^n b^n \mid n \geq 0\}$ одной формулой

<$a^n b^n$> ::= [a<$a^n b^n$>b]

Будьте осторожны, вот неправильное описание языка симметричных слов

<симметричное> ::= [{a|b}{<симметричное>|a|b}{a|b}]

Сами поймите, почему это описание неправильное.

Все рассмотренные нами до сих пор формальные языки не предназначались для записи алгоритмов, у них нет исполнителя. А вот если для правильных слов формального языка предполагается исполнитель, то такой язык и будет называться алгоритмическим языком. Машина Тьюринга и НАМ являются примерами таких алгоритмических языков. Исполнители этих языков, однако, нельзя реализовать, так как там есть неограниченное число символов в алфавите, бесконечная лента, любое

---

[1] В некоторых учебниках многоточие опускают, считая, что содержимое фигурных скобок всегда повторяется ноль или большее число раз. Тогда, однако наше {a|b} (т.е. выбор ровно одного раза) приходится обозначать отдельно, скажем, как (a|b), и круглые скобки тоже приходится делать служебными символами. При этом «обычные» круглые скобки (символы описываемого языка) приходится, например, обозначать «как в С» `\(`. Тогда и обратный слеш тоже будет служебным символом, всё это, на мой взгляд, неудобно.

число правил подстановки и т.д. А вот если для алгоритмического языка можно реализовать исполнитель (например, на ЭВМ), то такие языки и называются языками программирования. [1]

Далее перед нами встаёт такой интересный вопрос. Любой ли формальный язык можно описать в виде БНФ? К сожалению, ответ отрицательный. Для примера рассмотрим в алфавите $A=\{a,b,c\}$ *формальный* язык $L=\{a^n b^n c^n \mid n\geq 0\}$. В качестве хорошего упражнения напишите для него распознающий алгоритм (проще всего это сделать в виде НАМ). Описать этот язык с помощью БНФ невозможно (для хорошего усвоения темы попробуйте сделать такое описание).

Дело в том, что по так называемой классификации Хомского, все формальные языки по уровню своей сложности делятся на четыре типа с номерами от 0 до 3, с увеличением номера сложность языка *падает* (это так называемая иерархия Хомского). С помощью БНФ можно описать только языки, начиная со второго уровня (на этом уровне находятся так называемые контекстно-свободные языки), а язык $L=\{a^n b^n c^n \mid n\geq 0\}$ относится к первому уровню (контекстно-зависимых языков).


Ноам Хомски

Ноам (Наум) Хомски (Noam Chomsky), американский учёный (родился в 1928 году), выдающийся философ, психолог, профессор лингвистики Массачусетского технологического института, один из основоположников когнитивизма. Между 1980 и 1990 годами он был 8-м самым цитируемым во все времена и первым самым цитируемым из ныне живущих учёных! Из его работ по классификации *естественных* языков взято разбиение всех *формальных* языков на 4 типа:

0) неограниченные (с так называемой фазовой структурой),
1) контекстно-зависимые,
2) контекстно-свободные,
3) регулярные (вспомним наши регулярные выражения для описания простейших языков).

Более глубоко мы в эту тему вдаваться не будем.

Нас, как программистов, будет больше интересовать вопрос, а к какому уровню относятся известные языки программирования (Python, C, Pascal, Fortran и т.д.). К сожалению (или к счастью), все они относятся к *первому* уровню контекстно-зависимых языков, и поэтому с помощью такого простого метаязыка, как БНФ, описаны быть не могут. Кроме того, с помощью БНФ можно описать только *алфавит* и *синтаксис* формального языка, но не его *семантику*. Как быть?

Ну, во-первых, можно использовать более мощный метаязык, который сможет полностью описать синтаксис (да и формальную семантику) контекстно-зависимого языка.

В качестве примера можно упомянуть так называемый венский метод описания формальных языков (VDL – Vienna definition language) и его улучшенная версия венский метод описания формальных языков (VDM – Vienna development method). Эти методы позволяют описать синтаксис и так называемую операционную семантику формального языка. Такие описания, однако, весьма сложны и имеют большой объём.[2] Всё это требует много времени для своего изучения, и, по существу, предназначено не для программистов, а для разработчиков компиляторов.

Впервые VDL использован для разработки сложного языка PL/1, а VDM для строгого описания языка программирования Ада. Язык Ада был разработан фирмой IBM по заказу министерства обороны США, одним из требований была высокая надёжность, следовательно, язык надо было строго описать. Этот язык назван в честь леди Августы Ады графини Лавлейс, которая считается первой в мире программисткой (на языке низкого уровня). Именно она ввела такие понятия, как «команда (инструкция)», «рабочая ячейка», «цикл», «подпрограмма», «условная передача управления» и другие.

В качестве примера рассмотрим простой метаязык для описания контекстно-зависимого формального языка $L=\{a^n b^n c^n \mid n\geq 0\}$. У него будут метаформулы несколько другого вида, чем у БНФ, и иной механизм вывода. Вот описание языка:

---

[1] Как видим, мы «навели строгость», этим можно пренебречь, используя термины «алгоритмический язык» и «язык программирования» как синонимы.

[2] Группа разработчиков этого метода из компании IBM жила тогда в городе Вена (поэтому и Венский метод). Объём описания языка получился таким большим, что в шутку, метод называли VTD (телефонный справочник города Вена).

```
<a^n b^n c^n> ::= {abc}...
ba ::= ab
ca ::= ac
cb ::= bc
```

Здесь в левой части формул не всегда стоит только одна метапеременная, можно писать любые цепочки. Далее, как мы помним, механизм вывода БНФ останавливается, когда в цепочке уже нет метапеременных. Наш новый механизм вывода останавливается, когда для полученной цепочки неприменимо ни одно правило подстановки (вспомните остановку Нормального Алгоритма Маркова!). Пусть, например, для многоточия в первой формуле выпало число два, тогда получается такой вывод:

$$\langle a^n b^n c^n \rangle \Rightarrow abcabc \Rightarrow abacbc \Rightarrow aabcbc \Rightarrow aabbcc$$

Обязательно поймите, что выводятся все правильные слова, и ни одного неправильного. Ценность этого метаязыка невелика, ведь он создан для описания *конкретного* формального языка.

В качестве ещё одного примера можно упомянуть разработанные Дональдом Кнутом так называемые атрибутивные грамматики. Главным в этих сложных способах описания формальных языков является *усложнение механизма вывода*. Теперь этот механизм не только производит поиск и подстановку строк, но и вычисляет функции, описанные на метаязыке.

Во-вторых, давайте не будем паниковать и разберёмся, что значит утверждение «БНФ не описывает контекстно-зависимые языки»? Оказывается, большая часть конструкций описываемого языка выводится правильно. А вот для остальной части языка все *правильные* конструкции выводятся, однако выводится также и много *неправильных* (ошибочных). И тогда возникла идея описывать все конструкции контекстно-зависимого языка с помощью БНФ, а там, где выводятся неправильные конструкции, прикладывать правило (*семантический фильтр* или *правило статической семантики*), с помощью которого отсекать (отфильтровывать) неверные конструкции.

В качестве примера рассмотрим описание понятия «имя», оно присутствует во многих языках программирования (в качестве синонима часто используется термин «идентификатор»). Неформально имя определяется как последовательность букв и цифр, начинающаяся с буквы. Конечно, надо определить также, что такое буква и цифра. Можно предложить такое описание имени с помощью БНФ:

```
<имя> ::= <буква>{<буква>|<цифра>}...
<цифра> ::= 0|1|2|3|4|5|6|7|8|9
<буква> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|r|s|t|u|v|w|x|y|z|
          A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|R|S|T|U|V|W|X|Y|Z|_
```

Как видим, в состав букв, как и в большинстве языков программирования, включён символ подчёркивания. Правильное это описание или нет? Вообще говоря, это зависит от конкретного языка. Например, в одном из первых языков Алголе-60 и в языке Ада это было правильное описание имени, в вот в большинстве других языков – уже нет.

Дело здесь в том, что в этих языках имя должно иметь *ограниченную* длину. Например, хотя в стандарте языка C++ длина имени и не ограничена, но в реализации на платформе Microsoft длину уже ограничили 2048 символами. Язык Free Pascal в этом смысле «скромнее», длина имени ограничена 127 символами, поэтому приведённое выше определение имени будет *неправильным*. Как здесь быть? Конечно, можно и в БНФ задать имя, длина которого ограничена 127 символами (попробуйте это сделать!), однако такое описание будет очень громоздким и затруднит понимание сути дела. Поэтому мы просто к приведённому выше формальному описанию имени приложим семантический фильтр «Если длина полученного механизмом вывода имени будет больше 127 символов, то считать это имя неправильным». Так мы и будем поступать при описании конструкций языков программирования на БНФ.

2.3. Метаязык синтаксических диаграмм

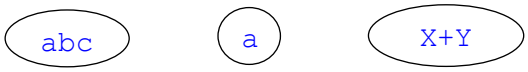
Вы ничего не поймете, пока вы не изучите это более чем одним способом.
Марвин Минский. «Машина эмоций»

В начальный период развития вычислительной техники пользователями ЭВМ и программистами становились математики, физики, инженеры и другие люди, имеющие естественнонаучное образова-

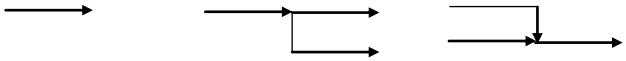
ние. Их было легко обучить понимать формулы БНФ, описывающие формальные конструкции. Однако к средине 70-х годов прошлого века ситуация изменилась. В программирование стали приходить люди с гуманитарным образованием (экономисты, врачи, лингвисты и т.д.). Им было трудно понимать описания языка, выполненные с помощью многочисленных формул. Ориентируясь на таких людей, Никлаус Вирт и создал метаязык синтаксических диаграмм (syntax или railroad diagrams),[1] с помощью которых можно было описывать формальные конструкции более наглядным способом.

Как ясно из названия, описание формального языка состоит из одной или нескольких диаграмм – рисунков (чертежей) на плоскости. По аналогии с БНФ, в синтаксических диаграммах тоже есть свои служебные конструкции. Во-первых, это прямоугольники, внутри которых записываются *имена метапеременных* (метасимволов), например:

Во-вторых, это кружочки или овалы, внутри которых записываются слова описываемого языка, т.е. строки из символов алфавита без метасимволов, например:

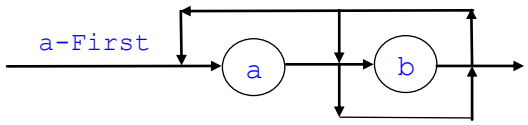


И, наконец, третьим элементом диаграмм являются стрелки. Стрелки похожи на дороги с однонаправленным движением, при этом одна стрелка может разветвляться на несколько, и, наоборот, несколько стрелок могут сливаться в одну, например:



Синтаксис метаязыка диаграмм весьма прост. В каждый прямоугольник и овал входит не менее одной стрелки, а выходит – ровно одна. Кроме того, у каждой диаграммы есть ровно одна (входная) стрелка, которая входит в диаграмму, но ниоткуда не выходит. Над входной стрелкой записывается имя метапеременной, которую описывает эта диаграмма, это имя не принято заключать в угловые скобки или прямоугольник. Далее, у каждой диаграммы есть ровно одна выходная стрелка. И, наконец, для каждой метапеременной в прямоугольниках должна найтись ровно одна синтаксическая диаграмма, над входной стрелкой которой указано имя именно этой метапеременной. Заметим, что с математической точки зрения каждая диаграмма есть связный ориентированный граф с одним входом и одним выходом. Как и в БНФ, одна диаграмма является главной (основной), а её метапеременная над входной стрелкой – главной метапеременной, из которой выводятся все правильные конструкции описываемого языка.

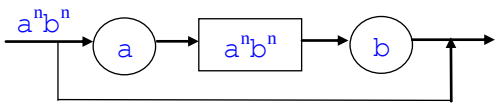
В качестве первого примера опишем синтаксической диаграммой язык $L=\{aX \mid X\in A^*\}$ всех слов, начинающихся с буквы a:



Вспомним, что на БНФ этот язык описывался одной компактной формулой

<a-First> ::= a{a|b}...

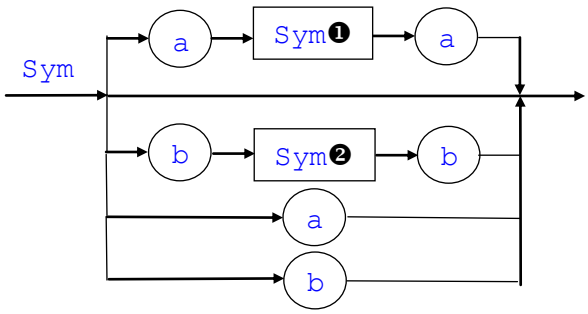
Опишем теперь в виде диаграммы язык $L=\{a^nb^n \mid n\geq 0\}$:



У синтаксических диаграмм свой механизм вывода, который похож на езду по стрелкам и работает следующим образом. Сначала порождаемое правильное слово об.является пустым и наш «автомобиль» въезжает по входной стрелке в главную диаграмму. При проезде через слова в кругах или овалах, эти слова дописываются в конец слова-ответа, а при въезде в метапеременную в прямоугольнике, «автомобиль» появляется на входной стрелке соответствующей диаграммы. При выезде из диаграммы «автомобиль» появляется на выходе метапеременной, в которую он въехал перед этим. Для того, чтобы лучше понять, как это работает, рассмотрим описание языка симметричных слов:

---

[1] Иногда их ещё называют синтаксическими схемами, или синтаксическими графами.

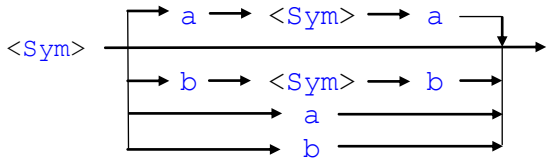


Одинаковые буквы в кружочках будем в процессе вывода для наглядности снабжать индексами a_1, a_2, a_3 и b_1, b_2, b_3. Далее, внутри этой диаграммы есть две одинаковых метапеременных с именем Sym, чтобы отличать их друг от друга, снабдим их значками ❶ и ❷. Будем этими значками отмечать, в какую именно из этих одноимённых метапеременных «въехал», но ещё не «выехал» механизм вывода. Проследим работу механизма вывода. Пусть сначала он один раз проедет по первой ветке (через букву a_1), потом два раза по второй ветви через букву b_1, а потом проедет через одиночную букву a_3:

$$\text{Sym} \Rightarrow a_1❶ \Rightarrow a_1b_1❶❷ \Rightarrow a_1b_1b_1❶❷❷ \Rightarrow a_1b_1b_1a_3❶❷❷ \Rightarrow$$
$$a_1b_1b_1a_3b_2❶❷ \Rightarrow a_1b_1b_1a_3b_2b_2❶ \Rightarrow a_1b_1b_1a_3b_2b_2a_1 \Rightarrow \text{abbabba}$$

Как видим, механизм вывода «выезжает» из метапеременных в обратном порядке, по сравнению со въездом.[1] Отметим, что по сравнению с БНФ, где выводимое слово растёт путём вставки текста в любое место слова (раздвигая остальные буквы), в диаграммах слово растёт, когда буквы приписываются только в конец слова (для человека это более понятно).

Заметим, что во многих книгах авторам лень рисовать прямоугольники и овалы, и используются упрощенная запись, позаимствованная из БНФ. При этом метапеременные записываются не в прямоугольниках, а в угловых скобках, а кружочки или овалы просто опускают. Тогда описание языка симметричных слов упрощается (хоть и становится менее понятным), и будет выглядеть так:



Как видим, синтаксические диаграммы предлагают более наглядный, но и более громоздкий способ описания формального языка, по сравнению с БНФ. В остальном эти два метаязыка эквивалентны. Для решения задач на БНФ и синтаксические диаграммы нужно использовать задачник [11].

Итак, мы описали формальный язык путём порождения (generation) всех его правильных слов. Далее, как мы говорили, для формального языка должен обязательно существовать распознающий алгоритм. Тогда, вообще говоря, можно пойти обратным путём и определить формальный язык путём распознавания (recognition), т.е. язык определяется как множество слов, для которых распознающий алгоритм говорит «да, слово принадлежит языку».

Для языков программирования распознающий алгоритм это *компилятор* с этого языка (точнее, его синтаксический анализатор), он, конечно, необходим, но для описания языка не подходит. Представьте себе, что мы определили некоторый язык программирования как все тексты, которые этот компилятор признаёт правильными ☑. Для естественного языка это всё равно, что составить словарь, в котором для каждого слова будет указано только «правильное» и «неправильное». Читая такой словарь, трудно что-нибудь понять о самом языке.

Вопросы и упражнения

Можно ответить на любой вопрос, если вопрос задан правильно.

Платон, V век до н.э.

1. Чем алгоритмический язык отличается от языка программирования?

---

[1] Обеспечивающая это структура данных называется стеком, о нём мы будем говорить позже.

2. Чем принципиально отличаются друг от друга естественные и формальные языки?
3. Что такое формальный язык над заданным алфавитом?
4. Что такое алфавит, синтаксис, семантика и прагматика формального языка?
5. Что такое распознающий алгоритм формального языка, приведите примеры хорошо известных программистам распознающих алгоритмов.
6. Какие Вы знаете способы описания формальных языков?
7. Что такое метаязык и для чего он нужен?
8. Является ли метаязык формальным языком?
9. Что такое нетерминальные символы метаязыка?
10. Почему набор метаформул определяет именно формальный язык?
11. Какие слова входят в формальный язык, описанный с помощью БНФ?
12. Все ли формальные языки можно описать с помощью БНФ?
13. Чем отличаются контекстно-свободные формальные языки от контекстно-зависимых?
14. Какие формальные языки называются алгоритмическими?
15. Что такое семантический фильтр и для чего он нужен?
16. Чем метаязык синтаксических диаграмм отличается от БНФ?
17. Чем механизм вывода БНФ отличается он механизма вывода синтаксических диаграмм? Когда механизм вывода останавливается?
18. В алфавите $A=\{a,b\}$ опишите язык, в словах которого одинаковое количество букв a и b.
19. В алфавите $A=\{a,b,c\}$ опишите язык $L=\{a^{n+2},b^{k+1},c^n \mid n\geq 0, k\geq 0\}$, где целые n и k принимают значения независимо друг от друга.
20. Какой язык описывают в алфавите $A=\{a,b,c\}$ показанные ниже метаформулы ?

```
<язык> ::= <язык>a<язык>b<язык>c |
           <язык>a<язык>c<язык>b |
           <язык>b<язык>a<язык>c |
           <язык>b<язык>c<язык>a |
           <язык>c<язык>a<язык>b |
           <язык>c<язык>b<язык>a | <пусто>
```