Обработка естественного языка на Прологе

Дж.Газдар, К.Меллиш, перевод:Чесебиев И.А.

2009 г.

1

2

3

4 Грамматики.

4.1. Грамматика как представление знаний.
4.2. Слова, правила и структуры.
4.3. Представление простых грамматик на Прологе.
4.4. Выделение подкатегорий и использование свойств.
4.5. Грамматика определенных простых предложений.
4.6. Классы грамматик и языков.

В "лице"ATN у нас было чрезвычайно гибкое средство для конструирования анализаторов текста на естественном языке, но эта гибкость достигалась ценой ее процедурного характера в ущерб описательной ясности и вызывало зависимость от типа приложения. В этой главе мы рассматриваем преимущества систем обработки языка, основанных на декларативном описании, грамматику, относящуюся к языку, которая нейтральна по отношению к предполагаемому приложению. Представлены концепции грамматик контекстно-свободных фразовых структур (CF-PSG); система записи PATR используется, чтобы показать, как такие феномены, как выделение подкатегорий и неограниченные зависимости могут быть элегантно описаны в виде декларативного формализма.

Наконец, рассматривается отношение CF-PSG к другим классам грамматик.

Грамматика как представление знаний.

Описание распознавателя или системы грамматического разбора для языка - это в некотором смысле описание этого языка, но нужно, чтобы оно было понятным без привлечения каких-либо средств. Более того, это могло бы быть определение для конкретной специфической разработки, но разработки - даже разработки языков программирования - чтобы это было хорошо понятно,- могут отличаться значительно и неожиданно. По этой причине в последние годы ученые-компьютерщики ушли от процедурных описаний семантики навстречу декларативному описанию денотационных семантик. Довольно похожих точек зрения мы придерживаемся, когда рассматриваем написание программ, осуществляющих ввод текста на естественном языке: и с точки зрения синтаксиса, и с точки зрения семантики мы нуждаемся в безопасном определении естественного языка (или некого приближения к естественному языку), который мы обрабатываем, если у нас есть какие-либо идеи о том, как система должна вести себя в зависимости от широкого диапазона условий.

Язык может рассматриваться как множество, членство в котором точно определяется при помощи правил.

Множество составных лингвистических выражений в естественном языке не конечно, поэтому мы не можем все их перечислить (например, "медленная машина "очень медленная машина "очень-очень медленная машина . . .). Насколько известно, ни один из естественных языков не является конечным. Диапазон конструкций, которые делают язык бесконечным, обычно довольно велик. В английском языке слова типа "and"позволяют неограниченному числу фраз соединяться вместе и зависимые простые предложения могут содержать глагольные фразы, которые содержат именные фразы, которые содержат зависимые простые предложения, которые. . .

То, что нам нужно, - это формальные (в смысле "математические") системы, которые определяют членство в бесконечных множествах лингвистических выражений и сопоставляют некую структуру каждому члену таких множеств. Такие формальные системы и есть грамматики. С этой точки зрения, различные виды сетей переходов, которые мы рассматривали в предыдущих главах, - это грамматики. Однако слово "грамматика"часто используется в значительно более ограниченном смысле в работе с естественным языком, чтобы сослаться на формальные системы, отвечающие не только критериям, отмеченным выше, но, вдобавок, и двум следующим критериям.

Во-первых, грамматики в смысле, обсуждаемом в этой главе, выражаются посредством декларативного грамматического формализма, который содержит информацию только о том, какие объекты комбинируют вместе и какими свойствами обладает результирующий объект, формализма, который не содержит внешней процедурной информации о том, как соединять объекты вместе, так как управляющая информация неявно закодирована в сетях переходов.

Во-вторых, грамматики, как мы будем их представлять, прозрачно обеспечивают каждую допустимую строку неким неявным структурным описанием, без необходимости явного описания, как эти структуры строить, как это требовалось, например, в ATN.

Как и FSTN и RTN и в отличие от многих ATN, грамматики, которые

мы представим, прямо устанавливают порядок элементов в строке вместо того, чтобы пытаться воссоздать некий гипотетический основополагающий порядок. И, так же как RTN, наши грамматики будут использовать простую рекурсию для преодоления структурной сложности. Грамматики, как мы будем их представлять, декларативны и, по большей части, основаны на разложении синтаксических категорий (грубо говоря, "частей речи") на компоненты, известные как свойства. Одно из достоинств таких формализмов станет ясным только когда мы рассмотрим семантики в главе 8: грамматики этого типа поддерживают композиционный подход к пониманию, одно из составляющих которого в том, что каждое правильно сформированное выражение имеет свой смысл, который может быть получен как композиция из смыслов выражений, из которых оно состоит. В таком контексте синтаксические структуры, определенные грамматикой для выражения, являются ключевым элементом определения его смысла.

С точки зрения обработки естественного языка, изучение грамматики - это отрасль представления знаний: грамматика - это просто путь представления определенных аспектов того, что то, что мы знаем о языке, явно и формально достаточно, чтобы быть понятым машиной. Рассмотрим по аналогии намного более простую область представления знаний - представление масс: она включает математические вопросы (какую подходящую систему исчисления использовать - целые, вещественные, рациональные, комплексные числа,. . . ?), вопросы записи (должны мы использовать арабские или римские цифры, точки на графе,. . . ?) и детализированные вопросы описания. В этой главе мы исследуем эти три вида вопросов применительно к грамматикам естественного языка.

Один важный математический вопрос, который интересует лингвистов последние 30 лет - это вопрос о том, насколько мощная формальная система нужна для того, чтобы описать любой естественный язык.

Мы уже выдели один пример этой проблемы в главе 2, где мы отметили невозможность для FSTN распознавать точно строки языка типа a\un\db\un\d , то есть языка, состоящего из строк, содержащих некоторое число a, после которых идет то же число b. A RTN, рассмотренная в главе 3, может распознавать в точности строки a , но не a . Есть несколько хороших причин, почему компьютерные лингвисты должны заботиться об этой проблеме. Если, например, мы используем RTN для описания нашего языка и встречаем конструкцию языка a , то мы понимаем, что время, потраченное на попытки заставить нашу RTN распознать эту конструкцию было потерянным. Или, предположим, что у нас есть доступ к аппаратному обеспечению, которое может использовать FSTN сверхбыстро и обнаружили, что в действительных появлениях конструкций a число n никогда не достигает трех - в этом случае могли бы принять решение собрать нашу RTN в FSTN с верхней границей n=3 (такая сборка возможна для любых конечных верхних границ со значением n). Знание основополагающей математики может помочь избежать потерь времени и, иногда, даже предложить подсказки. Мы рассмотрим некоторые из основных математических вопросов, чтобы работать с мощностью несколько более детально в заключительных секциях этой главы.

Для академических лингвистов первичным критерием в определении типа грамматики всегда была ее способность схватывать значимые обобщения внутри грамматики языка и по отношению к грамматикам других языков. Однако, выражение значимых обобщений - это скорее вопрос записи, а классы грамматик, взятые как множества математических объектов, обладают свойством независимости от способов записи, что может быть использовано для их определения. Таким образом, класс грамматик определяет множество языков, которые могут быть описаны с помощью его членов, множество единиц структурного анализа, которые могут быть соотнесены с предложениями в этих языках, и так далее.

Как и академическому лингвисту, лингвисту компьютерному нужно знать, чему в действительности подобен естественный язык на языке математики. Но, вдобавок, компьютерному лингвисту полезно знать, на что приблизительно похожи языки, или на что больше всего похожи, и проявляют ли конкретные языки (такие как английский или японский) определенные математические свойства. Эти последние вопросы находятся вне интересов академических лингвистов. Предположим, например, что суахили обладает определенным признаком, позволяющим отнести его математически к типу языков Y. Но далее предположим, что эти признаки статистически довольно не общеупотребительны в повседневном использовании, так что 99,5 такого использования может быть грамматически разобрано исходя из предположения, что суахили принадлежит математически более ограниченному классу и имеет более простой для компьютерной обработки тип X. Предположим, наконец, что мы имеем работающую систему грамматического разбора, основанную на грамматике X для суахили, справляющуюся с 85 процентами предлагаемых входных данных , которая по многим практическим причинам (некорректный ввод, вариации диалекта, нестандартное произношение, подражательные стилистические формы,. . .) не может быть улучшена далее (85% в действительности довольно хорошая цифра в практическом контексте). Ясно, что при таких условиях, переключение на систему грамматического разбора, основанную на грамматике типа Y, могло бы просто не стоить усилий, поскольку при прочих равных условиях оно в лучшем случае дало бы эффект улучшения с 85% до 85,5%. Возвратимся теперь к проблемам способа записи. Разработка и выбор грамматических формализмов для обработки естественного языка - это тема, привлекавшая к себе пристальное внимание в 80-е годы, и в них был достигнут большой прогресс. По меньшей мере, подходящими являются следующие общие критерии.

(1) лингвистическая естественность,

(2) математическая мощность, и

(3) удобство компьютерной обработки.

Во-первых, тем, кто использует такие формализмы, нужен способ записи, который позволяет им и поощряет их расшифровывать свои лингвистические описания в манере, удобной для понимания и изменения, согласно с их способом думать о языке, и выражает подходящие обобщения для этой области знаний. Например, если все глаголы языка во всех временных формах появляются в начальной позиции VP, формализм должен позволять нам сказать в точности это, а не заставлять нас делать 40 утверждений, по

одному для каждого подкласса глаголов, или вызывать какую-то неуклюжую многоречивость, чтобы сослаться на все слова, показав их разделение в зависимости от конкретных окончаний, многоречивость, которая появляется как только мы пытаемся выбрать все временные формы всех глаголов. Во-вторых, если мы решили, что естественный язык относится математически к типу Y и хотим, чтобы грамматики отразили этот факт в виде записи, описание грамматики должно быть способным выражать грамматики типа Y.

Как будет видно в заключительных секциях главы, некоторые ограничения на способ записи могут иметь эффект очень серьезных ограничений класса грамматик, которые могут быть выражены. Наоборот, очевидно незначительные ограничения в способе записи могут радиально повысить потенциальную математическую мощность характеризуемых систем. Мы можем не хотеть, чтобы это случилось, потому что желаем оградить пользователей формализма от выдумывания неестественных способов анализа, или по причине вычислительной неэффективности, к которой мы сейчас перейдем.

Грамматический формализм, используемый академическими лингвистами, читается обычно только другими академическими лингвистами. Но компьютерный грамматический формализм, такой как язык программирования, должен быть читаемым и понятным и для людей и для машин. Более того, определяя типичные цели обработки естественного языка, мы хотим, чтобы машины имели возможность понимать и использовать формализм за реалистичное количество времени.

В самом деле, проблемы, которые поднимаются в разработке грамматических формализмов, в точности те же, что поднимаются в разработке любых декларативных компьютерных языков для представления знаний в некоторой конкретной области.

Некоторые существующие грамматические формализмы предлагались для выражения лингвистических теорий. В таких формализмах самоограничения часто возникали из эмпирических требований природы естественного языка. Другие были разработаны для использования в качестве инструментов для академических и компьютерных лингвистов, таковые обычно свободны от налагаемых на способы выражения ограничений.

Формализмы, которые мы будем использовать в этой главе (и впоследствии), чтобы представить, что такое в основном грамматики контекстно-свободной фразовой структуры, - это Грамматика Определенных Простых Предложений (DCG) и PATR, и оба этих формализма принадлежат к последнему классу.

PATR должен, в ближайшие годы, стать потенциальным общеупотребительным языком в работах по обработке естественного языка, многие другие грамматические формализмы могут быть переведены на него.

Все виды грамматик, которые использовались в компьютерной лингвистике задействовали, в той или иной форме, следующее:

(1) Представление синтаксических категорий или "частей речи".

(2) Тип данных для слов (и, следовательно, лексикон, словарь или список слов).

(3) Тип данных для синтаксических правил.

(4) Тип данных для синтаксических структур.

Грамматика как целое может быть рассмотрена как определенный род типа данных, скомбинированных из первых трех элементов. Система грамматического разбора, таким образом, - это алгоритм, принимающий такой тип данных, совместно со строкой, и пытающийся вернуть один или более объектов типа данных "синтаксическая структура".

Так полный грамматический формализм обеспечивает, по крайней мере, язык для определения синтаксических категорий, язык для представления лексических входов, язык, на котором записываются правила (возможно, правила более, чем одного типа) и язык для представления синтаксических структур. Эти языки могут быть различными, или один или более из них могут совпадать.

Слова, правила и структуры.

Лексикон - это как минимум список слов, который сопоставляет каждому слову его синтаксические свойства. Самое важное из этих свойств - это их синтаксическая категория - например, глагол это слово или существительное. Вдобавок, в зависимости от сложности полной грамматики, лексикон будет содержать информацию типа подкатегории слова (такой как, является ли этот глагол переходным или непереходным), другие синтаксические свойства (такие как, род существительного в языках, где существует разделение по родам), и, возможно, также морфологическую и семантическую информацию. В самых простых грамматиках лексические входы могут выглядеть вроде этого(используя способ записи PATR):

Слово paid:

<cat> = V.

что просто сообщает нам, что "paid"является словом категории "глагол в то время как немного более усложненная грамматика может потребовать лексического входа типа:

Слово paid:

<cat> = V

<tense> = past

<arg1> = NP.

что сообщает, вдобавок, что глагол находится в прошедшем времени и что он переходный, так как <Аргумент1>=<Именная Группа> будет означать переходность, что будет объяснено позже.

Характер синтаксических правил изменяется значительно в зависимости от точного характера привлекаемой грамматики, и часто такие теории позволяют использовать более, чем один вид правил. Один из очень часто используемых типов синтаксического правила - это правило фразовой структуры, и большинство текущих построений компьютерной лингвистики предполагают существование таких правил. Правило фразовой структуры просто сообщает, что отдельная синтаксическая категория, стоящая в левой части правила, может быть получена некоторым сочетанием в правой части. Так, "Предложение->ИменнаяФраза ГлагольнаяФраза"сообщает нам, что предложение может состоять из именной фразы, за которой следует глагольная фраза. Если вы трактуете стрелку ->"как удобное сокращение для выражения "может состоять из когда смотрите на правила фразовой структуры, вы не далеко от истины. Правила фразовой структуры могут быть также использованы, чтобы вводить лексические единицы; так правило типа "Глагол->paid это просто альтернативный путь представления информации которую мы закодировали как:

Слово paid:

<cat> = V.

В принципе тогда мы могли бы на этом этапе, если бы мы хотели, объединить наши правила и наш лексикон в один компонент, поскольку оба компонента используют правила фразовой структуры в этом примере. На практике, однако, мы будем иногда обращаться с лексическими входами как с грамматическими правилами, а иногда рассматривать их как отдельный компонент, в зависимости от того как будет более понятно в данный момент.

Итак, мы обсудили грамматику абстрактно, даже без представления ее примера.

В этой и следующих секциях мы посмотрим на некоторые детали того, на что похожи грамматики и как они работают. Для начала давайте рассмотрим очень простую грамматику (Грамматику 1), использующую 4 синтаксические категории, а именно: именную фразу, предложение, глагольную фразу и глагол. Мы снабдим грамматику только тремя правилами: одно сообщить нам, что предложение состоит из именной фразы и следующей за ним глагольной фразы, другое позволит глагольной фразе состоять из глагола и именной фразы, и третье позволит глагольной фразе состоять из отдельно стоящего глагола. Нам также нужен лексикон, чтобы обеспечить нас готовыми к использованию глаголами и именными фразами. Мы представим, что мы начинаем построение языкового интерфейса для базы данных больницы, поэтому все наши лексические единицы будут взяты из этой области.

Грамматика 1

Правило формирование простого предложения

S -> NP VP.

Правило переходный глагол

VP -> V NP.

Правило непереходный глагол

VP -> V.

Слово Dr. Chan:

<cat> = NP.

Слово nurses:

<cat> = NP.

Слово MediCenter:

<cat> = NP.

Слово patients:

<cat> = NP.

Слово died:

<cat> = V.

Слово employed:

<cat> = V.

Лексикон сообщает нам, что "Dr.Chan "nurses "MediCenter"и "patients-именные фразы, в то время как "died"и "employed глаголы.

Грамматики, подобные Грамматике 1, называются грамматиками контекстно-свободной фразовой структуры (CF-PSG). Ключевой признак CF-PSG - привлечение конечного множества грамматических категорий и конечного множества правил для определения того, как категории левой части правила могут быть реализованы как последовательности элементов правой части. Элемент правой части может быть или категорией или отдельным символом языка. Когда правило CF-PSG определяет, что элемент левой части правила может быть реализован как отдельный элемент правой части, эта реализация должна пониматься безотносительно к контексту, в котором появляется элемент правой части. Правило не делает никаких ограничений на контекст, в котором это может случиться - следовательно, термин контекстно-свободен. Существует несколько возможных вариантов записи для CF-PSG, одна из

них, нормальная форма Бэкуса-Наура, -использовалась для языков программирования, но мы используем (подмножество) записи PATR для Грамматики 1.

Теперь мы полностью определили Грамматику 1. Но для чего она подходит? Что она может делать? Грамматика имеет две функции.

Первая - определить множество грамотных строк слов в рассматриваемом языке (или, более реалистично, в части языка). О таких строках говорят, что они "порождены"этой грамматикой. Такое использование порождения слов не предполагает существования каких-либо действительных процедур, которые производят предложения; оно показывает только, что определенные строки разрешены как грамотные этой грамматикой.

Так, в случае естественных языков типа английского, грамматика должна определять множество грамотных предложений, множество грамотных глагольных фраз, и так далее. При определении множества грамотных строк косвенно также определяются неграмотные строки, поскольку неграмотная строка - это просто строка, не являющаяся грамотной, то есть строка, которая не "порождается"этой грамматикой. Вторая функция, которая есть у грамматики, - это связать одну или более структур с каждой грамотной строкой. Эта синтаксическая структура фразы может быть понята как вид подтверждения того, что фраза грамотная. Оно показывает различные части фразы и способы, которыми комбинация этих составляющих допускается правилами грамматики. Как мы увидим, это разбиение фразы на части - ключевой фактор в вычислении семантики фразы. Обычно, если данная строка имеет две различные структуры, оно также будет иметь два разных смысла, и, следовательно, будет омонимичной. Часть работы грамматики, таким образом, состоит в том, чтобы делать корректные предположения об этом виде структурной омонимии.

Какую форму принимают эти структуры? Ответ на математическом языке - направленные ациклические графы: все современные грамматические построения, без исключения, задействуют некоторый вид ориентированного ациклического графа для представления структуры. Здесь мы не будем углубляться в теорию графов, однако, и ориентированным ациклическим графам в любом случае будет уделено больше внимания в главе 7. Удобно, что большинство, хотя никоим образом ни все, современные лингвисты выбрали для использования один отдельный вид направленного ациклического графа - а именно, свободный - для своих структурных описаний. И он выглядит наподобие дерева: он имеет корень, ветви и заканчивается листьями. Но компьютерные лингвисты, как и специалисты в генеалогии, условно представляют свои деревья сверху вниз, с корнем, утыкающимся в небо и листьями на земле. Рисунок 4.1 показывает такое дерево.

Как Грамматика 1 определяет множества строк и ассоциирует такие строки со структура, подобными показанной на рисунке 4.1? Рассмотрим вершину дерева: здесь у нас есть "Предложение которое имеет дочерние узлы "ИменнаяФраза"и "ГлагольнаяФраза появляющиеся именно в таком порядке, и более ничего. Первое правило Грамматики 1 говорит, что предложение состоит из именной фразы, за корой следует глагольная фраза, и это в точности то, что у нас здесь есть. Таким образом, грамматика узаконивает

эту часть структуры. Если мы теперь обратим наше внимание на глагольную фразу, мы снова обнаружим, что грамматика содержит правило, которое относится к этой части структуры. Наконец, мы смотрим на лексикон и находим, что и "MediCenter"и "nurses"являются именными фразами, как того требует дерево, и что 'employed' - это глагол. Таким образом, каждая подструктура в дереве относится к некоторому правилу в грамматике, и таким образом это дерево допускается данной грамматикой. Раз это дерево является допустимой структурой, то строка слов сделана из листьев этого дерева - а именно 'MediCenter employed nurses' - это, согласно Грамматике 1, грамотное выражение категории "Предложение". Это не единственная строка слов, которую Граммаика 1 допускает как предложение - есть еще 39 таких строк, включая 'Nurses died', 'Dr.Chan employed patients' и так далее.

Выше мы представили наше пояснительное дерево графически, но подобные диаграммы, учитывая ограничения современных компьютеров, не являются удобными для использования в обработке естественного языка. Наиболее простой путь работы с деревьями состоит в том, чтобы манипулировать ими как списками, в которых первый элемент - это корневая категория, а последующие элементы - дочерние поддеревья в порядке появления. Так дерево в верху должно быть таким:

```
[s,[np,'MediCenter'],[vp,[v,employed],[np,nurses]]]
```

Этот вид списка мог бы появиться в красиво напечатанном формате таким образом:

```
[s
    [np
        'MediCenter']
    [vp
        [v
            employed]
        [np
            nurses]]]
```

Общий принцип, использованный здесь состоит в том, что дерево представлено списком, чей первый элемент - верхняя метка, и чьи последующие элементы - непосредственные поддеревья по порядку. Эти элементы сами затем представляют из себя списки, представляющие поддеревья соответственно с теми же правилами.

В этой схеме, листовой узел дерева рассматривается как дерево с меткой но без непосредственных поддеревьев. Таким образом, он может быть представлен как список с одним элементом - меткой.

Альтернативным и эквивалентным представлением дерева является выражение:

s(np('MediCenter'),vp(v(employed),np(nurses)))

Мы закончил эту секцию несколькими комментариями о подходящем способе оценки отдельной грамматики для языка (фрагмента языка), дав грамматический формализм и основополагающую математическую интерпретацию для этого формализма. Есть три основных эмпирических критерия:

(1) Достаточно ли она порождает?
(2) Не порождает ли она избыточно?
(3) Подходящие ли структуры сопоставляются строкам, которые она порождает?

Еще раз, мы используем здесь слово "порождать"в нейтральном смысле, просто обсуждая адекватность грамматики как описания множества строк и игнорируя процедурную сложность, которая была бы привлеченная, если бы мы пожелали произвести действительный пример предложения из описания.

Грамматика для английского языка порождает недостаточно, если есть некоторые синтаксически правильно оформленные выражения, которым грамматика не сопоставляет структуру. Но множества правил, которые предлагают составители грамматики, редко предназначены для обобщения естественного языка в целом и, таким образом, вопросы недостаточного порождения обычно рассматриваются в связи с целями грамматики. Если грамматика как множество правил предназначена для управления всеми английскими простыми предложениями в составе сложного и все еще ошибается в сопоставлении структур для "that saw you"в "the man that saw you то эта грамматика очевидно с точки зрения опыта неадекватна по причине недостаточного порождения. В вычислительном контексте, недостаточное порождение грамматикой не обязательно является проблемой, если цель состоит в том, чтобы произвести подходящий язык; с другой стороны, это могло бы оказаться фатальным, если цель - распознать или понять.

Грамматика порождает избыточно, если она допускает строки, полученные из сопоставления им категорий посредством этой грамматики, которые не могут быть истолкованы как грамотные выражения языка. Это выглядит более простым, чем критерий достаточного порождения, поскольку по-видимому цели составителей грамматики не могут быть увязаны с этим критерием. Однако два фактора делают его менее простым, чем он кажется на первый взгляд. Во-первых, наша интуитивные суждения в вопросе, что такое грамотно или не грамотно, часто трудно отделить от наших суждений о том, что такое осмысленное. Является ли "stones bet stones stones stones"неграмотным, или только непохожим на имеющую полезный смысл? И, во-вторых, грамматика может определять грамотность на основе более, чем одного компонента,

поэтому факт, состоящий в том, что один компонент ошибается в запрещении некоторой строки, не означает, что грамматика в целом будет ошибаться в этом вопросе. Конечно, если все компоненты грамматики полностью определены, не должно быть проблем с оценкой. Но часто в академической лингвистической работе один или более синтаксических компонентов существует только в виде имени (логическая форма, правила интерпретации, и так далее). Это создает сложность или невозможность оценки эмпирической адекватности правил компонентов, которые полностью определены. В вычислительном контексте избыточное порождение грамматикой не обязательно является проблемой, если цель - распознать или понять правильно сформированный язык. С другой стороны, это могло бы оказаться фатальным, если цель - произвести правильно сформированный язык.

Вопрос того, сопоставляет ли грамматика корректные структуры, также достаточно тонок. Выражения естественного языка не приходят к нам со своей размеченной структурой: мы должны вывести подходящие структуры неявно посредством рассмотрения, например, того, какими другими фразами мы можем заменить данные подфразы без потери грамотности.

Вдобавок к этим эмпирическим критериям стандартные научные суждения, такие как простота и общность, прилагаются к грамматикам во многом тем же образом, как они делают это в любых других теориях, относящихся к природным феноменам. При прочих равных условиях грамматика с семью правилами должна быть предпочтительной по отношению к грамматике с 93 правилами. И, при прочих равных условиях, грамматика для фрагмента английского языка, которая может быть преобразована в грамматику для соответствующего фрагмента французского с некоторыми небольшими изменениями, должна быть предпочтительной по отношению к грамматике для фрагмента английского языка, которая не содержит никакой связи со своим французским двойником.

Также грамматика для шведских просты предложений в составе сложного, которая может быть расширена до того, чтобы управлять классом шведских вопросительных предложений с добавлением нескольких правил и свойств должна быть предпочтительной по отношению к той, которая требует полностью новый набор параллельных правил и свойств, чтобы обрабатывать вопросительные предложения.

Представление простых грамматик на Прологе.

Представить CF-PSG посредством структур данных Пролога, также как представить сети переходов, просто. Один способ состоит в том, чтобы завести предикат "правило который содержит два аргумента: для выражения в левой части и последовательности категорий, появляющихся в правой части (представленных как список). Вот как правила Грамматики 1 будут выглядеть в случае такого представления:

правило(s,[np,vp]).

правило(vp,[v,np]).

правило(vp,[v]).

Для лексических входов, мы можем принять то же соглашение, что и с сетями переходов, вводя сопоставления как:

слово(np,'Dr. Chan').

слово(np,nurses).

слово(np,'Medicenter').

слово(v,died).

...

Грамматика - это формальное множество правил, описывающее, какой должна быть строка в отдельном языке. Пролог, как язык, основанный на логике, в высшей степени подходит для написания формальных описаний и определений различных видов. Поэтому естественно рассматривать выражение грамматики напрямую как логическую спецификацию на Прологе. Фактически чрезвычайно просто перевести маленькую грамматику, которую мы использовали в качестве примера в предложениях Пролога, говоря прямо о грамотных строках слов. Как это бывает, результирующий код на Прологе может быть использован для чтобы точно распознать строки слов, для которых грамматика требует быть грамотными, и для порождения образцов грамотных предложений. Перевод правил грамматики в предложения Пролога требует только осторожного образа мыслей о том, что в действительности говорит каждое правило. Наше правило "предложение"может быть читаемо истолковано как требование того, чтобы строка слов Z, рассматривается как предложении, если есть строка слов X, которая рассматривается как именная фраза, и строка слов Y, которая рассматривается как глагольная фраза, и Z - это только Y, добавленная к X. И это истолкование может быть более сжато выражено в прологовской клаузе Хорна:

```
s(Z) :- np(X), vp(Y),
        append(X,Y,Z).
```

Правило для глагола с объектной глагольной фразой изоморфно:

```
vp(Z) :- v(X), np(Y),
        append(X,Y,Z).
```

А непереходный случай тривиален:

```
vp(Z) :- v(Z).
```

Нам конечно нужен лексикон, и он может быть представлен следующим образом:

```
np(['Dr. Chan']).

np(['MediCenter']).

np([nurses]).

np([patients]).

v([died]).

v([employed]
```

И вот у нас есть распознаватель для крошечного фрагмента английского языка, который может быть вызван так:

```
?- s([patients,died]).

?- s(['MediCenter',employed,nurses]).
```

Благодаря его декларативному характеру мы можем использовать тот же самый код, чтобы осуществлять порождение так же, как распознавание. Попробуйте следующее:

```
?- s(Sentence).
```

Несмотря на элегантность по сравнению с кодом, которым кому-то нужно было бы решить ту же задачу на процедурном языке, эта схема для написания распознавателя не идеальна. Анализ и эффективности и эстетичности приводит к тому, что мы должны исключить явный оператор append, если мы можем это сделать. Технология списка различий, которую мы уже использовали в наших программах прохождения сетей, позволяет нам и исключить раздражающий оператор append и сделать код Пролога более изоморфным по отношению к правилам грамматики, язык который распознан, таким образом одновременно адресуясь и к нашей заинтересованности в эффективность и эстетичности. Мы перекодировали распознаватель следующим образом:

```
s(X,Z) :- np(X,Y), vp(Y,Z).

vp(X,Z) :- v(X,Z).

vp(X,Z) :- v(X,Y), np(Y,Z).
```

```
np(['Dr. Chan'|X],X).

np(['MediCenter'|X],X).

np([nurses|X],X).

np([patients|X],X).

v([died|X],X).

v([employed|X],X).

?- s([patients,died],[]).

?- s(['MediCenter',employed,nurses],[]).
```

Первое правило здесь может быть истолковано как высказывание, что мы можем найти Предложение в начале X (с порцией строки Z, оставшейся после этого), если мы можем найти именную фразу в начале X, оставив после этого Y, и глагольную фразу в начале Y, оставив после этого Z.

Как и раньше, мы можем использовать этот же самый код, чтобы производить порождение так же как распознавание. Попробуйте следующее:

```
?- s(Sentence,[]).
```

Можно напрямую превратить распознаватель в систему грамматического разбора, просто добавив дополнительные аргументы. каждый предикат в программе, который охватывает понятие отдельной грамматической категории, снабжается дополнительным аргументом (появляющимся, скажем, первым), который описывает (в виде списка) структуру фразы.

В логическом переводе каждого правила фразовой структуры дерево, соответствующее левой половине правила категории, будет просто списком, состоящим из имен категорий, следующих в деревьях категорий правой части правила. Например:

```
s([s,NP,VP],X,Z) :- np(NP,X,Y), vp(VP,Y,Z).

np([np,nurses],[nurses|X],X).
```

Будучи переведено на английский язык (если забыть об аргументах, относящихся к части строки), первое предложение говорит, что одна из форм предложения - это ИменнаяФраза, за которой идет ГлагольнаяФраза. Если предложение имеет такую форму, дерево его грамматического разбора будет списком вида "[предложение, ИменнаяФраза,ГлагольнаяФраза] где ИменнаяФраза - это дерево разбора именной фразы, а ГлагольнаяФраза - дерево разбоа для глагольной. Код для нашего распознавателя списка

различий, приведенный выше, имел такую близкую связь с соответсвующей грамматикой, что кто-то мог бы резонно ожидать, что будет возможно перейти от последнего к предыдущему автоматически. Нам бы хотелось получить программу, чтобы превращать предложения, написанные в основном как правила фразовой структуры, в предложения, которые будут разрабатывать распознавание списком различий. Так наши входные данные для программы могли бы быть выражены так:

```
s --> np, vp.

np --> [nurses].
```

(объявляя подходящий оператор как -->), где объявленный выходной список различий, который мы хотели это:

```
s(_1,_2) :- np(_1,_3), vp(_3,_2).

np([nurses|_4],_4).
```

Это становится не таким трудным. Мы определяем предикат "перевести"следующим образом:

```
translate((LHS_in --> RHS_in), (LHS_out :- RHS_out)) :-

        LHS_out =.. [LHS_in,S0,Sn],

        add_vars(RHS_in,S0,Sn,RHS_out).
```

Таким образом устанавливается шаблон для правила входа и предложения выхода, формируется левая часть правила выхода путем создания оболочки для нескольких переменных списка различий, и в заключение передается правая часть правила через "добавить_переменные":

```
add_vars((RHS_in1,RHS_in2),S0,Sn,RHS_out) :- !,

        add_vars(RHS_in1,S0,S1,RHS_out1),

        add_vars(RHS_in2,S1,Sn,RHS_out2),

        combine(RHS_out1,RHS_out2,RHS_out).
```

.Если в правой части входа есть более, чем один элемент, нужно разбить ее на две на первой запятой, и рекурсивно переприменить "добавить_переменные"к двум кускам, а затем скомбинировать результат.

```
add_vars(RHS_in,S0,Sn,true) :-
```

```
        islist(RHS_in), !,

        append(RHS_in,Sn,S0).
```

Если правая часть правила просто состоит из списка, и является таким образом лексическим входом, нужно добавить в нее завершающую переменную списка различий, и затем унифицировать результирующий список начальной переменной списка различий.

```
add_vars(RHS_in,S0,Sn,RHS_out) :-

        atom(RHS_in),

        RHS_out =.. [RHS_in,S0,Sn].
```

Если правая часть правила - это просто одиночный атом, нужно обращаться с ним точно так же, как с символом левой части. Остается только определить "комбинировать":

```
combine(true,RHS_out2,RHS_out2) :- !.

combine(RHS_out1,true,RHS_out1) :- !.

combine(RHS_out1,RHS_out2,(RHS_out1,RHS_out2)).
```

Этим просто сообщается, что результаты правой части, которые унифицируются как "истина могут быть проигнорированы - в противном случае вы получите то, что ожидаете.

Выделение подкатегорий и использование свойств.

Как читатель уже мог заметить, среди того, что Грамматика 1 требует принять как предложение есть неграмотные строки типа "*Dr.Chan died patiets"или "*MedCenter employed"(лингвисты используют "*"чтобы отметить строки, которые интуитивно понимаются как неграмотные в рассматриваемом языке, в этом случае - английском). Проблема здесь касается того, что лингвисты называют "выделением подкатегорий". Хотя и "died"и "employed"глаголы, как того и требует Грамматика 1, они принадлежат к различным подкатегориям класса глаголов. А именно, "employed"переходный и требует следования за ним именной фразы, в то время как "died"непереходный и не может допускать следующей за ним именной фразы. Мы могли бы исправить Грамматику 1, заменив Глагол на две категории - ПереходныйГлагол и НепереходныйГлагол, и изменив правила для глагольной фразы, но такой подход быстро напорождает кучу отдельных частей речи, если будет рассматриваться большой класс предложений.

Вместо того, чтобы следовать решению "с потолка о котором было только что упомянуто, мы вернемся к более принципиальному подходу: тому, который задействует синтаксические свойства. Большинство, если не все, современные теории грамматики задействуют свойства, и пространство и сложность их использования значительно выросли в 80-е годы, хотя их использование в компьютерной лингвистике началось в конце 50-х. В современном синтаксисе теорий со свойствами атомные категории, такие как именная фраза и глагол, заменены множествами спецификаций свойств. Каждая спецификация свойства состоит из свойства, например ПАДЕЖ, и значения этого свойства, например, ВИНИТЕЛЬНЫЙ. Привычные имена для категорий, такие как ИменнаяФраза и Глагол могут в дальнейшем быть переопределены как значения отдельного свойства, которое мы будем называть КАТЕГОРИЯ. Фактически, мы уже сделали это движение в показанных лексических входах.

В свете этого, что мы будем делать теперь, чтобы перестроить Грамматику 1, используя свойства, создав, таким образом Грамматику 2, и решив проблему, обозначенную ранее как выделение подкатегорий. Чтобы сделать Грамматику 2 немного более интересной, чем это могло бы быть в противном случае, мы также введем правило для координации, которое позволит нам проиллюстрировать роль рекурсии в грамматиках.

Сделав это, мы далее расширим Грамматику 2 до Грамматики 3, и делая так, проиллюстрируем описательную мощь технологий теории свойств в диапазоне синтаксических феноменов.

В Грамматике 2 мы привлечем только 2 свойства - именно, КАТЕГОРИЯ и арг1(умент). КАТЕГОРИЯ будет иметь в качестве своих значений метки, которые использовались для самих категорий в Грамматике 1, но с одним добавлением, КООРДИНАЦИЯ, которое мы обсудим позже. arg1, на секунду, имеет только 2 значения: 0 (что значит, ничего) и ИменнаяФраза. Как мы должны интерпретировать эти свойства и их значения? Рассмотрим категорию, которая содержит Глагол в качестве значения свойства КАТЕГОРИЯ, что означает, что это глагол, и ИменнаяФраза в качестве значения свойства арг1. Мы будем интерпретировать последнее как нечто типа глагол, который требует следования за ним именной фразы; это переходный глагол. Глагол с 0 в качестве значения арг1 будет глаголом, который требует, чтобы ничего за ним не следовало; это непереходный глагол.

Теперь у нас есть проблема, как записать правила, исходя из того, что мы заменили не анализируемые (одноместные) категории на набор данных о свойствах. Чтобы было понятнее, мы продолжим записывать, например:

Правило формирование простого предложения

S -> NP VP.

Но теперь это должно быть истолковано как эллипсис для чего-то, что может быть более многословно выражено как следующее:

Правило формирование простого предложения

X0 -> X1 X2:

<X0 cat> = S

<X1 cat> = NP

<X2 cat> = VP.

Таким образом, в этой многословной версии записи нашего правила, мы должны поместить заместительные переменные X0,X1 и X2 в самом правиле, и затем коллекцию равенств, таких как <X0 cat>=Предложение, которые должны быть прочитаны как сообщение "значение свойства КАТЕГОРИЯ X0 равно Предложение". В сокращенной записи правила мы просто заменяем заместительные переменные в правиле на их значения в сопровождающих равенствах. Дав эти ремарки, мы можем теперь представить правила, привлекаемые Грамматикой 2.

Грамматика 2

Правило формирование простого предложения

S -> NP VP.

Правило непереходный глагол

VP -> V:

<V arg1> = 0.

Правило одиночные глаголы с дополнением

VP -> V X:

<V arg1> = <X cat>.

Правило координация идентичных категорий

X0 -> X1 C X2:

<X0 cat> = <X1 cat>

<X0 cat> = <X2 cat>

<X0 arg1> = <X1 arg1>

<X0 arg1> = <X2 arg1>.

Слово died:

<cat> = V

<arg1> = 0.

Слово recovered:

<cat> = V

<arg1> = 0.

Слово slept:

<cat> = V

<arg1> = 0.

Слово employed:

<cat> = V

<arg1> = NP.

Слово paid:

<cat> = V

<arg1> = NP.

Слово nursed:

<cat> = V

<arg1> = NP.

Слово and:

<cat> = C.

Слово or:

<cat> = C.

Первое правило выглядит точно так же как его двойник в Грамматике 1 и выполняет в точности ту же функцию. Второе и третье правило в Грамматике 2 выполняют функции, требуемые от второго и третьего правила в Грамматике 1, но, совместно с показанными лексическими входами, они гарантируют, что после переходного глагола будет идти именная фраза, в то

время как после непереходного глагола ничего идти не будет. Грамматика 2, таким образом, создает корректные требования к грамотности следующих примеров:

Nurses died.

*Nurses died patients.

*MediCenter employed.

MediCenter employed nurses.

Грамматика сопоставит структуры первому и последнему примеру, но для второй и третьей возможных структур нет. Глаголы "die"и "employ"принадлежат к двум разделенным с точки зрения свойств категориям в Грамматике 2: после первого не может идти именная фраза, после второго должна идти именная фраза. Следовательно, с точки зрения Грамматики 2, второй и третий примеры неграмотны.

Четвертое и последнее правило - это схема, которая уводит нас за пределы области, покрываемой Грамматикой 1. Эта схема вводит координирующую конструкцию и говорит, что данная категория может состоять из двух других объектов той же категории, разделяемых элементом категории КООРДИНАЦИЯ, которая будет исключена, чтобы быть реализованной как "and"или "or".

Возвращаясь теперь к лексикону для Грамматики 2, в то время как мы можем просто перенести входы именных фраз из Грамматики 1, мы должны изменить входы глаголов и добавить пару входов КООРДИНАЦИЯ. Обратите внимание, что было добавлено несколько глаголов, чтобы увеличить правдоподобие данных примеров.

Если не считать представления свойств, которые на этой стадии могут принести намного больше проблем, чем стоит, Грамматика 2 обнаруживает очень небольшие различия с Грамматикой 1. Но есть одно фундаментальное различие между ними. Грамматика 1 требует, чтобы в точности 40 строк слов были грамматическими объектами категории Предложение, в то время Грамматика 2 разрешает бесконечно много строк слов в качестве грамматических объектов предложения. Как это может быть? Ответ лежит в координирующей схеме, которую мы ввели в Грамматике 2. Она позволяет разделять любые категории на два объекта той же категории. Эти два объекта могут сами делиться, и так далее. Этот аспект грамматики обеспечивает образец рекурсии в синтаксических правилах (см. пример следующей Грамматики 3 как еще более контрастный пример этого феномена). Грамматика 2 разрешает рекурсию, в то время как Грамматика 1 - нет. Это должно стать яснее из примера на рисунке 4.2, который представляет одно из бесконечного множества деревьев, которые Грамматика 2 позволяет как грамотные. Как может быть видно, у нас есть два координированных предложения, второе предложение само содержит координирующую глагольную фразу. Таким образом, этот пример иллюстрирует, как категория Предложение может быть переопределить

категорию Предложение, и как категория ИменнаяФраза может переопределить категорию ИменнаяФраза. Грамматика не налагает ограничения на то, сколько раз категории могут быть переопределены координирующим правилом, и поэтому нет ограничения на число выражений, которое допускает Грамматика 2. Следовательно, Грамматика 2 допускает следующие примеры:

Nurses died.

Nurses died and patients recovered.

Nurses died and patients recovered and Dr. Chan slept.

Nurses died and patients recovered and Dr. Chan slept

and MediCenter employed nurses.

Nurses died and patients recovered and Dr. Chan slept

and MediCenter employed nurses and ... .

Хотя Грамматика 2 допускает как грамотные бесконечное множество предложений, на бы быстро надоело перечислять их. Частично это потому, что лексикон для Грамматики 2 очень мал и растянутые таким образом примеры будут заставлять нас повторять лексические элементы. Давайте поэтому разовьем Грамматику 2 в Грамматику 3 и, делая это, увеличим наш лексикон посредством более полной эксплуатации технологии использования свойства (arg1), чтобы закодировать категорию дополнений, которые мы привлекали для выделения подкатегорий в Грамматике 2. Правила для Грамматики 3 те же, что задействованы в Грамматике 2 с одним дополнением, правилом для развития предложных фраз:

Правило предложные фразы

PP -> P X:

<P arg1> = X.

Только когда мы перейдем к лексикону Грамматики 3, эти различия станут действительно явными. Грамматика 3 позволяет нам привлекать гораздо более широкий диапазон отдельных подтипов лексических элементов, включая 6 различных видов глаголов, посредством расширения лексикона Грамматики 2.

Лексикон 3

Слово approved:

<cat> = V

<arg1> = PP.

Слово disapproved:

<cat> = V

<arg1> = PP.

Слово appeared:

<cat> = V

<arg1> = AP.

Слово seemed:

<cat> = V

<arg1> = AP.

Слово had:

<cat> = V

<arg1> = VP.

Слово believed:

<cat> = V

<arg1> = S.

Слово thought:

<cat> = V

<arg1> = S.

Слово of:

<cat> = P

<arg1> = NP.

Слово fit:

<cat> = AP.

Слово competent:

<cat> = AP.

Слово well-qualified:

<cat> = AP.

Грамматика 3 обеспечивает нас структурами для всех следующих примеров, так же как и бесчисленных других:

Nurses thought Dr. Chan seemed competent.

Dr. Chan appeared well-qualified and disapproved of MediCenter.

Patients had believed nurses thought Dr. Chan had slept.

Этот последний пример иллюстрирует рекурсию в предложении (простом предложении) или глагольной фразе, дополняющей глагол, - мы видим трактовку рекурсивной сети переходов этого феномена в главе 3. Это достаточно общая форма рекурсии, находящаяся в естественных языках.

Грамматика нашего последнего примера содержит полную структуру, задействованную в Грамматике 2 и Грамматике 3. Поскольку Лексикон 4 будет идентичным во всех аспектах Лексикону 3, мы не будем повторять его здесь. Фактически, единственное действительное изменении, которое мы сделаем в Грамматике 4 - это добавление нового свойство, которое мы будем называть СЛЭШ благодаря соглашению о записи, посредством которого мы будем использовать X/Y чтобы представить категорию X0, чья КАТЕГОРИЯ это X, и чей СЛЭШ - это Y; то есть это <X0 КАТЕГОРИЯ>=X и <X0 СЛЭШ>=Y. Добавление этого свойства в систему требует определенных последующих добавлений в правила.

Выражения типа Предложение/ИменнаяФраза и ГлагольнаяФраза/ПредложнаяФраза становятся отдельным видом категорий, но каким именно? Каково интуитивное содержимое этой записи? Ответ достаточно прост: выражение категории X/Y - это выражение категории X, из которого исключено выражение категории Y. Таким образом, Предложение/ИменнаяФраза - это предложение (или простое предложение), в котором пропущена именная фраза, в то время как ГлагольнаяФраза/ПредложнаяФраза - это глагольная фраза, в которой пропущена предложная фраза. Чтобы сделать полезными, или даже чтобы сделать имеющими смысл, эти СЛЭШ-категории, нам нужно будет сделать несколько дополнений к правилам, которыми снабжена Грамматика 3.

Правило формирование простого предложения

S -> NP VP:

<S slash> = <VP slash>

<NP slash> = 0.

Правило непереходный глагол

VP -> V:

<V arg1> = 0

<V slash> = 0

<VP slash> = 0.

Правило одиночные глаголы с дополнением

VP -> V X:

<V arg1> = <X cat>

<V slash> = 0

<VP slash> = <X slash>.

Правило предложные фразы

PP -> P X:

<P arg1> = <X cat>

<P slash> = 0

<PP slash> = <X slash>.

Правило координация

X0 -> X1 C X2:

<X0 cat> = <X1 cat>

<X0 cat> = <X2 cat>

<C slash> = 0

<X0 slash> = <X1 slash>

<X0 slash> = <X2 slash>

<X0 arg1> = <X1 arg1>

<X0 arg1> = <X2 arg1>.

Правило актуализация

X0 -> X1 X2:

<X0 cat> = S

<X1 empty> = no

<X2 cat> = S

<X2 slash> = <X1 cat>

<X2 empty> = no

<X0 slash> = <X1 slash>.

Правило исключение слэша

X0 -> :

<X0 cat> = <X0 slash>

<X0 empty> = yes.

Первые пять правил грамматики 4 просто являются пересмотром тех же правил грамматики 3 для разрешения нам нового свойства: в любом случае, значение СЛЭШ (любое) на материнском уровне уравнено со значением СЛЭШ на дополнительном дочернем, или всеми дочерними, в случае координационного правила. Шестое и седьмое правила полностью новые. Сущность правила актуализации может быть более понятно выражено , используя нашу СЛЭШ-запись:

S -> X S/X.

Это правило говорит, что предложение может состоять из некоторой категории, за которыми следует предложение, в котором пропущена эта категория. И, такое чередование, взятое вместе с остатком Грамматики 4, позволит такие предложения как следующие (без запятых):

MediCenter, nurses disapproved of _.

Of MediCenter, nurses disapproved _.

Well-qualified, Dr. Chan had seemed _.

Как легко может быть видно, во всех предложения, за которыми следует запятая в этих примерах, имеется некий пропуск. За этот пропуск элементов в Грамматике 4 отвечает шестое правило, которое может быть так же записано следующим образом:

X/X -> .

Это просто говорит, что X, в котором пропущено X, можно понимать как ничто.

Таким образом, ИменнаяФраза/ИменнаяФраза, МестоименнаяФраза/МестоименнаяФраза и ФразаПрилагательного/ФразаПрилагательного все допустимы как ничто в структуре, определенной Грамматикой 4.

Заметьте, что наши модификации в правилах Грамматики 3, означают, например, что предложение, в котором пропущено Y, может состоять из именной фразы, за которой идет глагольная фраза, в которой пропущено Y (где Y может быть, скажем, именной фразой). Таким образом, то, что позволяют модифицированные правила - это передача информации о пропущенных категориях из материнского уровня на дочерний. Способ, каким это все работает, мог быть стать понятнее из рассмотрения подходящего дерева, представленного на рисунке 4.3.

В принципе, нет ограничений на количество внедренного материала, который может появиться между перемещенным образованием в начале предложения и пустым образованием, которое ему соответствует и которое появляется внутри предложения. Такие конструкции показывают то, что известно как несвязанные зависимости и поразительно общераспространенны в мировых языках. Английский, например, широко использует этот вид конструкций, наиболее значительно - в вопросительных предложениях и простых предложениях в составе сложного. В главе 3 мы указали, как трактовка расширенной сети переходов может закреплять такие конструкции, используя глобальный регистр HOLD, чтобы запоминать перемещенный материал.

Такой процедурный подход, который специализирован на разборе, отличается в основном от декларативного описания слэш-свойства, предложенного в Грамматике 4. Хотя анализ актуализации, данный здесь, "недоразвит он служит для того, чтобы иллюстрировать вид технологии передачи свойств, который очень распространен в современных работах компьютерной лингвистики по синтаксису.

Лексикон в Грамматике 3 почти адекватен тому, который содержится в Грамматике 4, исключая одну проблему. Правила грамматики явно требуют, чтобы слова типа глаголов и предлогов содержали 0 в качестве значения их слэш-свойства. Это предназначено для того, чтобы предотвратить перемещение таких слов с их нормальных позиций и деактуализации конструкций, содержащих

пропуски. Как результат, лексические входы нуждаются в явном установлении их слэш-свойства, как здесь:

Слово died:

<cat> = V

<slash> = 0

<arg1> = 0.

Просто, но трудоемко вносить необходимые изменения во все лексические входы Грамматики 3. Использование макросов в лексических входах, идея которая должна быть представлена в Главе 7, один из возможных путей осуществления таких глобальных присвоений свойств лексическим единицам.

Заметим, что наше использование свойств в Грамматике 2, Грамматике 3 и Грамматике 4 было очень ограниченным. Мы должны были просто заменить одноместные имена категорий типа Предложение и ИменнаяФраза на маленькое конечное множество пар значений атрибутов, где и атрибут и значение берутся из маленького множества крошечных элементов. Следовательно, все эти грамматики могут быть переведены совершенно без проблем в эквивалентные грамматики с одноместными категориями, хотя нет очевидных вычислительных причин так делать. Например правило: Правило формирование простого предложения

S -> NP VP:

<S slash> = <VP slash>.

Может быть заменено конечным числом правил вида:

Правило

S/S -> NP VP/S.

Правило

S/NP -> NP VP/NP.

Правило

S/VP -> NP VP/VP.

Правило

S/AP -> NP VP/AP.

Где S/S, VP/S и так далее должны читаться как одиночные символы, так же как NP и S. Поскольку число категорий в таких грамматиках конечно, а правила контекстно-свободны и конечны числом, мы не должны покидать область CF-PSGЮ даже если описательный аппарат намного более сложен, чем применявшийся в Грамматике 1. Мы рассмотрим математически более далеко идущие последствия использования систем свойств, представленных формализмом PATR, в конце этой главы.

Грамматики, которые мы разрабатывали, лингвисты назвали бы "игрушечными". Они служат для иллюстрации современных работ в области теоретических грамматик свойств, но не должны восприниматься слишком серьезно применительно к анализу английского языка. Факт, что Грамматика 3 и Грамматика 4 работают настолько хорошо, насколько они это делают во многом благодаря отдельному лексикону, который они используют. Читатель, который справился как минимум с механизмом Грамматики 3, может с пользой рассмотреть вопросы, которые следуют далее.

Грамматика определенных простых предложений.

Хотя перевод CF-PSG прямо в предложения Пролога прост, существует новая важная проблема, которую нужно рассмотреть при переводе грамматик, основанных на общих свойствах. В таких грамматиках категория - это связка свойств и поэтому не может быть представлена просто посредством маленького предиката, касающегося строк слов. Все что мы должны сделать - это перевести каждую категорию в предикат вместе с последовательностью аргументов, кодирующих свойства в связке так, как сообщающие правильную информацию о порциях строки. Как и раньше, было бы полезно иметь более сокращенную запись, что позволило бы записать такие грамматики как программы на Прологе, но без необходимости беспокоиться о списках различий. К счастью, многие Пролог-системы заканчивают с встраиванием в интерпретатор такого рода записи, записи Грамматики определенный простых предложений (Definite Clause Grammar - DCG).

Фактически, мы почти увидели DCG в предыдущем параграфе - следующие выражения - это правила DCG:

```
s --> np, vp

np --> [kim].
```

Интерпретатор Пролога, снабженный транслятором DCG, автоматически переведет эти выражения в формат распознавания списка различий, который мы уже обсудили. С точки зрения этого примера DCG - просто фантастическое имя для незначительной вариации стандартного формализма PSG. Но этот пример вводит в заблуждение, что формализм DCG позволяет кому-то делать что-то, не разрешенное в PSG, а именно - ассоциировать как аргументы, термы и/или логические переменные, так и многое другое, что Вам нужно, с

категориями. Следовательно следующее правило также допустимое выражение в формализме DCG:

s –> np(Лицо,Число,именительный_падеж),

vp(Лицо,Число,не_неопределенная_форма).

который снабженная DCG Пролог-система переведет в нечто, похожее на:

```
s(_12,_13) :- np(_14,_15,именительный_падеж,_12,_16),
      vp(_14,_15,не_неопределенная_форма,_16,_13).
```

Принимая, что переменные и константы в оригинальном правиле DCG именуются не так извращенно, правило требует, чтобы предложение состояло из именной фразы в именительном падеже, за которой следует глагольная фраза не в неопределенной форме, где именная и глагольная фразы имеют одни те же значения лица и числа. В PATR это могло бы быть выражено чем-то вроде:

Правило S -> NP VP:

<NP person> = <VP person>

<NP number> = <VP number>

<NP case> = именительный_падеж

<VP vform> = не_неопределенная_форма.

Заметим, что в версии DCG число и порядок аргументов фиксирован и является ключевым. Так в то время как 'np(третье_лицо, множественное_число, Падеж)' вероятно и осмысленно и полезно в контексте данного правила, внешне сходные 'np(третье_лицо, множественное_число)' 'np(множественное_число, Падеж, третье_лицо)' более вероятно и бесполезны и вводят в заблуждение. (Мы ограничиваем здесь наши требования, поскольку истина зависит от того, как выглядит остаток грамматики. Кто-то может использовать "Падеж"как переменную, перебирающую некоторое число значений, или использовать "не_неопределенная_форма"как имя падежа, но делать так было бы извращением.

Нотация PATR не страдает от этих фиксированных арностей и фиксированного порядка требований в уравнениях, комментирующих эти правила. Заметим также, что в то время как мы выбираем мнемонические имена переменным в правиле нашего примера, эти имена обычно не были помещены внутрь Пролог-системы, и таким образом эффект от производства разбора может быть представлен выражениями типа

'np(_43,_76, винительный_падеж,_15,_16)','vp(_43,_56,_97,_16,_19)',

таким образом требующие кого-то, чтобы запоминать, что третий аргумент в vp имеет дело со временем, второй аргумент в np имеет дело с числом, и т.д. Последняя проблема может быть преодолена, хотя и ценой небольшого занудства - посредством использования термов, разделенных двоеточиями (или -"или любыми другими символами, определенными как инфиксный оператор) вместо голых переменных. Первый компонент этих термов может быть константой, обеспечивающей мнемоническое имя для функции слота, в которой она появляется, так:

```
s –> np(лицо:Л,число:Ч,род:Р,падеж:именительный),
        vp(лицо:Л,число:Ч,род:Р,глагольная_форма:не_неопределенная).
        vp(лицо:Л1,число:Ч1,род:Р1,глагольная_форма:Г1) –>
        xv(лицо:Л1,число:Ч1,глагольная_форма:Г1),
        np(лицо:Л2,число:Ч2,род:Р2,падеж:винительный),
        vp(лицо:Л2,число:Ч2,род:Р2,глагольная_форма:неопределенная).
```

И в грамматике после этих строк лексические входы должны выглядеть так:

```
np(лицо:3,число:единственное,род:_,падеж:_) –> [kim].
np(лицо:3,число:единственное,род:женский,падеж:именительный)
        –> [she].
det(число:единственное) –> [a].
det(число:_) –> [the].
nn(число:множественное,род:нет) –> [scissors].
nn(число:_,род:_) –> [sheep].
nn(число:множественное,род:нет) –> [scissors].
bv(лицо:_,число:множественное,глагольная_форма:не_неопределенная)
–> [are].
tv(лицо:_,число:_,глагольная_форма:не_неопределенная) –> [ate].
xv(лицо:3,число:единственное,глагольная_форма:не_неопределенная)
–> [sees].
```

Поскольку DCG позволяет помещать термы и переменные на произвольную глубину, правила, подобные следующим, вполне допустимы:

s –> np(лицо:P,число:N,род:S,падеж:C),

s(slash:np(лицо:P,число:N,род:S,падеж:C)).

vp(лицо:P,число:N,род:_,глагольная_форма:V,slash:np(лицо:_,

число:_,род:_,падеж:винительный)) –>

tv(лицо:P,число:N,глагольная_форма:V).

Заметим, что запись ": хотя это ведет к более ясной грамматикам и разборам, не освобождает пишущего грамматику от заботы о постоянном использовании тех же самых позиций аргументов предикатов для тех же свойств (нечто, о чем пользователь PATR не должен беспокоиться).

Наши примеры таким образом сконцентрированы на использовании DCG в качестве распознавателей, основанных на свойствах, но не формализме заставляющем их распознавать. Поскольку аргументы для категорий могут содержать произвольные термы, мы можем использовать их для разбора, или, в более общем случае, синтаксической трансляции. Так для разбора мы могли бы написать правила типа:

s([s,NP,VP]) –> np(NP),vp(VP).

vp([vp,XV,NP,VP]) –> xv(XV),np(NP),vp(VP).

или типа:

s(s(NP,VP)) –> np(NP),vp(VP).

vp(vp(XV,NP,VP)) –> xv(XV),np(NP),vp(VP).

в зависимости от того, хотим мы, чтобы разбор возвращался в форме списка или терма. И, поскольку нет ограничения на число разрешенных аргументов, мы можем, если захотим, просто добавить эти конструкции разбора к уже проиллюстрированным аргументам свойств.

Формулирование грамматики, основанной на свойствах, на DCG, как было проиллюстрировано, основано на фундаментальном свойстве, относящимся к грамматикам, а именно на том, что есть признак (мы назвали его "категория"), чье значение элементарно и всегда получаемо в любых описаниях категорий, данных в грамматике. Это свойство необходимо, так как нам необходимо использовать свойство cat как предикат Пролога в формулировках, а в Прологе невозможно иметь цель, предикат которой неизвестен (является переменной). Конечно в правиле вроде:

Правило координация

X0 -> X1 C X2:

...

Конечная цель не в том, чтобы определить свойство категория для X0, X1 и X2. С грамматиками, содержащими правила этого рода, у нас не остается большого выбора, кроме как использовать категория как любое другое свойство и обеспечить подставной предикат, скажем "p используемый со всеми категориями. Этот предикат должен всегда иметь то же число аргументов (по аргументу для каждого возможного свойства) по порядку, так как могут быть описания категорий, аналогичные X0 выше, чтобы было соответствие с каждой возможной категорией. Результат трансляции

грамматики PATR этого рода в DCG будет множеством правил, подобных следующим:

p(s,Лицо1,Число1,Падеж1,ГлагФорма1) –>

p(np,Лицо,Число,именительный_падеж,ГлагФорма2),

p(vp,Лицо,Число,Падеж2,не_неопределенная_форма).

(где аргументы p представляют категорию, лицо, число, падеж и глагольную форму по порядку).

Суммируя сказанное, формализм DCG представляет довольно естественное расширение CF-PSG в контексте Пролога. Он удобен, будучи стандартно обеспечиваемым Пролог-системами, и понятен, его семантика такая же декларативная, как у Пролога, и, конечно, он хорошо интегрируется с Прологом. Аргументы, связанные с категориями, позволяют осуществлять и разбор и трансляцию более общим образом (например, перевод с английского на язык логики, как мы увидим в главе 8). Как мы увидим в следующей главе, система разбора, являющаяся стандартной интерпретацией, обеспечиваемой формализма, это система разбора слева направо с возвратом (бэктрэкингом), фактически система, имеющая стратегию разбора в точности ту же, что и Пролог сам по себе. По этой причине, система разбора DCG быстра (поскольку задействует Пролог напрямую), но не эффективна (поскольку она не может хранить промежуточные результаты) хотя и не хуже с этой точке зрения, чем типичные ATN сравнимой области применения. С точки зрения пишущего грамматику, главный недостаток формализма DCG - это принятие "унификации термов"в противоположность "унификации графов"в PATR. Это означает, что все слоты для информации о свойствах должны быть явно обеспечены в фиксированном порядке на любой категории, к которой они применяются, даже когда нет доступной информации и слот просто содержит переменную. Это ведет к грамматическому и лексическому многословию, в то время как формализмы, основанные на унификации графов позволяют пресекать неизвестную и неподходящую информацию в свойствах.

Это тема, к которой мы вернемся в главе 7.

Классы грамматик и языков.

В этой секции мы коснемся не формализмов, которые были задействованы в работе с грамматикой, а скорее контуров формальных грамматик, которые были привлечены, и выводимых из них классов языков. Иерархия языков Хомского, представленная на рисунке 4.4, дает примерный размер грамматической мощности, здесь нижний уровень соответствует классу 3 языков, также известных как регулярные языки или языки конечных состояний, наивысший уровень - класс языков 0, также известных как рекурсивно-перечислимые множества. Между ними идет несколько других классов языков, включая индексированные языки и тип 2 или контекстно-свободные языки.

Контекстно-свободные грамматики и языки.

Языки, которые могут быть описаны посредством CF-PSG, которые в то же время и языки описываемые посредством RTN, известны как контекстно-свободные языки (CFL). CF-PSG - относительно просто записать и модифицировать.

Грамматики CF-PSG связаны с успешной традицией реализации на компьютере, что обеспечивает нас полным пониманием того, как разбирать их, транслировать и компилировать. Одним из мотивов интереса к CF-PSG является существование относительно высокоэффективных алгоритмов распознавания и разбора контекстно-свободных языков. Такой разбор - базовое средство для компьютерной науки, включая обработку естественных языков. Были даже предложения включить реализацию системы грамматического разбора CF-PSG в состав специальных аппаратных средств. Хотя CF-PSG и RTN эквивалентны по мощности, в общем случае первая грамматика предпочтительнее для описания естественных и формальных языков. RTN включает онтологию состояний вместе с их свойствами(начальное, конечное)- ничего подобного не требуется в CF-PSG. Обычный взгляд на RTN состоит в том, чтобы использовать жестко определенный вид обработки (сверху вниз, слева направо, с поиском в глубину), в то время как CF-PSG оставляет все вопросы обработки полностью открытыми. RTN имеет одно небольшое преимущество перед CF-PSG: они могут напрямую осуществлять итерацию (посредством дуг, которые возвращаются назад в состояние, из которого они вышли), в то время как CF-PSG вынужден использовать рекурсию чтобы получить итерацию. Вводные учебники по лингвистике и даже по искусственному интеллекту и другие, ориентированные на обучение, работы часто утверждают, что такие феномены как зависимости между несмежными словами при согласовании субъекта с глаголом характеризуют английский как не контекстно-свободный язык. Это не так. Даже языки с конечными состояниями могут демонстрировать зависимости между символами, отстоящими произвольно далеко друг от друга. Чтобы привести искусственный пример, предположим, что последнее слово в каждом предложении должно нести специальный маркер, определяемый первой морфемой в предложении. Конечный автомат для данного языка мог бы просто закодировать в своих состояниях информацию о том, какая морфема стоит в начале, и отмечать последнее слово в зависимости от состояния непосредственно перед его обработкой. Однако, в последнее время был найден по крайней мере один очевидно корректный пример языка, не являющегося контекстно-свободным - швейцарский немецкий. Какое отношение должна принять исследовательская работа и разработки в области обработки естественных языков в связи с появлением свидетельств того, что эти языки не все относятся к контекстно-свободным? Фундаментальная вещь, которую следует запомнить состоит в том, что ошеломляющее большинство структур любых естественных языков может быть элегантно и эффективно разобрано при помощи технологий контекстно-свободного разбора. Почти все конструкции почти во всех языках могут быть разобраны с использованием технологий, которые ограничивают системы анализом контекстно-свободных языков.

СНОСКА:

Диалекты разговорного немецкого в окрестностях Цюриха в Швейцарии демонстрируют пример подходящих образцов порядка слов в определенных подчиненных неопределенных простых предложений: за произвольным числом именных фраз может идти глагол не в неопределенной форме и определенное

число глаголов в неопределенной форме, несколько именных фраз, являющихся функциями лексических свойств глаголов. В добавок семантические отношения между глаголами и именными фразами демонстрируют пересекающиеся серийные образцы: глаголы, находящиеся далее вправо в строке глаголов, воспринимают в качестве своих объектов именные фразы, находящиеся далее вправо в строке именных фраз. Решающие подстроки имеют форму NP\ v'-.4m'm\ v'.4m'V[n]. В простом случае, когда m=n=5 такая подстрока могла бы иметь значение типа:

Claudia watched Helmut let Eva help Hans make Ulrike work.

но в соответствующем порядке слов:

Claudia Helmut Eva Hans Ulrike watched let help make work NP\ d1\ u NP\ d2\ u NP \ d3\ u NP\ d4\ u NP \ d5\ u V\ d1\ u V \ d2\ u V\ d3\ u V \ d4\ u V\ d5\ В швейцарском немецком есть специфическое свойство, которое делает этот феномен подходящим для решения проблемы грамматической мощности: определенные глаголы требуют скорее дательный, чем винительный, падеж, указывающий на их объект. Этот образец, в общем, не будет одним из тех, которые может описать CF-PSG. Например, если мы ограничим ситуацию (технически, пересечением подходящим регулярным языком) простыми предложениями, в которых перед всеми именными фразами в винительном падеже будут идти все именные фразы в дательном падеже (NP\ dd\ u), то грамотными простыми предложениями будут в точности те, в которых перед глаголами, требующими винительный падеж (V\ da\ u), будут идти глаголы, требующие дательный падеж (V\ dd\ u), и количества должны соответствовать, схематично: NP\ da\ u\ v'-.4m'm\ v'.4m' NP\ dd\ u[n] V\ da\ u[m] V\ dd\ u[n]

Но эта схема имеет форму языка строк типа a\ v'-.4m'm\ v'.4m'b[n]c[m]d[n], где n больше 0, для которого можно показать, что он не является контекстно-свободным.

Грамматики конечных состояний и регулярные языки.

Какой наименьший известный класс формальных языков разумно брать для обработки натуральных языков? Самый ограниченный кандидат, который когда-либо рассматривался в этой роли - это класс конечных состояний, регулярный или 3-го типа языка. Это в точности те языки, которые могут быть распознаны сетью переходов конечных состояний, представленной в главе 2. Они могут также быть охарактеризованы грамматиками типа Грамматики 1, но при условии, что каждое правило представляет не более одной категории (не терминальной) в конечной позиции или все правила, представляющие категории, должны помещать их в начальную позицию. Вот грамматика конечных состояний для языка, порождаемого Грамматикой 1, которая относится к регулярным языкам:

Rule NP -> Dr. Chan. Rule S -> Dr. Chan VP. Rule NP -> nurses. Rule S -> nurses VP. Rule NP -> MediCenter. Rule S -> MediCenter VP. Rule NP -> patients. Rule S -> patients VP. Rule VP -> died. Rule VP -> died NP. Rule VP -> employed. Rule VP -> employed NP.

Однако, есть справедливый аргумент в пользу того, что не все естественные языки регулярны. Он основан на том факте, что строка следующего вида: A

doctor (whom a doctor)m (hired)n hired another nurse. составляет допустимое предложение английского языка только если m и n равны. В обычных грамматических терминах это потому, что каждое появление фразы 'a doctor' - это именная фраза, которая нуждается в таком глаголе как 'hired', чтобы закончить простое предложение, в котором она служит субъектом. Тот факт, что естественный язык не является регулярным, не обязательно означает, что технологии разбора регулярных языков не подходят в обработке естественных языков. Некоторые преподаватели многие годы предлагали слушателям скорее обрабатывать предложения так, как если бы они были конечными автоматами (или как если бы они были автоматами со стековой памятью с ограниченной глубиной стека), чем акцентировать внимание на поведении, несущем характеристики более мощного устройства. Несмотря на развитие разбора естественных языков, теория регулярных языков и конечных автоматов в изучении естественных языков будет по-прежнему важной, даже если они и не регулярны. В самом деле, мы уже видели пользу преобразователей на конечных состояниях в области морфологии в главе 2.

Индексированные грамматики и языки.

Если, в добавок к конечному множеству пар значений атрибутов, категории могут также привлекать одиночное рекурсивное свойство, называемое, скажем, STACK, и если правила можно назначать, проверять и расширять этим свойством, результирующая выразительная мощность относится к классу грамматик, известных как индексированные грамматики. Например, предположим мы разрешаем категории X, которые могли быть описаны следующим способом:

<X cat> = S, <X stack top> = a, <X stack stack top> = a,

<X stack stack stack top> = z. И имели грамматику справилами: Rule поместить маркер конца в стек S -> a A: <A stack top> = z <A stack stack> = <S stack>. Rule поместить 'a' в стек A0 -> a A1: <A1 cat> = A <A2 cat> = A <A1 stack top> = a <A1 stack stack> = <A0 stack>. Rule копировать стек от A до B A -> B: <A stack> = <B stack>. Rule вытолкнуть 'a' из стека B0 -> b B1 c: <B0 cat> = B <B1 cat> = B <B0 stack top> = a <B0 stack stack> = <B1 stack>. Rule вытолкнуть маркер конца из стека B -> b c: <B stack top> = z.