Заметки об NLP (часть 1)

В преддверии Нового года решил начать небольшой цикл статей, посвящённых наиболее интересующему меня лично направлению в обработке текстов на естественном языке. (То есть NLP в заголовке означает natural language processing — ваш К.О.) Синтаксический анализ, семантика, машинный перевод, поиск смысла слова в контексте — в общем, вся радость компьютерного лингвиста :)

Наверно, сразу имеет смысл определиться с уровнем изложения. Я сам пытаюсь заниматься компьютерной лингвистикой (с переменными успехами). Постараюсь рассказать о том, что конкретно волнует, чего уже можно, чего пока нельзя, и над чем как раз сейчас надо работать. Быть может, эти статьи помогут мне самому отструктурировать информацию в голове и опираться на уже готовую структуру в новом году. А если у читателей появятся свои идеи или мысли о сотрудничестве — ещё лучше.

Сначала — немного об общей структуре нашего разговора. К области NLP можно отнести кучу самых разных вещей — от морфологического анализа слов до поиска адресов в текстовом файле. Я буду говорить лишь о том, чем занимаюсь. Подход — скорее научный, чем практический. Это не значит, что меня не волнуют применения, вовсе нет! Просто я не буду заниматься потенциально тупиковой ветвью, даже если она даёт некий быстрый (продавабельный) сиюминутный результат. Также меня не волнует, на каком языке что написано, будет это в интернете или на винчестере, виндоус или линукс и тому подобные вещи. Главное — качество идей, их потенциальная развиваемость, а также применимость в разных задачах и к разным языкам.

Начнём с небольшой экскурсии «по уровням» NLP, чтобы было понятно, на каком из них будет вестись основной разговор.

Извлечение фактов и разбивка текста

Есть довольно обширный класс задач, которые удаётся решать без особенного привлечения средств NLP. Допустим, требуется узнать, что в интернете пишут о Президенте России (не сомневайтесь, соответствующие службы этим занимаются :)). Соответственно, надо искать сочетания «Дмитрий Медведев», «Президент России», «Д. Медведев» и прочее подобное. Конечно, здесь тоже не всё так просто, в тексте могут встретиться и фразы вроде «бывший Президент России», и просто «глава государства» (надо понять, какого) — вот эти штуки и заставляют изобретать алгоритмистов различные эвристики. Того же уровня задача — поиск почтовых адресов. В принципе, есть более-менее понятная структура: индекс, край, город, улица, дом, квартира. Но разные люди по-разному могут написать, и в целом задача эта нетривиальная.

Примерно в ту же «корзину» я бы определил задачу нахождения концов предложений. Вроде бы NLP, но не совсем. В простейшем случае ищем

точку, а за ней заглавную букву. Вот тебе и конец, вот и начало. Но на самом деле, конечно, находится миллион исключений: «На **ул. Иванова** находится наш офис», «At 10 **p.m. London** time, the Reuters reported...»

В общем, вроде бы и NLP, но всё-таки назвать это полноценной «компьютерной лингвистикой» как-то не получается.

Обработка на уровне слов

Здесь в первую очередь приходит в голову автоматический морфологический анализ: требуется по слову определить часть речи и «атрибуты», определяющие словоформу. Например, «красная» — это прилагательное женского рода в именительном падеже, единственном числе, с начальной формой «красный». А «стекла» — либо существительное «стекло» в родительном падеже, единственном числе, либо глагол «стечь» в третьем лице, женском роде, прошедшего времени.

Здесь же стоит и обратная задача синтеза: по данным атрибутам и начальной форме сгенерировать требуемую словоформу.

Морфология — штука, прямо скажем, интересная, и для разных языков имеет совершенно различную сложность. Для английского, к примеру, это просто. Для русского — достаточно сложно. Для финского — совсем сложно, там очень развитая морфология. Для японского — вроде бы не очень сложно, но там есть другая проблема: надо сначала понять, где одно слово кончается, а другое начинается!

Самое хорошее в этой задаче то, что её можно считать решённой. Сейчас есть очень неплохие системы (различной степени неплохости для разных языков). Со своими плюсами и минусами, не без этого. Но если задаться целью, скачать / купить приличный анализатор (и даже синтезатор) можно.

Сам по себе, однако, морфологический анализ и синтез трудно назвать очень полезным. Понятно, что такой анализатор или синтезатор — всего лишь модуль в более крупном проекте. Впрочем, я думаю, что для человека, изучающего иностранный язык с развитой морфологией, такой софт был бы хорош сам по себе. Как дойдут руки, выложу ссылку на свой прототип аналазитора/синтезатора русской морфологии для учащихся.

Обработка на уровне предложений

Сразу скажу: именно здесь мы в основном работать и будем. Поэтому я пока лишь вкратце поясню, о чём идёт речь, а подробнее поговорим в следующей статье.

Основная цель анализа предложения — построить дерево зависимостей или дерево грамматического разбора (parse tree). Оно показывает структуру предложения, в частности, какие слова от каких зависят. Зачем это нужно? Например, в системе машинного перевода. Допустим, есть английское предложение «I have a red ball and a blue shovel». При переводе

на русский компьютер должен понять, что red относится к ball, и переводить это слово надо в мужском роде («красный мяч»). А вот лопата женского рода, стало быть, она «синяя».

Надо сказать, гугл-переводчик в разборе совсем плох, и фразу «I have a blue shovel» переводит как «У меня есть синий лопатой». Это, конечно, никуда не годится.

Наверно, здесь же можно упомянуть об уточнении значений слов с помощью локального контекста. Например, слово «разбить» переводится на английский язык, по крайней мере, десятью различными способами: to break (разбить чашку), to defeat (разбить врага), to lay out (разбить парк) и т.п. Есть и явно экзотические варианты по типу «его разбил паралич»; понятно, что по-английски там никакого «разбивания» нет (he was paralysed). В сложных случаях нужно анализировать широкий контекст фразы, но на практике часто достаточно и локального контекста. Если разбивают чашку, значит to break, если сад / парк — to lay out и так далее.

Я хочу остановиться на этом уровне не только потому, что он лично мне интересен, но и потому, что, как мне кажется, именно здесь продолжается основная работа. Всё-таки разбор фраз пока ещё не столь хорош, как нам того хотелось бы.

Обработка на уровне текста

Конечно, уровнем предложений наши проблемы не ограничиваются (однако заметим, что без хорошего разбора предложения дальше ехать бесполезно). Очень часто требуется более глобальный контекст. Пример: «I have a sibling. She is beautiful». По-английски «sibling» — это «брат или сестра». Так как дальше речь идёт явно о женщине («she»), переводить «sibling» надо как «сестра». Это называется отслеживанием ссылок. Другие примеры из той же оперы: «Minulla on veli. Hänellä on auto» — «У меня есть брат. У него есть машина» (финский). Тут штука в том, что «Hänellä» — это и «у него» и «у неё» (в финском нет различия между «он» и «она»), и надо правильно выбрать вариант. Бывают штуки и похуже: как перевести на финский «он и она — одна сатана»? :) Иногда приходится собирать целый арсенал сведений об интересующем объекте. Например, на японский невозможно перевести фразу «мой брат — студент», так как в японском нет слова «брат» — есть только «старший брат» и «младший брат». Соответственно, будет «watashi no ani wa gakusei desu» или «watashi no ototo wa gakusei desu» (и это мы не упоминаем о степенях вежливости речи в японском, что тоже проблема).

Но тут, как видите, получается крутой коктейль из data mining, natural language processing, всякой эвристики и даже трудно предположить чего ещё. Задачи интересные, но я пока в них соваться не готов :)

Наверно, на этом статью пора заканчивать (ибо объём). Виноват, если получилось пока что не слишком информативно — дальше будет интереснее.

Заметки об NLP (часть 2)

Хотя в первой части я и говорил, что не собираюсь останавливаться на морфологии, видимо, совсем без неё не получится. Всё-таки обработка предложений сильно завязана на предшествующий морфологический анализ.

Лирическое отступление. Наш с вами родной русский язык очень хорош (для нас) и труден (для иностранцев) богатой фонетикой и разнообразием грамматических средств. Поэтому нам, в принципе, должно быть не очень сложно изучать иностранные языки. Во-первых, в них не так много незнакомых нам фонем. Окей, надо потратить некоторое время на тренировку «th», «w» и «r» в английском, но представьте себе изучающего русский иностранца, в родном языке которого нет различия между «б» и «п» или, скажем, между «р» и «л»! Ещё многих страшат сочетания согласных («Кржижановский»), которые мы практически без труда щёлкаем. Во-вторых, обилие грамматических явлений редко сталкивает нас с чем-либо непонятным. А для американца, например, само понятие рода или падежа совершенно неочевидно. Есть языки без личных форм глаголов, есть языки без предлогов.

И всё же морфология

Теперь о морфологии. В принципе, на первый взгляд тут говорить особо не о чем. Автоматические морфологические анализаторы работают хорошо. Разумеется, они не могут самостоятельно определить контекст, и выдают все возможные варианты трактовок (например, слово «русский» может быть как существительным, так и прилагательным). Если кому интересно посмотреть, как работает автоматический анализатор — можно поэкспериментировать на сайте С.А. Старостина. Смею предположить, что едва ли не все морфологические анализаторы русского так или иначе опираются на Грамматический словарь Зализняка. Кроме того, модуль так или иначе должен принимать во внимание регулярность структуры языка, и «угадывать» (по возможности) новые слова. В регулярности русского нетрудно убедиться с помощью известной фразы *«глокая куздра штеко будланула бокра и кудрячит бокрёнка»*. В ней без труда угадываются части речи и словоформы, хотя понятно, что ни одного слова данной фразы в словарях не найти. В общем, не дудонь бутявку.

Существуют и загружаемые модули. Сам я пользуюсь разработками Алексея Сокирко, «обёрнутыми» в удобный интерфейс на сайте Lemmatizer. Хочу обратить внимание на то, что здесь доступен не только анализатор форм, но и синтезатор, позволяющий автоматически сгенерировать требуемую форму слова. Есть, конечно, и некоторые огрехи. Например, меня слегка раздражает нелюбовь анализатора к букве «ё», а также некоторые технические особенности генерации атрибутов. Анализатор умеет и угадывать неизвестные слова, хотя иногда смешно ошибается. Например, он считает, что слово «крокодилица» есть словоформа начальной формы «крокодилицо» :)

Уже говорилось, что морфологический анализатор сам по себе лишь

модуль в составе проекта, но мне кажется, что для изучающих язык он может быть интересен и сам по себе. Я на досуге написал GUI для модулей анализатора, но пока нет времени разослать куда-нибудь, разрекламировать :) Выложил только небольшое описание здесь, но этого явно недостаточно.

Впрочем, я, наверно, нарисовал слишком радужную картину морфологического анализа :) Есть и трудности. Первая — техническая. Далеко не все языки одинаково легко анализируются. Судите сами: упомянутый русский морфологический анализатор Алексея Сокирко оперирует базой данных в 18,5 мегабайт. Его же версия для английского требует лишь 1,6 мегабайт.

Вторая проблема связана с терминологией. Как ни странно звучит (все ведь в школе учились) в морфологии слов не всё так однозначно. Да, все мы знаем, что «стол» — это существительное, «красный» — прилагательное, в русском языке шесть падежей и так далее. Но существует и масса тонкостей, в которых «среди товарищей согласья нет». Например, всё тот же анализатор считает, что «о лесе» и «в лесу» — формы предложного падежа. Хотя многие лингвисты будут настаивать, что вторая форма — это локатив, практически вымерший в русском языке падеж. Есть и другие «реликтовые формы». Например, звательный падеж («Гриш! А Гриш!») Насколько знаю, он достаточно активен в украинском. Есть и частичный падеж, он же партитив: «официант, ещё чаю!» (вместо «чая»). Партитив буйным цветом цветёт в финском, доставляя изучающим массу радостных моментов.

Нет единства по поводу отнесения некоторых слов к той или иной части речи. Например, что такое «нельзя», «пора», «жаль»? На Грамоте предлагают относить их к «предикативам», но общепринятого подхода нет.

Можно спросить, а какая, собственно, разница. Ну разбирает анализатор и разбирает. Считает предикативом — отлично, наречием — замечательно. Видит разницу между «лесу» и «лесе», не видит — это всё игры филологов. К сожалению, с выбранным анализатором вам ещё жить и жить. Если в дальнейшем, допустим, вы будете считать, что в данном контексте предложения может появиться только предложный падеж, сработает ваше утверждение или нет — зависит и от морфологического анализатора. Обзовёт он «локативом» сочетание «в лесу», и плакал ваш предложный падеж :)

Соответственно, однажды выбранный анализатор не факт что легко удастся заменить другим без дополнительных усилий. Например, ещё одна «фича» анализатора Сокирко: он называет глаголы в личной форме («бегаю») глаголами, а в начальной форме («бегать») — инфинитивами. То есть, получается как будто бы две разные части речи. Соответственно, если другой анализатор считает любой глагол глаголом, а инфинитиву добавляет дополнительный флажок «это инфинитив», без конвертера-переходника не обойтись.

Всё, если по морфологии не будет изобилия вопросов, перейдём в следующей части к предложениям.

NLP: проверка правописания —

взгляд изнутри (часть 3)

(Часть 1, Часть 2) В прошлый раз я преждевременно упомянул токенизацию; теперь можно поговорить и о ней, а заодно и о маркировке частей речи (POS tagging).

Предположим, мы уже выловили все ошибки (какие догадались выловить) на уровне анализа текста регулярными выражениями. Стало быть, пора переходить на следующий уровень, на котором мы будем работать с отдельными словами предложения. Разбиением на слова занимается модуль токенизации. Даже в столь простой задаче есть свои подводные камни. Я даже не говорю о языках вроде китайского и японского, где даже вычленение отдельных слов текста нетривиально (иероглифы пишут без пробелов); в английском или в русском тоже есть над чем подумать. Например, входит ли точка в слово-сокращение или представляет собой отдельный токен? («др.» — это один токен или два?) А имя человека? «J. S. Smith» — сколько здесь токенов? Конечно, по каждому пункту можно принять волевое решение, но в дальнейшем оно может привести к различным последствиям, и это надо иметь в виду.

Примерно так я рассуждал на начальных этапах нашего проекта, теперь же склоняюсь к тому, что в задачах обработки текстов частенько приходится подчиняться решениям других людей. Это будет уже ясно на примере маркировки частей речи.

Маркировка частей речи

Зная разбиение предложения на слова, можно уже искать по тексту часто встречающиеся опечатки. Например, переправлять «egg yoke» на «egg yolk» (эта опечатка, видимо, так популярна, что о ней даже упоминает Википедия). Но настоящий прогресс по сравнению с регулярными выражениями обеспечивает маркировка частей речи, то есть сопоставление каждому слову текста его части речи:

«I love big dogs.» -> «I_PRP love_VBP big_JJ dogs_NNS ._.»

В этом примере используются следующие маркеры: PRP — местоимение; VBP — глагол настоящего времени, единственного лица, не третьего числа; JJ — прилагательное; NNS — существительное во множественном числе. Ну а точка — это просто точка.

Зная части речи отдельных слов, можно формулировать более сложные паттерны ошибок. Например, «DT from» -> «DT form». Маркер DT обозначает «определяющее слово» — артикль или указатели вроде this/that. Если в тексте встретилось сочетание «the from» или «this from», скорее всего, это опечатка, и имелся в виду не предлог from, а «форма» — form. Можно ещё хитрее: «MD know VB» -> «MD now VB». Здесь идёт отлов опечатки «know вместо now» — паттерн «модальный глагол + know + глагол». Под него попадает, скажем, фраза «I can know say something more».

Само собой, несложно реализовать простейшие операции, такие как «или» («если встретилось то или это») и отрицание («встретилось не это»). Именно на таких выражениях работает уже упомянутая система LanguageTool. Поскольку распространяется она по лицензии LGPL, я решил перенести все её правила в нашу систему. Почему бы и нет? Люди проделали большую работу, было бы глупо не воспользоваться результатами, коли разрешают. Об ограничениях этого подхода мы ещё поговорим, а пока вернёмся к маркировке частей речи.

Наиболее популярный на сегодня способ POS-разметки сводится к той же самой задаче классификации, на сей раз уже в полном варианте. Мы даём на вход обучающемуся алгоритму слово и его контекст — обычно начальные и конечные символы слова, а также данные о предыдущих словах предложения — сами эти слова и соответствующие им части речи. Также мы сообщаем часть речи слова в текущем контексте, и алгоритм эту информацию запоминает. Если теперь подать на вход контекст, алгоритм сможет сделать разумную догадку о части речи.

Здесь тоже частенько используют модель максимальной энтропии. Хотя можно бы и поиграться с другими алгоритмами. Например, существует разработка на основе support vector machines (SVMTool).

Аннотированные корпусы, великие и ужасные

В прошлый раз я не заострял на этом внимание, но теперь уж точно пора. Чтобы POS tagger заработал, его нужно натренировать на большой коллекции текстов, где каждому слову приписан тег части речи. Тогда возникает резонный вопрос: а где ж эту коллекцию взять?

Такие коллекции («аннотированные корпусы») существуют, хотя их не так много. Чаще всего встречается POS-маркировка, реже — глубокое аннотирование, то есть маркировка синтактико-семантических связей между словами в предложении. Крупнейший глубоко аннотированный корпус английского языка называется Penn Treebank и содержит почти три миллиона слов. Хорошие корпусы существуют также для немецкого и русского — это из тех, что я лично изучал.

Теперь подумаем вот о чём. Существуют языковые тонкости, относительно которых различные лингвисты придерживаются различных мнений. Например, сколько падежей в русском языке? Ответ школьника — шесть, однако я могу назвать, по крайней мере, восемь-девять. В английском языке какой частью речи является слово book в сочетании book market? Я бы сказал, что это прилагательное, но можно отстаивать и трактовку как «существительного в роли прилагательного».

Таким образом, можно по-разному размечать текст, исходя из каких-либо лингвистических или практических соображений. К сожалению, наши соображения вряд ли воплотятся в итоговой системе, ибо пользуясь готовым корпусом, мы вынужденно принимаем правила игры его разработчиков. Если я тренирую POS tagger на корпусе Penn Treebank, придётся смириться, что «book» в роли прилагательного всё равно

трактуется как существительное. Кому не нравится — может создавать свой собственый корпус и размечать его по своему усмотрению.

Аналогично, в Penn Treebank знак препинания всегда является отдельным токеном, поэтому запись «etc.» — это два токена, а «J. S. Smith» — пять токенов, даже если это соглашение для меня неудобно. Выбора нет. Это, кстати, к вопросу о наличии лингвистов в проекте. Если бы у меня были неограниченные бюджеты и куча времени, можно было бы попытаться сделать полностью свою систему, воплощающую наши взгляды на проверку правописания. Однако в реальных условиях существующие NLP инструменты и текстовые корпусы направляют действия по достаточно чёткому маршруту, оставля не так уж и много простора для фантазии.

Да, ещё замечание. Естественно, готовые коллекции содержат корректные тексты, лишённые явных грамматических ошибок. Что это значит для нас? Ну, возьмём тот же POS tagger. Сначала мы его тренируем на корректных текстах (где он никогда не видит сочетаний вроде «I has»), а потом используем его для маркировки слов в текстах с ошибками. Будет ли он столь же хорош в новых условиях? Да кто ж его знает; но создавть корпус с типичными ошибками ради тренировки разметчика для нас слишком большая роскошь.

Продолжим в следующей части.

Заметки об NLP (часть 4)

(Начало: 1, 2, 3) На сей раз хочу немного отвлечься и порассуждать (а точнее, похоливарить) на тему статистических алгоритмов и вообще «обходных путей» компьютерной лингвистики.
В первых частях нашего разговора речь шла о «классическом пути» анализа текста — от слов к предложениям, от предложений к связному тексту. Но в наше безумное время появились и соблазны решить проблему «одним махом», найдя, если угодно, баг в системе или «царскую дорогу».

Кстати, о царских дорогах в науке и учёбе «вообще». Да простят меня читатели за пространную цитату:

> Дезидерий. Как подвигаются твои занятия, Эразмий?
> Эразмий. Кажется, Музы не очень ко мне благосклонны. Но дело пошло бы удачнее, если бы я мог кое что от тебя получить.
> Дезидерий. Отказа ни в чем не встретишь — только бы это было тебе на пользу. Итак, говори.
> Эразмий. Не сомневаюсь, что нет ни единого из тайных искусств, которого бы ты не знал.
> Дезидерий. Если бы так!
> Эразмий. Говорят, существует некое искусство запоминания, которое позволяет почти без хлопот выучить все свободные науки.
> Дезидерий. Что я слышу? И ты сам видел книгу?
> Эразмий. Видел. Но именно что видел: учителя не нашлось.
> Дезидерий. А в книге что?
> Эразмий. Изображения разных животных — драконов, львов, леопардов, разные круги, а в них слова — и греческие, и латинские, и еврейские, и даже из варварских языков.

Дезидерий. А в названии указывалось, за сколько дней можно постигнуть науки?
Эразмий. Да, за четырнадцать.
Дезидерий. Щедрое обещание, ничего не скажешь. Но знаешь ли ты хоть одного человека, которого бы это искусство запоминания сделало ученым?
Эразмий. Нет, ни одного.
Дезидерий. И никто иной такого человека не видел и не увидит, разве что сперва мы увидим счастливца, которого алхимия сделала богатым.
Эразмий. А мне бы так хотелось, чтобы это было правдой!
Дезидерий. Наверно, потому, что досадно покупать знания ценою стольких трудов.
Эразмий. Конечно.
...

Если рассказ заинтересовал, окончание прочтите сами. Это Эразм Роттердамский, «Разговоры запросто» (1524 год). На дворе XXI век, а книги из серии «за 21 день» не переводятся, заметим мы.

Так вот, предпринимаются и попытки анализа текста без какого-либо понимания его структуры. Причём как на уровне синтаксического анализа (создать дерево предложения, ничего не зная о законах построения фраз), так и на уровне дальнейшей работы, например, машинного перевода. Как это в принципе возможно? Ответ заключается в магическом заклинании «статистика».

Блеск и нищета статистики

Статистика — штука замечательная, и имеет массу применений, в том числе и в компьютерной лингвистике. Но не панацея. Поскольку текстов за историю человечества накоплено уже несметное множество, возникает резонный соблазн изучать структуру новых текстов на основе существующих (предположительно, корректных). Надо сказать, что в предыдущих частях я **нигде** ещё не упоминал, каким именно образом строится дерево разбора фразы. Да, говорилось о грамматиках Хомского, но лишь как об идее, *из которой выросла* концепция phrase-structure parsing. Я специально нигде не писал, что грамматика Хомского реально используется для построения таких деревьев. Это необязательно так.

Как можно рассуждать о корректности фразы на основе накопленных данных? Например, так. Есть фраза «я съел пирожное». Давайте посмотрим, часто ли она встречается в существующих документах? А фраза «я съел веник»? Скорее всего, редко. А фраза «я съел пирожному», наверно, вообще не встречается. Отсюда и вывод, что первая фраза корректна, вторая небесспорна, а третья неправильна. Можно искать «коррелирующие словосочетания». Если какие-то слова встречаются друг с другом часто, вероятно, они зависят друг от друга. Так можно построить всё дерево. Хотя заметим, что такая система никогда не объяснит вам, чем именно фраза плоха. Просто сообщит, что так не говорят. Сами понимаете, для изучающего иностранный язык человека это не ахти какая помощь.

Можно пойти ещё дальше. Допустим, требуется перевести документ на другой язык. Какова вероятность, что вашу фразу никто и никогда не переводил? Вероятно, можно разыскать готовый перевод, хотя бы для части фразы. «Базой знаний» в таких проектах служат корпусы двуязычных текстов. Например, очень любят протоколы заседаний канадского парламента, так как ведутся они на двух языках: английском и французском. При этом тексты формальные, перевод строгий, без вольностей. Стало быть берём кусок текста, находим кусок соответствующего текста — и вуаля! (Конечно, я сильно упрощаю реальное положение вещей, но базовая идея такова). Отсюда и получаются шуточки с непонятным переводом. Было made in China, стало «сделано в республике Беларусь». Ну я это типа злободневно пошутил, а на самом деле именно так и происходит.

Вы не подумайте, что я нападаю на статистические алгоритмы в принципе. Существует масса прекрасных идей. Мне, например, нравится мысль анализировать трибанки (treebanks), но об этом в другой раз.

Поверить алгеброй гармонию

А теперь я хочу немного поиграть в «верю — не верю». Во что верю, во что не верю.

Я верю, что с помощью анализа готовых текстов можно многое сделать. Я не верю, что «царская дорога» в машинной лингвистике существует. Лет тридцать назад казалось, что создание программы для игры в шахматы примерно эквивалентно созданию искусственного интеллекта. Нынешние результаты, когда компьютер может обыграть **любого** гроссмейстера, были приняты со смешанными чувствами. С одной стороны, да, успех, а с другой — очевидно, что алгоритмы не очень-то продвинулись, просто компьютеры резко подтянулись, и стало возможно просчитывать миллионы комбинаций и хранить обширнейшие библиотеки готовых партий .

В лингвистике аналогичным способом можно совершить рывок, но я уверен, что у подобного подхода есть теоретический потолок. Как ни крути, по крайней мере, создание «портретов объектов» необходимо. Ну как можно перевести на русский «sibling», если не знать, шла речь о брате или о сестре? Можно набить обширную базу, сделать так, чтобы компьютер переводил Байрона (по известным переводам), но по сути это будет та же Китайская комната Сёрля. Пока знаем входные куски, переводим, а шаг влево, шаг вправо — приехали. Да и машинный перевод — не единственная цель. Целью может быть, например, *понимание текста*, что бы ни стояло за этим термином. Скажем, пополнение базы знаний об описываемом в тексте мире. (Впрочем, это уже разговор о *прагматике языка*, явно не сегодняшнего дня тема).

То есть в каком-то смысле подход того же Google translate вызывает у меня противоречивые чувства. С одной стороны, спасибо за быстрый и удобный сервис. С другой стороны, мне кажется, они сместили «центр тяжести» в сторону статистики. Думаю, через несколько лет они упрутся в максимум, и тогда придётся искать другие методы. Особенно это очевидно для языков со свободным порядком слов и богатой морфологией — тут переводчик

просто сходит с ума, так как вариантов перевода масса, однозначную статистику набрать трудно, и множество разнообразных входных фраз тоже велико.

В конце концов, никому же не приходит в голову писать статистический компилятор Паскаля, хотя программ на Паскале тоже уже написано превеликое множество. Впрочем, в Гугле берут на работу вполне видных компьютерных лингвистов, так что, может быть, и у них будет не всё столь однозначно с используемыми алгоритмами.

Так, получилось как-то ядовито и эмоционально :) Но это ничего, в следующих частях вернёмся к более продуктивному разговору. Хотя и здесь, видимо, не всё сказано. Ну да ладно, напишу продолжение, ежели чего.

Заметки об NLP (часть 5)

Что ж, продолжим. (Первые части: 1 2 3 4). Долго выбирал, что будет лучше для следующей темы — пофилософствовать о прагматике языка или поговорить конкретно об алгоритмах разбора. Учитывая, что предыдущая часть была неформальной, решил всё-таки переключиться на конкретику, а там посмотрим.

Итак, синтаксический анализ предложения. Давайте сразу определимся, что речь пойдёт о разборе в рамках концепции dependency parsing, причём определяющей методологией разбора будет точный анализ (не статистический). Начнём с небольшого обзора происходящего вокруг.

Например, у меня на столе лежит книжка под названием Dependency parsing. По названию и аннотации можно подумать, что нас ждёт детальный обзор существующих методик, но это, к сожалению, не совсем так. Авторы сравнительно быстро «съезжают» к своей теме, и половина книги посвящена их подходу, в то время как многие другие методы даже не упомянуты.

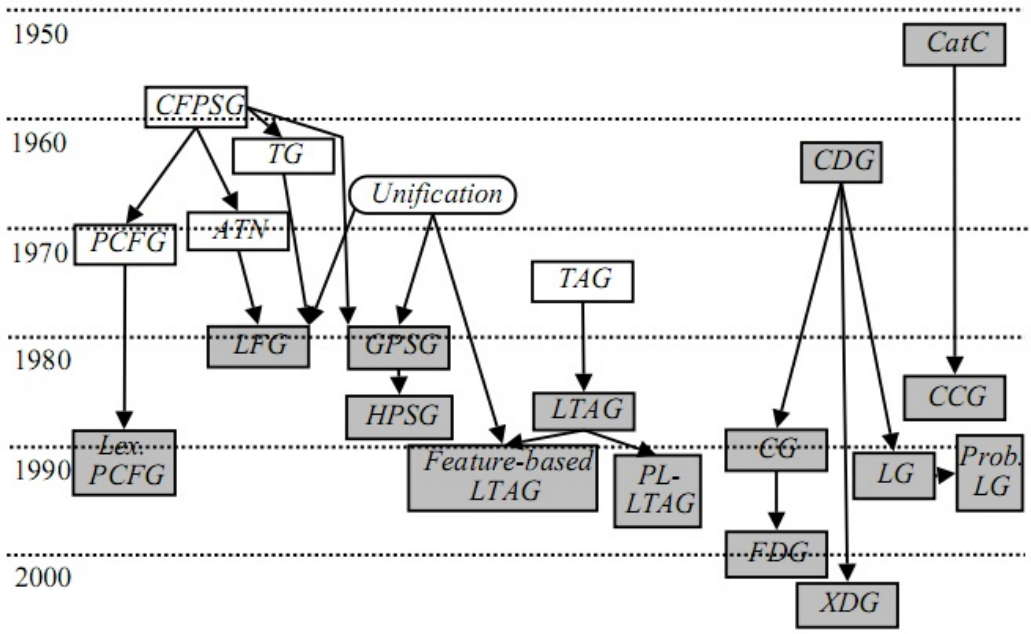
Не подумайте, что я их критикую — в том и штука, что данная книга достаточно характерна для нашего времени. Нынешнее состояние отрасли я бы охарактеризовал как «разброд и шатание». Может, заблуждаюсь, хотелось бы верить в лучшее :) Тому есть масса причин. Каждый естественный язык «особен» по-своему. Можно взять какой-нибудь сомалийский и всю жизнь адаптировать для него известные методы, всегда найдётся место новизне. Сформировались солидные лаборатории со своими сложившимися инструментами — та же группа в Стэнфорде, написавшая Stanford parser. Они менять в своих идеях вряд ли что будут в скором времени. Ко всему прочему, качество подходов трудно оценивать. Целая приличного объёма диссертация моего коллеги посвящена методикам оценки и сравнения алгоритмов синтаксического анализа! А он не такой болтун как я, пишет сжато. *(Кстати, диссер рекомендую — там содержится хороший обзор современных методов разбора. Да, текста много, но лишь потому, что методов много — в других источниках объём будет не меньше)*.

Если почитать книги, особенно старые (пусть год издания не вводит в заблуждение, это перепечатка текста начала восьмидесятых годов),

складывается впечатление, что всё давным-давно сделано. Однако, видимо, многое желаемое выдаётся за действительное, либо не так отполировано, как хотелось бы. Например, поиск морфологического анализатора для финского языка убеждает, что всё сделано ещё в 1984 году известным в этой области специалистом Киммо Коскенниеми. Однако быстро находится и сайт проекта Omorfi — морфологического анализатора финского, который сейчас пишется (и ещё далёк от завершения) в Хельсинкском университете [имени Линуса Торвальдса] под руководством того же Коскенниеми! Это как бы намекает.

Разобраться в одних только формализмах, доступных на сегодняшний день, непросто. А уж понять, какой из них чего стоит — вообще невозможно. Думаю, это вопрос веры.

Вот они, теории синтаксического разбора:



За каждым подходом — своя школа, свои парсеры, проекты… И до сих пор трудно понять, где лидер. Мой любимчик — XDG, но и там не всё хорошо, и повального энтузиазма по поводу этой разработки в научных кругах пока не слышу. Сам пытался читать и про многие другие подходы. Много умного есть в разных теориях, и нередко они пересекаются.

Так что простите, полного обзора тут не будет. Читайте упомянутую диссертацию доктора Какконена. Рисунок я взял оттуда.

Лексикализация

Старые формализмы не спешат на пенсию. Как видно из рисунка, многие достаточно древние методы успешно развиваются и по сей день. Но один тренд чётко ясен: переход к *лексикализованным* (lexicalized) моделям. На рисунке они обозначены серым цветом.

Смысл этого термина достаточно прост: в записях синтаксических правил языка так или иначе фигурируют настоящие слова из словаря языка. В нелексикализованных моделях используются более общие понятия. Например, в лексикализованном правиле может быть сказано: здесь должно находиться слово «стол». В нелексикализованном правиле может быть написано разве что «существительное мужского рода», ну и какие-то уточняющие атрибуты.

В принципе, насколько я понимаю, грань достаточно тонка. С одной стороны, объект может быть так «зажат» условиями, что кроме слова «стол» под его определение ничего и не подойдёт. С другой стороны, в лексикализованных правилах также могут встречаться не только конкретные объекты, но и абстрактные понятия: «субъект», «наречие».

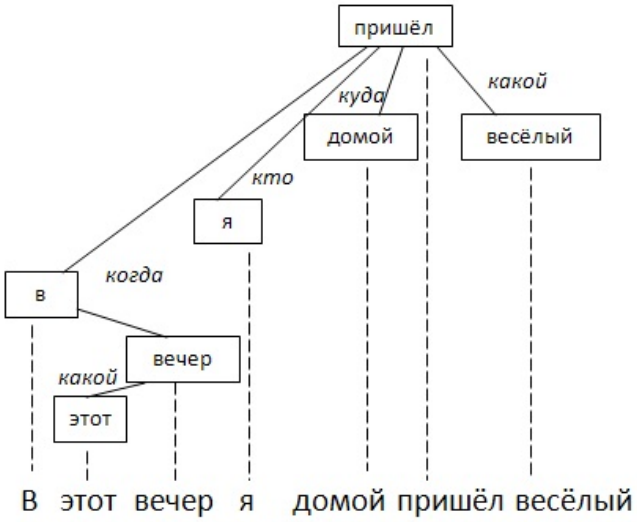
На практике нелексикализованные своды правил («грамматики») ассоциируются с чем-то небольшим по объёму и, вероятно, статистически выведенным. Лексикализованные грамматики — это толстые словари, по уровню детализации доходящие до описания отдельных слов.

Проблемы проективности и множественности деревьев разбора

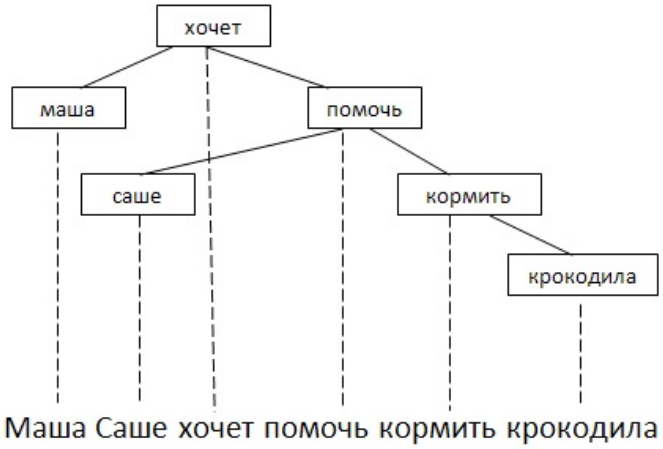
Как видите, хоть обещал разбор, но до разбора всё никак не дойдём. Будем считать, что мы итеративно к нему приближаемся :)
Я уже упоминал, что граф разбора может быть не только деревом (то есть может быть графом общего вида), но это ситуация специфическая, и при чистом синтаксическом анализе она не возникает. Так что, может быть, вернёмся к этому сценарию позже, а пока что будем считать, что при разборе получается дерево.

Так вот, если мне кто-то захочет показать новый парсер (dependency-парсер), первое, что я у него спрошу — поддерживает ли алгоритм построение непроективных деревьев и умеет ли строить всё многообразие вариантов разбора.

Проективными называются деревья, ветви которых не перекрещиваются с проекциями на слова исходного предложения. Такое получается, если всегда объединять в общий узел только соседние элементы. В принципе, в большинстве случаев именно так и происходит:



Но бывает, что надо соединять между собой слова, «перепрыгивая» через третье слово:



(Признаюсь, глагольные «колбасы» типа «хочет помочь кормить» в разных теориях обрабатываются по-разному. На рисунке приведён лишь один из вариантов.)
Насколько понимаю, при phrase-structure-разборе такие ситуации вообще не обрабатываются. Потому что грамматики Хомского по определению описывают лишь непосредственно примыкающие друг к другу элементы. Когда апологеты dependency parsing стали говорить, что этот подход позволяет работать с более свободным порядком слов (чем в английском), хомскианцы ответили: а вы сначала парсер напишите, который умеет строить непроективные деревья — без них выгоды от потенциальной свободы невелики.

Насколько же актуальна проблема? В одной статье пишут, что при анализе реальных текстов на чешском языке «непроективность» была выявлена в 23% случаев. Часто. Проблема непроективных деревьев в том, что если разрешено пытаться клеить любое слово к любому другому, мы тут же выходим за все предусмотренные вежливостью нормы по объёму вычислений. Фактически, в худшем случае получается полный перебор

всех возможных графов с N вершинами-словами, то есть задача экспоненциальной сложности.

Тут подключились теоретики и доказали, что ценой небольших ограничений возможностей парсера можно свести его вычислительную сложность к «приемлемой» (я не изучал детали, точные выкладки привести не могу). При этом для той же чешской коллекции документов «не по зубам» этому ограниченному анализатору будет уже всего лишь 0.5% предложений.

Кстати, первые упоминания о непроективном разборе предложений достаточно свежи — примерно 1997 год. Это к вопросу о том, насколько можно в принципе доверять литературе в нашей области. Особенно этим замечательным книжкам из восьмидесятых, в которых «уже всё решено».

Теперь о множественности разбора. В принципе, это та же самая проблема, но с другой стороны. Если пытаться построить все допустимые деревья разбора, мы проваливаемся в ту же «экспоненциальную яму». Понятно, что вариантов разбора будет два-три, ну четыре. Однако в процессе попыток склеить что угодно с чем угодно возникают очевидные накладные расходы :)

Понятно, что многие слова неоднозначны, но их трактовка не влияет на вид дерева, то есть на синтаксический анализ: «я держу деньги в банке». Здесь разбор однозначен: держу деньги (где?) — в банке. При этом неважно, стеклянная банка у меня или каменное здание банка.

Есть «промежуточные» варианты. «По улице шла девушка с косой». Можно всегда разбирать так: девушка (с чем?) — с косой. А можно выбирать: девушка (какая?) — с косой (если речь о причёске); девушка (с чем?) — с косой (если речь о металлической косе).

Бывают и очевидные случаи совершенно разных деревьев. Мой любимый пример: «он увидел её перед своими глазами».
Первый вариант ясен: он увидел (кого?) её (где?) перед своими глазами. Но есть и «маргинальная» трактовка: он увидел (что?) её перед (т.е. переднюю часть) (чем/как?) — своими глазами.

Не буду врать, но кроме XDG/XDK мне не приходит на ум проектов, умеющих строить всё множество допустимых деревьев.

Пожалуй, на сегодня закончим. Спать хочется :)

Заметки об NLP (часть 6)

(Первые части: 1 2 3 4 5). Надеюсь, разговор о естественном языке читателей ещё не утомил! По-моему, тематика действительно интересная (хотя популярность топиков явно идёт на убыль :)). Что ж, посмотрим, на сколько частей меня ещё хватит. Думаю, экватор мы уже прошли, но три-четыре темы затронуть ещё можно.

На сей раз заметка полностью посвящена проекту XDG/XDK, который я пытаюсь изучать на досуге. Назвать себя специалистом по XDG пока ещё не могу. Но потихоньку двигаюсь.

Итак, мы остановились на том, что будем рассматривать dependency parsing, причём, по возможности, с непроективными конструкциями и множественной генерацией деревьев. Это естественным путём приводит нас к проекту XDG/XDK, удовлетворяющему все эти требования. Я сам пришёл «в тему» с симпатиями к lexicalized dependency parsing, но в остальном начинаю почти что с чистого листа, и выбор XDG продиктован холодным отсеиванием прочего. Изучал, что есть, выбирал, что лучше. На сегодня так оказалось, что XDG вроде как лучше (по крайней мере, в рамках заданной концепции).

Что же это такое? XDG расшифровывается как Extensible Dependency Grammar, а XDK, соответственно, как XDG Development Kit. XDG представляет собой формализм для описания допустимых связей между словами в предложении. Это лексикализованный формализм: для каждого конкретного слова языка придётся описать, каким же образом и с чем его можно клеить. XDK включает в себя набор инструментов, наиболее важным из которых является Solver, он же синтаксический анализатор. Solver принимает на входе грамматику языка и предложение на нём. На выходе генерируется множество деревьев, соответствующих разобранному предложению. Как видите, на словах всё очень просто :)

Сочинение грамматики

Откуда взять грамматику? Вот здесь я хотел бы подчеркнуть: да, в простейшем сценарии (хотя для разработчика он явно не самый простой :)) грамматика пишется вручную. Однако не будем забывать, что грамматические правила представляют собой лишь точное формальное описание языковых конструкций. И откуда они получены — написаны экспертами или выведены каким-то образом из существующих текстов — вопрос второй. Как бы то ни было, для точного анализа предложений нужны правила, и грамматика — всего лишь разновидность их записи.

Что касается способа записи этих правил — мне он представляется достаточно очевидным и адекватным. Полагаю, что с помощью XDG можно описать не только естественный язык, но и, скажем, Паскаль. Конечно, такой парсер будет далёк от совершенства, но работать, по идее, должен :)

Даже не знаю, почему раньше никто за такую работу (dependency parser kit) не брался. Проект-то достаточно свежий, текущая версия датирована 2007 годом (впрочем, я уже говорил, что особых восторгов по поводу XDK не замечаю пока, и на данный момент проект не развивается).

Наверно, имеет смысл дать немного «пощупать» язык предлагаемых грамматик. Он зиждется на трёх базовых принципах:

Лексикализация. Элементами грамматики являются слова языка. Каждому слову может быть приписан набор атрибутов, собственно, и определяющий, что слово может приклеить к себе и к чему оно может приклеиться само. У каждого атрибута есть имя, тип и значение. Чаще всего используются типы «строка» и «множество строк».

Принципы создания графа. Помимо ограничений на «склейку», задаваемых на уровне отдельных слов, можно задать ограничения для всего графа разбора фразы. Такие ограничения называются «принципами». Простейший пример принципа — tree principle, требующий того, чтобы генерируемый граф был деревом. О некоторых других полезных принципах поговорим чуть позже.

Многомерность. Каждый атрибут слова определяется в некотором пространстве имён (задаваемом пользователем). Принципы создания графа также описываются в определённом простанстве имён. Каждое такое пространство называется «измерением». Граф разбора фразы для каждого измерения строится отдельно. Таким образом, можно получить сразу несколько графов разбора, отражающих те или иные структурные особенности предложения (то есть пользователь может посмотреть на предложение как бы под разными углами зрения). Например, наряду с dependency-графом можно построить вырожденный граф-список, показывающий порядок следования слов в предложении. Или, скажем, граф, соединяющий между собой все существительные фразы.

Принцип валентности

Теперь пару слов о том, какие бывают принципы. Хочется прежде всего упомянуть *принцип валентности*, указывающий, что связь между двумя словами w1 и w2 может быть установлена только если атрибут **out** слова w1 совпадает с атрибутом **in** слова w2. Поясню на примере:

```
defentry { dim lex { word: "eats"}
           dim syn {in: {root} out: {subj obj adv*}}}
defentry { dim lex {word: "Peter"}
           dim syn {in: {subj? obj?}  out: {} } }
defentry { dim lex {word: "spaghetti"}
           dim syn {in: {subj? obj?} out: {} } }
defentry { dim lex {word: "today"}
           dim syn {in: {adv?} out: {} } }
```

Спецификация атрибута **out** глагола «eats» указывает, что слово может присоединить к себе один субъект (subj), один объект (obj) и произвольное количество обстоятельств (adv*). Спецификация атрибута **in** слова «Peter» разрешает этому слову быть субъектом или объектом. Такова же и спецификация слова «spaghetti». Слово «today» описано как потенциальное обстоятельство. Таким образом, если мы скормим парсеру фразу «Peter eats spaghetti today», итоговое дерево будет иметь «eats» в качестве корня, «Peter» и «spaghetti» будут присоединены к корню в качестве субъекта и объекта, а слово «today» — в качестве обстоятельства .

Существует ещё «принцип упорядоченности» (пожертвую им ради экономии места), с помощью которого можно указать, что более раннее слово в предложении имеет приоритет в качестве субъекта (таким образом, «spaghetti» субъектом не станет).

Принцип согласуемости

Это ещё один важный принцип, мимо которого пройти не могу. Он устанавливает следующее требование: связываемые слова должны согласоваться по какому-либо атрибуту. Например, можно указать, что глагол «eats» в качестве субъекта требует третье лицо, единственное число. В то же самое время глагол «eat» присоединит что угодно, кроме третьего лица, единственного числа:

```
defentry { dim lex { word: "eats" }
           dim syn { agrs: {["3" sg]}
                     agree: {subj} } }
defentry { dim lex { word: "eat" }
           dim syn { agrs: {["1" sg] ["2" sg]
                       ["1" pl] ["2" pl] ["3" pl]}
                     agree: {subj}}
```

Чтобы слово «Peter» присоединялось и теперь, придётся указать его атрибуты числа и лица:

```
defentry { dim lex { word: "Peter" }
           dim syn { agrs: {["3" sg]}
                     agree: {}} }
```

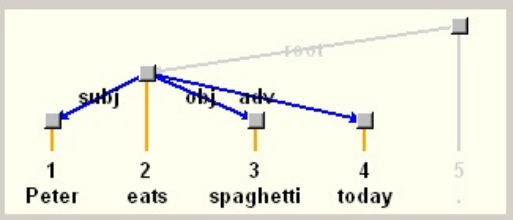
Как видно из этих примеров, грамматика напрямую не поддерживает морфологию. Так, слова «eat» и «eats» для неё разные. Впрочем, я и не совсем понимаю, как в универсальный формализм можно запихнуть поддержку «морфологии вообще». Решается эта проблема, однако, достаточно просто: надо прогонять входное предложение через морфологический анализатор и генерировать правила грамматики (с правильными окончаниями и атрибутами слов) автоматически, «на лету».

Разбор заканчивается тогда, когда удалось собрать всю конструкцию и присоединить её к слову, имеющему атрибут «вершина дерева». Например, в качестве такого слова можно описать точку (ибо точкой кончается предложение):

```
defentry {
  dim lex {word: "."}
  dim syn {in: {}
           out: {root}
           agrs: top
           }}
```

Пуск!

Если полученную грамматику (разумеется, тут приведены лишь её фрагменты!) скормить анализатору вместе со входной строкой «Peter eats spaghetti today .», на выходе будет сгенерировано вполне ожидаемое дерево:



Здесь предлагаю поставить если не точку, то точку с запятой, потому что «кратко» продолжить не получится. Следующий «блок мыслей» тянет на отдельный пост (завтра, вероятно?..) Продолжим про XDG, скорее всего.

Разве что, дабы не возвращаться к этому вопросу, перечислю проблемы XDK, бросающиеся в глаза. Во-первых, нынешний билд собран под ASCII с поддержкой западноевропейских языков. То есть кириллица превращается в крякозяблы (разбирать всё равно будет, но дебажить неудобно). Впрочем, проблема это небольшая, как мне кажется. Во-вторых, автор защитил диссер и ушёл работать в промышленность, больше он проект не поддерживает, а никто другой пока не подобрал, даром что open source. В-третьих, «войти в проект» не так-то просто, ибо всё написано на сравнительно маргинальной среде программирования Mozart/Oz в парадигме constraint programming. За это как раз автора сложно критиковать — задача уж очень специфическая. Я уже говорил, что полноценный разбор тянет на NP-полную задачу. Автор же рассмотрел парсинг как constraint satisfaction problem: даны узлы графа и дан набор условий, ограничивающих рёбра. Требуется найти удовлетворяющий условиям граф. То есть задачу поставили в более общем виде, и в итоге было решено применить специализированный инструмент для такого рода вещей. Правильно сделали. Есть ещё кое-что, не хватающее лично мне, но об этом в следующий раз.

Заметки об NLP (часть 7)

(Первые части: 1 2 3 4 5 6). Как и обещал вчера, продолжаем обсуждать XDG и движемся к следующим темам. Возможно, мы двигаемся слишком быстро, и действительно имело бы смысл публиковать одну статью раз в два-три дня, чтобы оставалось время всё обсудить. Но, наверно, пока «бензин есть», я буду продолжать писать. А потом можно будет вернуться и обговорить ранее освещённые вопросы. Мне кажется, что в компьютерной лингвистике разные темы настолько тесно связаны друг с другом, что разговор об одной из них без связи с другими малопродуктивен. А мы ещё не обо всём беседовали, так что лучше охватить взглядом как можно больше аспектов компьютерного анализа текста, а потом уже рассуждать о конкретике в рамках общей картины происходящего.

Ещё об XDG

В принципе, самые главные свойства и возможности XDG мы уже рассмотрели. Остальное проходит по разряду «вкусностей». Например, можно указать набор атрибутов, и приписать их не конкретному слову, а «классу». Далее слово попросту объявляется экземпляром этого класса и автоматически получает перечисленные атрибуты.

Из важного: продуманы количественно-качественные вопросы присоединения зависимых слов (в примере это уже было, но хочу упомянуть явно). Так, для глагола можно указать, что у него только один субъект, только один объект и сколько угодно обстоятельств (где, когда, почему). Для каждого присоединяемого слова указывается свой набор согласуемых атрибутов. Скажем, тот же глагол согласуется по лицу и числу с субъектом. Да, ещё: одно и то же слово может быть описано несколько раз в разных контекстах. Например, «столовая» — это и прилагательное, и существительное.

Также достаточно гибок механизм вывода результатов. Деревья можно рисовать на экране, выводить в текстовый или xml-файл. Принципы (такие как valency principle, tree principle) можно разрабатывать самостоятельно, пополняя библиотеку проекта. Для этого, конечно, придётся освоить Mozart/Oz :)

Так что скажем спасибо автору этого проекта Ralph'y Debusmann'y и больше копать XDG вглубь не будем. Маленькая подсказка для желающих ознакомиться с XDG поближе: читать лучше всего не только мануал, но и диссертацию автора. Мануал-справочник не очень годится в качестве учебника, читать его тяжело. А диссертация — наборот, аккуратно знакомит читателя с идеями XDG на простых примерах.

Трибанки

Теперь ещё пару слов о таком важном явлении как трибанки (treebanks). К сожалению, в них я пока слабо разбираюсь — всё руки не доходят.

Мы тут уже рассуждали о том, откуда берутся правила грамматики. Понятно, что их можно либо сочинять вручную, либо «каким-то образом» извлекать из существующих текстов. Простые статистические подходы (наводящие на меня тоску) просто анализируют чистый текст в первозданном виде, опираясь на довольно условные «эвристические» критерии. Например: стоящие рядом слова с большей вероятностью зависят друг от друга, чем слова, стоящие поодаль. Мне кажется, далеко на таких принципах не уедешь; для себя я пользуюсь таким критерием: можно ли на идеях предлагаемого парсера естественного языка написать парсер несравнимо более простого Паскаля? Если нельзя, то, по-моему, тут и говорить не о чем.

Но статистический метод можно «натравить» и на специально подготовленный текст, а это уже совсем другое дело. *Трибанк* — это коллекция разобранных предложений (то есть графов разбора),

подготовленных вручную. По-моему, именно такие банки и должны использоваться для автоматического извлечения правил генерации деревьев.

Логичным образом трибанки делятся на phrase-structure treebanks и dependency treebanks. Вероятно, самый известный пример банка первого рода — Penn treebank, синтаксис аннотаций которого является де-факто стандартом для «хомскианских» трибанков. Dependency treebanks — явление более свежее, их пока явно меньше. Наверно, чаще других упоминается The Prague Dependency Treebank чешского языка. Есть и другие, конечно (см. проекты со словом «dependency» в названии). Для русского, как обычно, какая-то работа «в процессе», но ничего конкретного сказать пока нельзя.

Я пока в трибанки глубоко не лезу, потому что там полно технических особенностей. Как выполняется аннотирование, каков синтаксис, что конкретно описывается, а что не описывается и так далее. В общем, есть такое ощущение, что можно закопаться с головой в тему и не выкопаться. Так что закапываясь, лучше бы точно знать, чего ищешь, а я пока ещё этого для себя не решил.

Но что для меня интересно, есть попытки автоматически извлечь из трибанка правила в форме XDG. Правда, этот опыт трудно назвать очень удачным. Автор пишет, что правила всё равно получаются слишком «вольными», и XDK слишком многое позволяет себе клеить друг к другу. Может, трибанк недостаточно велик, трудно сказать. Он ещё пишет, что в XDG нет «вероятностных» правил. Если в трибанке хотя бы однажды что-то встретилось, это уже правило того же уровня, что и правило, полученное из сотен примеров. Впрочем, думается, это не проблема XDG. Если правило такое редкое, лучше его в грамматику вообще не вставлять. Ещё жалуются, что правил получается превеликое множество, и грамматика «разбухает». На это повторю свой старый тезис, что грамматику можно генерировать для каждого конкретного предложения.

Пожалуй, для меня самым сложным вопросом в этой теме остаётся такой: допустим, правила из трибанка извлекли, и они даже позволяют получить хороший разбор текста. А что дальше? Поможет ли такой парсер в дальнейшем анализе текста? Мы это сейчас обсудим.

Прагматика

Честно говоря, тема подкралась неожиданно :) Я хотел сначала рассказать о своих идеях использования XDG, но лучше отложим их до следующего раза. А здесь остаётся как раз немного места для слегка отвлечённого философствования.

Мы уже говорили о *синтаксисе* предложения. Понимать синтаксис — значит уметь строить граф синтаксических зависимостей между словами. На следующем уровне понимания находится *семантика* — понимание смысла предложения. Под «пониманием» могут скрываться самые разные вещи, но давайте пока оставим этот термин как есть. Следующим уровнем является *прагматика*, отвечающая за использование языка.

Когда вы пишете компьютерную программу, почти вся обработка исходного текста компьютером так или иначе сводится к преобразования одного вида текста в другой вид текста, то есть, фактически, к машинному переводу. Программа на языке высокого уровня может быть превращена в C-код, далее C-код может быть автоматически соптимизирован и превращён в текст на ассемблере или сразу в машинные инструкции. На входе текст, на выходе текст — этим и занимаются все компиляторы, трансляторы, оптимизаторы и иже с ними. Но в самом конце этой цепочки возникает этап, ради которого всё и затевалось: *выполнение* программы процессором. То есть набор пассивных данных (текст) превращается в управляющую последовательность, запускающую некий *активный процес с*. В этом и состоит смысл языка программирования: управлять действием.

С естественным языком всё значительно сложнее. Если я пишу компилятор Паскаля, я твёрдо знаю конечную цель: это машинный переводчик, транслирующий программу на Паскале в программу в машинных кодах, исполняемую центральным процессором. Если я пишу парсер (то есть пока лишь синтаксический анализатор — относительно ранний и простой модуль компилятора!) естественного языка, надо чётко решить для себя, что будет дальше, на следующих этапах. Даже если вопрос с семантикой/смыслом текста будет успешно решён, всё равно не ясно, что будет происходить за семантикой.

На практике очевидны три применения обработки текста. Первое: машинный перевод. Здесь вопрос о прагматике даже не возникает — текст переведён на другой язык, и пусть читатель сам думает, что с ним дальше делать :) Второе: пополнение некоей «базы знаний» или «базы фактов» на основе сведений, содержащихся в тексте. Третье: речевой интерфейс. Здесь по сути естественный язык (вернее, его узкое подмножество) заменяет собой язык программирования. Открыть окно, нажать кнопку, запустить приложение.

Наверняка можно придумать и массу других применений — от игровых и учебных до аналитических. Можно играть в «съедобное — не съедобное»: человек вводит «я съел X», а компьютер, в зависимости от X, пишет, верит он вам или не верит. Можно использовать парсер как часть системы проверки правописания или распознавания речи: допустим, при распознавании мы не можем понять, сказал человек «мой конь и ржёт, и скачет» или «мой кони ржёт и скачет». Парсер сразу поймёт, что «мой кони ржёт» — конструкция некорректная, и поможет системе выбрать первый вариант.

Штука в том, что дальнейший сценарий использования (языка или парсера) оставляет отпечаток на самом парсере, как бы странно это ни звучало. Например, если мы играем в «съедобное — не съедобное», можно настроить грамматику (XDG, допустим) так, чтобы при попытке обработать конструкцию вида «я съел *'несъедобное'*» происходила ошибка разбора.

Менее надуманный пример — система машинного перевода. Фразы «они разбили лагерь» и «они разбили машину» разбираются совершенно одинаково — субъект-глагол-объект. Однако дальше возникает другая проблема: что такое «разбили»? Первое «разбили» на английский следует переводить как «set up», второе как «crash». То есть синтаксически мы вроде бы разобрались, но в переводе нам это несильно помогло. При

этом, заметим, синтаксический анализатор, не видящий разницы между «set up» и «crash» всё равно прекрасно сгодится для проверки правописания русского. Разбили и разбили — сад, палатку, машину, чашку, сердце. Какая разница, как это по-английски?

В этом я вижу некоторую проблему (хотя могу и заблуждаться) для автоматического извлечения правил из трибанка: вся эта работа сконцентрирована вокруг задачи получения правильного дерева разбора, но что с этим деревом делать дальше, не обсуждается. Можно возразить: задача парсера — парсить, а что там с семантикой, пусть следующие модули в цепочке разбираются. Однако если следующий модуль (скажем, семантический анализатор) будет столь же сложен, что и парсер, толку с такого парсера? Мне кажется, что на этапе синтаксического анализа хорошо бы выжать максимум информации из текста в рамках поставленной задачи.

Есть попытки «объективно» оценить качество того или иного парсера, но всё обычно сводится к тому, генерирует ли парсер правильные деревья разбора или нет.

Так, на сегодня ставим точку. Завтра (ну или когда будет настроение :)) продолжим. А может, на следующей части и закончим :)

Заметки об NLP (часть 8)

(Первые части: 1 2 3 4 5 6 7). В этой части я расскажу о синтактико-семантическом анализаторе — как я его вижу. Обратите, кстати, внимание на часть 7 — она до главной страницы не добралась, так что не уверен, что все интересующиеся её видели.

За пределами синтаксического анализа

В прошлой части я затронул, как мне кажется, важную тему: какого рода информацию, помимо древовидной структуры, требуется извлекать из предложения? В предложении «он разбил мамину чашку» знание структуры фразы позволяет понять, что прилагательное «мамину» относится к слову «чашку», стало быть, при переводе на английский его следует переводить как притяжательный падеж: «mother's». Однако одной структуры недостаточно, чтобы найти правильный перевод для глагола «разбил» из едва ли не десятка альтернатив.

Отчасти эту проблему можно решить с помощью анализа семантики, то есть изучая наряду с синтаксисом фразы смысл входящих в него слов. Здесь я прежде всего хочу отдать должное учителю — Виталию Алексеевичу Тузову, чьим аспирантом я в своё время был. Мои взгляды на проблематику во многом сформировались под влиянием его идей, и основные мысли, которые я собираюсь изложить, либо принадлежат ему, либо, по крайней мере, известны мне благодаря его посредничеству.

Прежде всего очертим круг задач. Начнём с относительно простого — word sense disambiguation, то есть определения смысла слова в данном конкретном контексте на уровне фразы (UPD: похоже, в сегодняшней части дальше этого мы и не продвинемся :)). Далеко не все слова могут быть однозначно поняты в пределах предложения. «По улице шла девушка с косой». О какого рода косе идёт речь — мы пока знать не можем. «I have a sibling» — перевести слово «sibling» как «брат» или как «сестра»? Это тоже неизвестно в рамках локального контекста.

Однако многие вещи вполне «по зубам» и анализатору, разбирающему текст на уровне фраз. Например, перевод слова «разбил» явно зависит от объекта. Если мы знаем, что разбили, можем перевести и глагол. (Тузов любил такую аналогию: чему равен sin(x)? На этот вопрос можно ответить тогда и только тогда, когда известен x).

Дальнейшие действия в общем-то напрашиваются. Нужно ввести онтологию (иерархию понятий), и каждому слову приписать «класс». Далее в зависимости от класса можно делать те или иные выводы о переводе или, более широко глядя, о смысле слова.

Возвращаясь к примеру со словом «разбил», можно включить в грамматику языка следующие правила:

```
посуда    ЧАШКА()      cup
транспорт АВТОМОБИЛЬ() car
люди      АРМИЯ()      army

РАЗБИВАТЬ(субъект, объект:посуда)     to break
РАЗБИВАТЬ(субъект, объект:транспорт)  to crash
РАЗБИВАТЬ(субъект, объект:люди)       to defeat
```

Здесь «посуда», «транспорт» и «люди» — это типы данных в иерархии. Такая строгая типизация позволяет описывать и устойчивые выражения достаточно естественным образом:

```
property/bald BALD()               лысый
property      WHITE()              белый
bird          EAGLE(property)      орёл
bird          EAGLE(property/bald) белоголовый орлан (а не "лысый орёл" :) )
```

Здесь, конечно, нужны не просто типы, а именно древовидная структура, чтобы можно было указывать требуемый уровень детализации типа объекта. В одних случаях требуется чёткое указание подкласса, в других достаточно и надкласса.

Например, в приведённом выше примере только «bald eagle» переводится как «белоголовый орлан», в то время как другие прилагательные (не-bald) не изменят перевода по умолчанию «орёл» (т.е. «white eagle» будет просто «белый орёл»).

Аналогично можно решить проблему «девушки с косой»: если коса русая,

например, то это волосы, а если металлическая — сельскохозяйственный инструмент. Если она просто коса или, скажем, «чёрная» — вопрос остаётся открытым.

Иерархия: мечты и реальность

Если принять предлагаемую концепцию (впрочем, я не вижу в ней ничего радикально расходящегося с житейской мудростью), во весь рост встаёт тяжёлая проблема категоризации всего и вся. И вправду, возможно ли построить разумное дерево «всех объектов мира»? По этому вопросу я думаю следующее. Для начала, такие работы уже есть (EuroWordNet, WordNet). Впрочем, как раз к «универсальной каталогизации всего» я отношусь довольно прохладно. По зрелому рассуждению становится более-менее ясно, что иерархия объектов для конкретного синтактико-семантического анализатора зависит от двух тесно связанных вещей: (а) поставленной задачи; (б) картины мира анализируемого языка.

В чистом виде «поставленная задача» влияет на детализацию дерева в ту или иную сторону. Можно разработать анализатор с глубокой иерархии транспортных средств от велорикш до ракет. Можно, напротив, ограничиться единственным классом «транспортные средства». Можно создать анализатор, в котором все объекты будут делиться на «большие», «маленькие» и «абстрактные».

В задаче машинного перевода «поставленная задача» уже резко пересекается с языковой картиной мира. Для англичанина слово «разбил» переводится десятью способами, поэтому при создании русско-английского переводчика придётся сочинять множественные описания для слова «разбивать». Если же в качестве целевого языка перевода выступает какой-либо язык с менее развитой системой альтернатив для «разбивать», описаний будет меньше.

Картина исходного (анализируемого) языка тоже имеет значение. Например, если англичанин куда-то едет, для него это всегда «to»: to travel to Moscow, to travel to England, to travel to Cyprus. Для нас с вами существуют, по крайней мере, «на»-места и «в»-места: поехать в Москву, поехать на Кипр, поехать в Крым, на Сахалин. Логики здесь не так уж много; есть, конечно, общие правила вроде того, что остров — это «на», а страна, регион — «в». Но говорим же мы «на Руси» (как и «на Украине», даже если не все украинцы это любят — постараемся всё-таки вынести язык за пределы политической сферы).

При этом картина одного языка, естественно, может заметно отличаться от картины другого. В финском тоже есть «на»-места и «в»-места. Подбираются они по своим критериям, к критериям русского языка отношения не имеющим (в основном тот же рандом, что и у нас).

На самом деле из всех этих наблюдений я делаю сравнительно оптимистичный вывод: нам не нужна «универсальная классификация всего и вся» (которая всегда будет небесспорной и вряд ли практически достижимой) — требуется лишь классификация в рамках языкового поля. С одной стороны, это плохо: для каждого языка нужно строить свою иерархию. С другой стороны, это хорошо: языковая картина мира

однозначно может быть изучена на основе существующих текстов, и, думается, построение требуемой иерархии, по крайней мере, теоретически достижимо.

Да, напрашивается вопрос: а иерархия ли? Может, это более сложная структура, своего рода «множественное наследование» с одновременной принадлежностью объектов к разным ветвям иерархии? Честно говоря, не знаю. Пока остановился на иерархии, навскидку не могу привести примера, когда древовидной системы классов было бы недостаточно. Но могу допустить, что и такое бывает.

Здесь, кстати, вылезает одна из проблем XDG: иерархические типы не поддерживаются. Впрочем, для себя я нашёл выход: надо генерировать отдельные типы для всех возможных надклассов. Например, существует ветвь «объект-материальный-посуда». Тогда в рамках XDG слово «чашка» из класса «посуда» превращается в три описания:

```
объект                        ЧАШКА()
объект/материальный           ЧАШКА()
объект/материальный/посуда    ЧАШКА()
```

Всё, на этом закончим. В следующей, вероятно, последней части собираюсь рассказать о «семантическом языке» и немного порассуждать о том, куда в нашей области можно пытаться копать, какие есть сложности в «научно-политическом» отношении.

Заметки об NLP (часть 9)

(Первые части: 1 2 3 4 5 6 7 8). Да возрадуются минусующие, сегодня представляю вниманию читателей последнюю, по всей видимости, часть «Заметок». Как и предполагалось, мы поговорим о дальнейшем семантическом анализе; также я порассуждаю немного о том, чем в принципе можно заняться в нашей области и какие есть трудности «научно-политического» характера.

В предыдущей части мы обсуждали «синтакстико-семантический анализатор», при разборе фразы обращающий внимание не только на синтаксис фразы, но и на смысл входящих в предложение слов. Тогда мне показалось, что проще всего приравнять «семантику» к переводу: умеем перевести слово, значит, понимаем его смысл.

Надо сказать, что в некоторых случаях знание базовой семантики слова необходимо даже для синтаксического анализа, если под синтаксическим анализом понимать не только установление связи между словами, но и выявление её роли. Например, фразы «Иван пришёл из дома» и «Иван пришёл из вежливости» синтаксически устроены одинаково. Однако в первом случае «из дома» — это обстоятельство места (откуда), а во втором — причины (почему). Если не видеть разницы между «домом» и «вежливостью», определить роль члена предложения не получится.

Однако это так, ремарка в сторону. Основной вопрос в другом: существует

ли другой способ определения «смысла» кроме перевода на иной язык? По всей видимости, слова можно определять только через другие слова, и в этом проблема. В комментариях уже обсуждались «языки-посредники» — интерлингва и аймара. По сути дела, их назначение как раз и состоит в «эталонном» описании семантики слова. Слова естественного языка (русского, английского и т.д.) каким-либо образом переводятся на интерлингву, а далее с интерлингвы на целевой язык. Соответственно, задача интерлингвы — хранить смысл слова, полученного из языка-источника, и отображать смысл на целевой язык.

Однако есть и попытки подойти к задаче более научно, а именно разработать специальный формальный язык для описания смыслов. Мой научный руководитель (Виталий Алексеевич Тузов) сам занимался этим направлением. Понятно, что его подход мне известен лучше всего. Впрочем, думаю, что между различными подходами больше общего, чем разного.

Толково-комбинаторный словарь

Наверно, будет честно ещё до Тузова упомянуть об известном Толково-комбинаторном словаре Мельчука. Это поистине грандиозное начинание: не могу точно сказать, сколько слов русского языка в него вошло, но для французского за 20 лет работы (не в курсе, насколько активной) было описано всего около пяти сотен слов. То есть дело трудное.

Смысл работы таков. Любой толковый словарь трактует слова неформально: одно определяется через другое. В известной степени, это требует от нас определённого культурного «бэкграунда», иначе мы рискуем попасть в ловушку лемовских сепулек, то есть ходить по кругу, вместо искомого смысла получая себе на голову новые и новые слова.

Вот математика, скажем, работает не так. Имеются аксиомы, их немного. А дальше с помощью строгого языка рассуждений над этими аксиомами надстраиваются теоремы. Можно ли описывать таким же образом слова русского языка? Давайте посмотрим, что получилось у Мельчука:

АРЕСТОВЫВАТЬ

АРЕСТО́ВЫВА|ТЬ, *ю, ет*, несов.

1. *X арестовывает Y-а за Z* = орган власти 1 (в лице своего представителя) X или люди X, претендующие на власть 1, лишают человека Y свободы за действия Z или с целью предупредить нежелательные для X-а действия Y-а.

Ср. БРАТЬ В ПЛЕН, ИНТЕРНИРОВАТЬ, ТЮРЬМА.

1 = X [кто лишает свободы]	2 = Y [кого лишают свободы]	3 = Z [за какие действия лишают свободы]
$S_{им}$	$S_{вин}$ обязательно	*за* $S_{вин}$

1) Часто M_1 = $\varnothing^{люди}$, А. – сов.

Полиция арестовывает студентов (за хранение нелегальной литературы); *Его арестовали.*

Syn_3	:	разг. *брать*, разг. *забирать*
Syn_c	:	разг. *хватать*, прост. *сцапать*, прост. *зацапать*
Syn_n	:	*задерживать* // ; *сажать* //
$Conv_{2\flat}$	:	*попадаться* //
$Conv_{21\flat}$	:	*попадать в руки* [$S_{род}$]
Anti = $Liqu_{(1)}$Result	:	*освобождать*
Gener	:	*лишать свободы*; *репрессировать*
Ver	:	*на законном основании*
AntiVer	:	*незаконно*
Perf	:	*арестовывать*
S_0Perf	:	*арест 1*
S_2Result	:	*арестованный*, устар. *арестант*

Полиция арестовала преступника прямо на борту самолета. Он был арестован при попытке перейти границу.

Видите — слово описано как можно более формально, даже с буквенными переменными :) Обратите внимание на эти чеканные формулировки:

СВИ́НСТВ|О, *а*, мн нет, ср.

1. бран. Грязь и беспорядок, к которым говорящий относится отрицательно, каузированные живым существом, поступившим, как свинья 2б.

(«каузированное» значит «вызванное»). Заметьте, человек поступает даже не просто «как свинья», а как «свинья 2б»:

2б. бран. Нечистоплотный человек, к которому говорящий относится отрицательно, – как бы свинья 1а по нечистоплотности [по коннотации 2].
Ср. КОЗЁЛ 2.

Соответственно, упомянутая здесь «свинья 1а» описана следующим образом:

СВИН|ЬЯ́, *ьи́, сви́ньи, ье́й, сви́ньям*, одуш, жен.
1а. Всеядное животное, в треть роста человека, толстое, коротконогое, с вытянутой тупой мордой и маленькими светлыми глазами – домашнее животное, функция которого состоит в получении от него мяса, жира, шкуры и щетины.
Коннотации: 1) чрезмерная телесная толщина; 2) нечистоплотность; 3) неразборчивость, всеядность; 4) примитивное наглое поведение, хамство; 5) неблагодарность.
Ср. КОРОВА, ЛОШАДЬ, ...; ПТИЦА 2; СОБАКА, КОШКА.

Syn : разг. *хрюшка*, детск. *хрю-хрю*; прост. *чушка*; ласк. *свинка 1*

К счастью, Мельчук — человек известный, он не нуждается в моих комплиментах. Но в любом случае хочу сказать, что сама идея такого словаря очень прогрессивна, и независимо от той роли, которую словарь в итоге займёт в науке и культуре XX века, я бы отдал дань уважения этой работе. Вот вы подумайте, многие ли гуманитарии способны таким вот математическим способом излагать толкования слов?

Помимо строгого описания слов, Мельчуку принадлежит ещё одна прекрасная идея: лексические функции. Суть их в том, что мы используем самые разные слова для описания одной и той же семантической операции. Например, у Мельчука используется функция Magn(), «усиливающая» переданное ей слово-аргумент. Чего общего между словами «тяжёлый», «бурный» и «большой»? А вот что:

```
Magn(болезнь) = тяжёлая болезнь
Magn(аплодисменты) = бурные аплодисменты
Magn(радость) = большая радость
```

Машинно-читаемый семантический словарь

Тузов попытался сократить описания Мельчука до машинно-читаемых формул. Для этого, во-первых, была расширена «аксиоматика». Скажем, слово «свинья» было сочтено атомарным — то есть можно считать, что «свинья» есть некий объект семантического поля языка, и «объяснять» его компьютеру, в принципе, незачем. Можно приписать атрибуты, если есть желание. Объяснение же неизбежно выльется в длинную абракадабру, освещающую самые разные свинские аспекты.

Во-вторых, был разработан механизм «семантических фунцкий», похожих на лексические функции Мельчука. Разница в том, что Мельчук всё-таки создавал словарь для людей — он объясняет слова языка. Тузов, по большому счёту, оперирует только смыслами, а какие конкретно слова за ними стоят — вопрос второй. Если какое-либо слово можно описать с помощью «атомов» и семантических функций, так оно и описывается. Например, имеются функции «становиться», «иметь свойство» и «усиливать»: IncepCopul(), Copul() и Magn(). Тогда с их помощью можно выразить слова *мулатка* , *краснеть* и *огромный* следующим образом:

```
мулатка Copul(мулат, женщина) // мулат, который является
женщиной
краснеть IncepCopul(x, красный) // "x становится красным"
огромный Copul(x, Magn(большой)) // x огромен, если он
имеет сильно выраженное свойство "большой"
```

Здесь я не хотел бы акцентировать внимание на особенностях данного конкретного формализма — подходы, идеи могут сильно отличаться. Должна быть ясна главная мысль: разработать систему «атомарных» понятий и «функций» над понятиями, с помощью которых можно описывать новые понятия. Тогда можно попытаться разработать формальный толковый словарь, математически описывающий слова (а не гоняющий по кругу, пытаясь определить одно через другое). Таким образом, мы уходим от слов и приходим к смыслам.

Если слово описано формулой, а предложение, соответственно, является композицией формул, то слова и предложения можно как-то изучать математически. Например, «раскрывать скобки», применять операции и так далее. Пищи для ума в этой идее предостаточно… Кстати, что-то подобное высказывается в работах Харриса об Operator Grammars, но честно признаюсь, его трудов я не осилил. Текст там куда ближе к традиционной лингвистике (много слов и мало математики :)).

Семантические формулы могут быть полезны хотя бы в машинном переводе. Даже если вы не можете перевести какой-нибудь кусок, можно просто «раскрыть» формулу, заменяя лексические функции чеканным математическим текстом. Допустим, вы знаете, как по-английски «мулат» и «женщина», но у вас нет английского слова «мулатка» в словаре. Зато есть русскоязычный толковый семантический словарь. Тогда, встретив «мулатку», на английский можно перевести соответствующую фразу из семантического словаря: «мулат, который является женщиной».

В идеале, целые словосочетания из разных языков должны отображаться на одни и те же семантические формулы. Учительница — это «учитель, который является женщиной». А по-английски одного слова «учительница» нет, напишут что-то вроде «female teacher», если надо подчеркнуть пол — но на выходе будет та же самая формула.

Пополняя базу данных переводов семантических формул, можно добиваться всё меньшей и меньшей корявости перевода. То есть автоматический переводчик начинает с «мулата, который является женщиной», и при полной базе данных уже понимает, что правильный перевод — это просто «мулатка». Так же, в принципе, как и мы с вами — не зная точного перевода, начинаем объяснять слово через более простые .

А иногда это будет (впрочем, не забывайте, я теоретизирую!) просто спасать положение, если требуемого слова в целевом языке попросту нет — тогда «раскрытие формулы» может дать понятие о происходящем. Пример не праздный. Скажем, в финском нет глагола «иметь», как ни странно это звучит. Можно сказать «у меня есть» («у тебя есть», «у него есть»), но абстрактно «иметь» нельзя. И перевести даже такую простую фразу как «хорошо иметь собаку!» оказывается сложнее, чем казалось бы. Есть, правда, глагол «владеть», но по-фински «хорошо владеть собакой!» звучит так же глупо, как и по-русски.

А теперь перейдём от семантики к политике :)

Куда податься?

Вот тут я не специалист, сам в процессе :) Но некоторые мысли хотелось бы озвучить. Вообще говоря, сейчас мы всё ещё переживаем последствия AI Winter. На ИИ в целом и на компьютерную лингвистику в частности возгалались слишком большие надежды поначалу. Много чего лопнуло, и амбициозные проекты вроде Толково-комбинаторного словаря сейчас не в моде. Пока умные люди размышляют, почему так произошло, распределители ресурсов сконцентрировались на чётких приложениях отдельных методик. Умеем выполнять морфологический анализ? Отлично! Умеем разделять текст на предложения? Прекрасно — применим! Так мало-помалу добились отдельных неплохих результатов. Но в целом определённая стагнация присутствует. Например, едва ли не самым популярным коммерческим машинным переводчиком остаётся SYSTRAN, а это технология 60-70-х годов!

Проблема компьютерной лингвистики в том, что проверка любой гипотезы требует массы усилий. Ну представьте себе, вот завтра я начну писать XDG-грамматику для русского. Может, к пенсии закончу. При этом (побрюзжу немного, ладно?) пропихнуть в научное сообщество маленькие прототипы тоже непросто. Например, пару раз мои статьи отклоняли с таким разъяснением: непонятно, насколько излагаемые идеи способны справиться со всем разнообразием сложностей естественного языка. Мысль вполне понятная, и даже правильная. Но, с другой стороны, как можно доказать способность «справиться со всем разнообразием сложностей», не написав грамматику/словарь/whatever, умеющую справляться со всем подряд? А это и есть полномасштабный проект.

Для себя я пока придумал такую штуку: *NLP в образовании* . Посмотрим, насколько она сработает. Суть в следующем. Если вы посмотрите на софт, обучающий физике или химии, там всё уже прекрасно: самые настоящие виртуальные лаборатории. От программ же, обучающих иностранному

языку, просто плакать хочется. Все, думаю, видели эти убожества: кошмарные клипартовские картинки, пара сотен озвученных слов, цветастые интерфейсики, сделанные на коленке… А на выходе просто гибрид книжки и магнитофона, то есть реально компьютер не стал чем-то более прогрессивным по сравнению с традиционным оборудованием.

Вот я думаю, чего не хватает образовательному софту (для изучения языка), чтобы превратиться в виртуальную лабораторию? А вот NLP и не хватает! Начнём с простого: тот же морфологический анализатор/синтезатор. Уверен, такая софтина пригодится каждому человеку, изучающему язык. А если пойти дальше? Например, сейчас обкатываю идею «лего-кубиков» для слов. Представьте себе карточки со словами на экране компьютера. Их можно склеивать в словосочетания, а словосочетания — в предложения. Но при этом не сочетающиеся между собой слова клеиться не будут. Например, «красный» и «корова» не склеятся, пока не согласуешь их в роде, числе и падеже. Можно придумать мощные средства проверки правописания (не как в Ворде, а именно для нужд и типичных ошибок новичков...)

А почему, собственно, образование? Да потому, что словарный запас новичка мал, и используемые им конструкции тоже ограничены. Стало быть, даже прототип на 500 слов может быть реально полезен, может кому-то пригодиться, привлечь внимание… А то выльется и во что-то более серьёзное. Не знаю пока :) Но примкнуть к существующему проекту пока не надумал. Что где хорошее есть? Насколько помню, в Чехии пытаются работать с XDK. Но это чешский язык — так что, начнём с изучения чешского, а там примкнём, если проект ещё будет жив на тот момент? ;) И так везде!

В общем, на этой мажорной ноте закончим сей длинный цикл. Дальше — скорее всего, по следам дискуссий и по заявкам читателей. Ну или когда появятся интересные новости в нашей замечательной области. Всем спасибо за внимание!