ПОЧЕМУ ОТРИЦАНИЕ НЕ ТРЕБУЕТ ОТДЕЛЬНОГО МЕХАНИЗМА

И что это говорит о формализме в обработке языка

varshchik
me@varshchik.dev · github.com/varshchik/nsai

РЕЗЮМЕ.

В большинстве систем символьного рассуждения отрицание реализовано как отдельный примитив: унарный оператор ¬, флаг полярности, специальное отношение. NSAI v2 — граф соседств слов с привязкой рёбер к контекстам предложений — даёт всё необходимое поведение отрицания без отдельного механизма, не как ленивый пропуск, а как следствие архитектурного выбора. В этой заметке разбирается, как именно это работает на примере с четырьмя предложениями про спящего кота, и формулируется методологический принцип, который объясняет ещё несколько похожих архитектурных решений: прежде чем кодировать наблюдаемое поведение как примитив, стоит проверить, не появляется ли оно побочно из более простой механики, уже имеющейся в системе.

1. ВВЕДЕНИЕ

Несколько месяцев назад я возился с обработкой отрицания в NSAI — небольшой исследовательской системе для работы с текстами, которую разрабатываю — и поймал себя на странной мысли. Естественный план был такой: завести оператор НЕ, добавить флаг полярности к фактам, написать ветку обработки для запросов с отрицанием. Так делают почти все системы такого типа: классическая логика построена вокруг унарного ¬ [1], expert systems в 80-90-е писали явный NOT [3], современные knowledge graphs ставят специальные отношения для отрицательных утверждений [4]. Я начал писать эту ветку и где-то посредине заметил, что не понимаю зачем именно я её пишу.

Вопрос звучал примерно так. Откуда берётся убеждённость, что отрицание нуждается в собственном механизме? Из языка — в нём есть «не». Из логики — в ней есть ¬. Но это связка вида «X есть в наблюдаемом феномене → X должен быть и в реализации». А такая связка вовсе не самоочевидна. Из того, что вода течёт, не следует, что в водопроводе должен быть модуль flow(). Из того, что в физике массы притягиваются, не следует, что в физическом симуляторе должна быть подпрограмма, реализующая притяжение как примитив. Поведение, наблюдаемое в системе, и механика, его порождающая, — это разные вещи, и спутывать их — типовая категориальная ошибка, на которую особенно легко попасться, когда ты строишь то, что должно демонстрировать определённое поведение.

Я выключил то, что писал, и попробовал думать с другой стороны. Что если отрицание — не примитив, а наблюдаемое поведение? Тогда вопрос становится не «как реализовать оператор НЕ», а «какие более простые механизмы могут дать поведение отрицания как побочный эффект». Это совсем другой класс задачи. И когда я начал её решать всерьёз, ответ оказался смущающее коротким: ничего специально реализовывать не нужно. То, как NSAI устроен на самом базовом уровне — граф соседств слов с прикреплёнными к рёбрам контекстами предложений — уже даёт всё необходимое поведение, если только не мешать ему специальными костылями. Эта заметка — про то, как и почему.

2. АРХИТЕКТУРА

Архитектура, о которой я говорю, чрезвычайно простая, и в её простоте — половина рассуждения. Текст поступает в систему по одному предложению за раз. Каждое предложение разбирается на токены, и для каждой пары непосредственных соседей записывается ребро в граф. Параллельно сохраняется само предложение под уникальным идентификатором — назову его ctx_id — и обратный индекс: для каждого ребра запоминается, в каких контекстах оно возникало. Никакой лемматизации, никаких POS-тегов, никаких склеек частиц с предикатами — слова идут в граф как они есть в тексте. Узел — это слово. Ребро — это «вот это слово стояло рядом с вот

этим». Контекст — это «вот в каком предложении я это видел». Всё.

На таком хранилище живут несколько операций, из которых для нашего разговора важна одна: классификация связи между двумя словами. Когда система находит путь между двумя узлами, она смотрит на множества ctx_id, в которых живут рёбра этого пути. Если у всех рёбер пути есть общий контекст — это УТВЕРЖДЕНИЕ: в корпусе действительно существует предложение, где эта связь высказана прямо. Если общего контекста нет — это РАССУЖДЕНИЕ: цепочка собрана из независимых предложений, и переход между ними проходит через узел, в котором контекст меняется. Никаких порогов, никаких эвристик, никаких меток: пересечение множеств идентификаторов и больше ничего.

3. ДЕМОНСТРАЦИЯ

Чтобы убедиться, что отрицание не требует отдельного обращения, я загрузил в чистую базу четыре простых факта про кота: «Кот спит на диване», «Кот спит днём», «Кот не спит ночью», «Кот не спит когда голоден». В графе появились узлы кот, спит, не, на, диване, днём, ночью, когда, голоден — каждое слово как самостоятельный узел, без всякой склейки не_спит и без всякой лемматизации. Ребро спит↔кот обзавелось контекстами {ctx0, ctx1} — предложениями, где утверждается, что кот спит. Ребро спит↔не обзавелось контекстами {ctx2, ctx3} — предложениями, где утверждается, что кот не спит. Узел спит ↔ кот в графе один; различие между «спать» и «не спать» живёт не в узлах, а в КОНТЕКСТНЫХ МНОЖЕСТВАХ РЁБЕР, которые из этого узла выходят. Теперь самое интересное. Я задал четыре запроса и снял их вывод. Первый запрос — «связан ли кот со спит».

```
кот ↔ спит: УТВЕРЖДЕНИЕ (2 пути)
  ✓ [утверждение, прочность=2] кот → спит
      ctx0: «Кот спит на диване»
      ctx1: «Кот спит днём»
  ✓ [утверждение, прочность=2] кот → не → спит
      ctx2: «Кот не спит ночью»
      ctx3: «Кот не спит когда голоден»
```

Система не выбирает между «кот спит» и «кот не спит». Она находит ОБА факта, классифицирует ОБА как утверждения, и для каждого показывает источники в корпусе. С точки зрения дискавери оба верны: корпус действительно содержит и одно, и другое; противоречия здесь нет, потому что условия разные (днём/на диване против ночью/когда голоден). Если бы у системы был оператор отрицания, ей пришлось бы выбирать; здесь выбора нет, потому что нет двоичной полярности — есть два разных пути от кот к спит, каждый со своим набором контекстов, и оба валидно показываются. Второй запрос — «связан ли кот с ночью».

```
кот ↔ ночью: УТВЕРЖДЕНИЕ (1 путь) + 1 рассуждение
  ~ [рассуждение, прочность=1] кот → спит → ночью
      смены через: спит
  ✓ [утверждение, прочность=1] кот → не → спит → ночью
      ctx2: «Кот не спит ночью»
```

Тут уже работает разница между утверждением и рассуждением. Путь кот →спит→ночью существует структурно (оба ребра есть в графе), но рёбра приходят из разных контекстов: кот↔спит живёт в {ctx0, ctx1} — там, где кот спит, — а спит↔ночью живёт в {ctx2} — там, где он не спит. Их пересечение пусто, и система помечает это как рассуждение через смену контекста на узле спит. То есть «кот спит ночью» — это НЕ УТВЕРЖДЕНИЕ КОРПУСА, а домысленная склейка, и система это видит. А путь кот→не→спит→ночью целиком живёт в ctx2, где так и сказано — «Кот не спит ночью», — и это утверждение. Различение работает без всякого знания о том, что не — это отрицание; работает потому, что три ребра пути физически делят общее предложение, а у альтернативного пути они общего предложения не делят.

Что меня здесь зацепило больше всего — это полное отсутствие в коде того, что можно было бы назвать «обработкой отрицания». В системе нет узла не_спит. Нет правила «если перед глаголом частица — склеивай». Нет ветки if is_negative_query. Нет поля polarity у фактов. Узел не в графе ничем не отличается от узла на или узла когда — просто слово, которое стояло рядом с другими словами в определённых предложениях. Различие между «кот спит» и «кот не спит» появляется только тогда, когда кто-то задаёт запрос; до запроса в графе никакого различия нет, есть только рёбра с прикреплёнными контекстами. Поведение, которое классическая система назвала бы «обработкой отрицания», здесь растворено в обычной механике классификации путей, и отдельного места для него в коде просто нет.

Любопытная деталь, которую стоит проговорить отдельно. На запрос не↔спит система отвечает утверждением, и это утверждение опирается на ctx2 и ctx3. Узел не живёт в графе самостоятельно, у него есть собственная контекстная окрестность — кот, спит. И эта окрестность совершенно симметрична окрестности любого другого «модификатора», который мог бы оказаться на том же месте в предложении. Если завтра в корпусе появится «Кот ВРЯД ЛИ спит ночью», то узлы вряд и ли будут обрабатываться ровно так же: соберут свои контексты, образуют соседства с кот и со спит, и запросы будут классифицироваться по тем же правилам. Никаких отдельных правил для «частиц», «модальных слов», «отрицания» — ничего, кроме рёбер и контекстов.

4. ЧЕТЫРЕ ФОРМЫ ОТРИЦАНИЯ

Лингвистическая типология отрицания [5] выделяет несколько форм, и стоит проговорить их явно. Отрицание в естественном языке — это не только частица «не» перед глаголом. Есть антонимы: «горячий» и «холодный», без всякой грамматической связки между ними, но функционально близкие к паре «спит» и «не спит». Есть имплицитное отрицание: «промолчал», «отказался», «проигнорировал» — положительные слова, в которых смысл отрицания зашит лексически, без всякого морфологического маркера. Есть модальные ослабления — «вряд ли», «едва ли», «навряд ли» — частичные отрицания, которые в классической бинарной логике вообще не помещаются. И есть префиксные отрицания: «невнимательный», «непонятный», «несогласие» — отрицание прячется внутри слова.

Система, в которой отрицание реализовано как специальный оператор НЕ, оказывается заложницей одной из этих форм — а именно той, под которую её настроили. Если оператор работает на частице, антонимы выпадают: для них нет грамматического маркера, на который оператор мог бы среагировать. Имплицитное отрицание тоже выпадает по той же причине. Модальные ослабления выпадают, потому что булев оператор не умеет в градацию между «истиной» и «ложью». А с префиксами начинаются вопросы вида «считать ли невнимательный отрицанием внимательный», и любой ответ создаёт несоответствие с поведением частицы — если да, то почему мы не делаем то же самое для всех слов с приставкой «не-» вне зависимости от их смысла; если нет, то почему частицы и приставки обрабатываются по-разному, если по смыслу они часто близки. Чтобы покрыть все четыре формы, приходится изобретать четыре механизма, каждый со своими правилами, и потом ещё пятый, чтобы их между собой согласовать.

Архитектура раздувается, и значительная её часть оказывается посвящена работе с разными типами отрицания, которые обещали быть одним явлением. Эта фрагментация подходов хорошо

задокументирована в обзорах computational linguistics [6].

В графе соседств с контекстными рёбрами все четыре формы получают одинаковое обращение. Каждое слово — узел. Каждое наблюдение в предложении — это набор рёбер с привязкой к этому предложению. Узлы спит и не соседствуют — потому что слова «спит» и «не» стояли рядом в тексте. Узлы холодный и тёплый будут соседствовать в одних контекстах, противопоставляться в других — это проявится через разные пересечения их контекстных множеств с другими узлами. Узел промолчал накопит свою окрестность, через которую система обнаружит, что он функционально похож на отказ, через цепочки к словам вроде «не сказал», «не ответил» — но это будут рассуждения с уже знакомой пометкой смены контекста, не утверждения. Узел невнимательный будет жить как самостоятельное слово, со своими соседями, и если корпус содержит достаточно фактов про невнимательность, его окрестность стабилизируется. Маршрут, которым каждое из этих слов попало в граф, разный — но на уровне обхода это неразличимо. Узел есть узел; обходится он одинаково; контекстная окрестность накапливается одинаково; различение «это сказано прямо» и «это домысленная склейка» делается одинаково. Одна механика, четыре формы покрыты, и не покрыта только пятая, шестая и седьмая, которые я ещё не встретил, потому что их я тоже не буду специально обрабатывать.

5. МЕТОДОЛОГИЧЕСКИЙ УРОК ШИРЕ ОТРИЦАНИЯ

Я говорю всё это про отрицание, но методологический урок шире. Он формулируется примерно так: прежде чем кодировать наблюдаемое поведение системы как её внутренний примитив, имеет смысл спросить, не может ли это поведение появиться как побочный эффект чего-то более простого, уже имеющегося в системе. Удивительно часто ответ — да. И тогда правильный шаг — не добавить механизм, а удалить идею, что механизм нужен.

В архитектуре NSAI этот же фильтр сработал в нескольких других местах, и каждый раз тоже привёл к удалению, а не к добавлению. ИДЕНТИЧНОСТЬ — нет отдельного механизма «это та же сущность»; узел в графе есть узел, и никакая операция идентификации сверху не нужна. КОНФЛИКТ МЕЖДУ ФАКТАМИ — нет операции «найти противоречие» внутри системы; конфликт возникает как наблюдение снаружи, через ту же классификацию путей, которая обрабатывает любой запрос. СИЛЛОГИЗМ — нет отдельного модуля modus ponens; цепочки вывода эмерджентно собираются тем же обходом графа, которым обрабатывается любой запрос, и дедукция растворена в обходе. РАЗЛИЧЕНИЕ ПРЯМОГО УТВЕРЖДЕНИЯ И КОСВЕННОГО ВЫВОДА — нет специального компонента-валидатора; разница между «так и сказано в предложении N» и «склеено из независимых источников» вытекает из того, делят ли рёбра пути общий ctx_id.

Каждый из этих случаев устроен по одному шаблону. Классическая символьная архитектура постулирует для соответствующего поведения отдельный компонент. NSAI обходится без него, и поведение всё равно наблюдается — потому что нужная асимметрия уже есть в данных, в графовых распределениях, в контекстных множествах рёбер. Архитектура от этого не становится проще случайно; она проектировалась с этим принципом с самого начала, и каждый раз, когда я ловил себя на желании добавить ещё один модуль, я заставлял себя сначала проверить, не делает ли уже имеющаяся механика того, что мне нужно. В половине случаев — делала.

Среди всех этих случаев отрицание — самый наглядный. Соблазн «реализовать» здесь особенно велик, потому что тысячи лет логической традиции говорят, что отрицание — фундаментальная операция, и сложно даже представить себе систему рассуждений без него. Если оказалось, что отдельный механизм не нужен даже здесь — разумно проверить каждое следующее «обязательное» допущение системы, которую разрабатываешь. Скорее всего, половина из них тоже не нужна. Это, может быть, наиболее ценный архитектурный навык, который у меня выработался за время работы над NSAI: подозревать любой компонент, который оправдывается тем, что «в литературе так принято».

6. ШИРЕ NSAI: ФОРМАЛИЗМ В ОБРАБОТКЕ ЯЗЫКА

Я говорил пока про отрицание и про несколько внутренних случаев в NSAI. Но тот же фильтр работает на ступень выше — на уровне самого подхода к обработке языка. Этот сдвиг я могу показать наглядно, потому что у проекта есть две версии: v1 и v2 лежат рядом в репозитории. v1 (она же NSAI v0.1) использовала весь стандартный формальный аппарат, который принято считать необходимым для работы с русским текстом: POS-тагирование через natasha [14], контекстную лемматизацию через pymorphy3 [13], дерево синтаксических зависимостей, типизированные факты вида (предикат, [аргументы]) с тремя классами — relation, entity, observation, маппинг UD-POS [15] → pymorphy POS для дизамбигуации. Каждый компонент казался необходимым: морфология — чтобы привести разные формы словак одной лемме, синтаксис — чтобы вытянуть структурные отношения, типы фактов — чтобы знать, как с фактом обращаться дальше.

Стандартная картина: формальный pipeline, через который проходит текст, и каждая стадия добавляет к данным свой слой структуры. При переходе к v2 каждая из этих стадий прошла тот же тест, который я описывал применительно к отрицанию: даёт ли этот формальный компонент поведение, которое без него не возникает — или поведение возникает и так, а компонент просто навязан литературной традицией работы с языком?

Ответ оказался везде один. POS-тагирование не нужно: если два слова стоят рядом в предложении, их совместность фиксируется в графе независимо от того, какие у них части речи. Лемматизация не нужна: разные формы одного слова — это разные узлы графа, которые накапливают свои собственные контекстные окрестности, и факт что они «вариант одного слова» либо проявится как сходство их соседств, либо не проявится, — и в обоих случаях это НАБЛЮДЕНИЕ ИЗ ДАННЫХ, не аксиома сверху. Синтаксический парсер не нужен: для классификации связи между двумя словами достаточно знать, лежат ли рёбра пути в общем контексте, без всякого знания о ролях. Типы фактов не нужны: ребро есть ребро, контекст есть контекст, дальнейшее различение делается через ту же пересеченную-непересеченную проверку, что обрабатывает любой запрос. Конкретно это выглядит так. v1 — около полутора тысяч строк с тяжёлыми зависимостями (natasha, pymorphy3, sqlite-индексы под формат предикат-аргумент). v2 — около ста пятидесяти строк, чистый Python без внешних зависимостей. Эффект, который мне кажется наиболее показательным, не в количестве строк, а в области применимости: v1 в принципе не могла работать с не-русским текстом, потому что весь её аппарат был привязан к natasha и pymorphy3, специфичным для русского. v2 работает с любым текстом на любом языке, потому что узлами в ней становятся последовательности символов, разделённые пробелами, без всяких предположений об их грамматической природе. Это не оптимизация. Это смена того, что система ОБЕЩАЕТ делать, — и эта смена стала возможной потому, что v1 обещала слишком много.

Это тот же методологический ход, что описан в случае отрицания, но применённый на масштабе целого pipeline. Каждый отдельный компонент — POS, морфология, синтаксис, типизация — заслуживает собственного разбора, и они будут разворачиваться в следующих заметках. Здесь важно зафиксировать только то, что отрицание — не изолированный случай. Это НАИБОЛЕЕ НАГЛЯДНЫЙ пример общего паттерна, который, когда я начал его последовательно применять, привёл к архитектуре, отказавшейся от значительной части того, что в литературе по обработке языка считается обязательным.

Стоит зафиксировать ещё одну деталь, чтобы было видно, что именно убрано в v2. Лемматизация, POS-тагирование, синтаксический парсинг — в v1 это были ПРЕПРОЦЕССОРЫ: текст проходил через них и попадал в рассуждение уже преобразованным. Рассуждающая часть v1 никогда не видела исходного текста — только результат пре-обработки. То есть формальный аппарат языка в v1 никогда не был частью рассуждения; он был ПЕРЕВОДНОЙ ПРОСЛОЙКОЙ между текстом и рассуждением, обещавшей упростить вход и вместо этого зашивавшей в него предположения о конкретном языке. Убрать эту прослойку — значит не убрать когнитивный модуль, а позволить рассуждению работать с языком как с ДАННЫМИ, а не с предобработанным набором допущений о структуре. В этом и состоит цель NSAI как программы: на каждом следующем шаге — будь то POS, синтаксис, типизация

фактов или что-то ещё — спрашивать, не является ли это очередной переводной прослойкой, которую можно убрать без потери поведения; и каждый утвердительный ответ обещает не только упрощение архитектуры, но и расширение её применимости.

Методологически этот подход не нов. Brooks [12] в робототехнике начала 1990-х аргументировал отказ от внутренних моделей мира в пользу behavior-based архитектур; the world is its own best model — формулировка, методологически близкая тому, что v2 делает для языка. Технически v2 наследует дистрибутивной гипотезе Firth [7] и Harris [8]: знание о слове извлекается из его соседств в корпусе, без формального описания через грамматику. Word2vec и его потомки [9] реализуют эту идею в векторном пространстве; v2 делает это в графовом, с тем отличием что граф сохраняет привязку к источникам, а векторное представление эту связь теряет. На уровне когнитивной парадигмы Newell & Simon [2] сформулировали physical symbol system hypothesis — позицию, которой v2 имплицитно оппонирует, не отрицая её, а показывая что один из её типичных компонентов (явные операторы) можно убрать без потери поведения.

Один момент стоит проговорить отдельно. Bender & Koller [10], восходящие к классической постановке symbol grounding problem [11], аргументируют что система обученная только на форме не может выучить meaning в строгом смысле. NSAI v2 работает именно с формой и не претендует на семантику в их строгом смысле — она решает другую задачу: дискавери связей в корпусе с трассируемостью к источникам, где утверждение и рассуждение явно различены. Это не возражение их позиции, а уточнение скоупа: v2 не пытается выйти за пределы формы, потому и не сталкивается с проблемой grounding'a.

7. ЗАКЛЮЧЕНИЕ

Код, тесты и подробности архитектуры лежат на github.com/varshchik/nsai. Там же — principles_ru.md, короткий документ, в котором собраны принципы устройства системы, включая упомянутые здесь случаи. Это компактнее, чем код, и читается независимо. Следующие заметки в этой серии будут разворачивать остальные случаи отдельно. На уровне внутренней архитектуры NSAI — конфликт без модуля противоречий, силлогизм как обход графа, различение утверждения и рассуждения через контексты рёбер. На уровне формального аппарата обработки языка — POS-тагирование, лемматизация, синтаксический парсер, типизация фактов. Сроков обещать не буду, пишу по мере того, как соответствующие части системы дозревают до состояния, в котором их имеет смысл описывать публично. Если в этой заметке есть места, где аргумент кажется хрупким или не сходящимся — буду рад, если напишете на me@varshchik.dev, охотно отвечу. NSAI — сольный исследовательский проект, и внешний взгляд для меня особенно ценен.

ССЫЛКИ

[1] McCarthy, J. (1959). Programs with Common Sense. In Mechanisation of Thought Processes, Vol. 1. London: HMSO.

[2] Newell, A. & Simon, H. A. (1976). Computer Science as Empirical Inquiry: Symbols and Search. Communications of the ACM 19(3), 113–126.

[3] Buchanan, B. G. & Shortliffe, E. H. (1984). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley.

[4] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., et al. (2021). Knowledge Graphs. ACM Computing Surveys 54(4), Article 71.

[5] Horn, L. R. (1989). A Natural History of Negation. University of Chicago Press. (Reprinted 2001, CSLI Publications.)

[6] Morante, R. & Sporleder, C. (2012). Modality and Negation: An Introduction to the Special Issue. Computational Linguistics 38(2), 223–260.

[7] Firth, J. R. (1957). A synopsis of linguistic theory, 1930–55. In Studies in Linguistic Analysis. Oxford: Blackwell, 1–32.

[8] Harris, Z. S. (1954). Distributional Structure. Word 10(2–3), 146–162.

[9] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems 26 (NIPS 2013).

[10] Bender, E. M. & Koller, A. (2020). Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020), 5185–5198.

[11] Harnad, S. (1990). The Symbol Grounding Problem. Physica D: Nonlinear Phenomena 42(1–3), 335–346.

[12] Brooks, R. A. (1991). Intelligence without representation. Artificial Intelligence 47(1–3), 139–159.

[13] Korobov, M. (2015). Morphological Analyzer and Generator for Russian and Ukrainian Languages. In Analysis of Images, Social Networks and Texts (AIST 2015), Communications in Computer and Information Science, vol. 542. Springer, 320–332.

[14] natasha — Solutions for basic NLP tasks for Russian. Software library. https://github.com/natasha/natasha

[15] de Marneffe, M.-C., Manning, C. D., Nivre, J. & Zeman, D. (2021). Universal Dependencies. Computational Linguistics 47(2), 255–308.