WHY NEGATION REQUIRES NO SEPARATE MECHANISM

And what this says about formalism in language processing

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

ABSTRACT.

In most symbolic reasoning systems, negation is implemented as a separate primitive: a unary ¬ operator, a polarity flag, a special relation type. NSAI v2 — an adjacency graph of words with edges bound to sentence contexts — produces all the necessary behavior of negation without a separate mechanism, not as a lazy omission but as a consequence of an architectural choice. This note breaks down how it works on a four-sentence example about a sleeping cat, and formulates a methodological principle that explains several similar architectural decisions: before encoding observed behavior as a primitive, it's worth checking whether the behavior emerges as a side effect of simpler machinery already in the system.

1. INTRODUCTION

A few months ago I was tinkering with negation handling in NSAI — a small research system for working with text that I'm developing — and caught myself thinking something odd. The natural plan was to introduce a NOT operator, add a polarity flag to facts, write a branch for handling queries with negation. That's how almost all systems of this type do it: classical logic is built around the unary ¬ [1], expert systems in the 80s and 90s wrote explicit NOT [3], modern knowledge graphs add special relations for negative statements [4]. I started writing that branch and somewhere midway realized I didn't understand exactly why I was writing it.

The question went roughly like this. Where does the conviction come from that negation requires its own mechanism? From language — it has "not". From logic — it has ¬. But this is the kind of link "X exists in the observed phenomenon → X must exist in the implementation". And that link is not self-evident at all. From the fact that water flows, it doesn't follow that the plumbing must have a flow() module. From the fact that masses attract in physics, it doesn't follow that a physical simulator must have a subroutine implementing attraction as a primitive. The behavior observed in a system and the mechanics that produce it are different things, and conflating them is a typical category error, especially easy to fall into when you're building something that's supposed to demonstrate a particular behavior.

I set aside what I was writing and tried thinking from the other direction. What if negation isn't a primitive but an observed behavior? Then the question becomes not "how to implement the NOT operator", but "what simpler mechanisms could produce the behavior of negation as a side effect". This is a different class of problem entirely. And when I started solving it in earnest, the answer was embarrassingly short: nothing needs to be implemented specially. The way NSAI is built at the most basic level — an adjacency graph of words with sentence contexts attached to edges — already gives all the necessary behavior, as long as you don't interfere with special workarounds. This note is about how and why.

2. ARCHITECTURE

The architecture I'm talking about is extremely simple, and in its simplicity lies half the argument. Text enters the system one sentence at a time. Each sentence is split into tokens, and for every pair of immediate neighbors an edge is recorded in the graph. In parallel, the sentence itself is stored under a unique identifier — I'll call it ctx_id — and a reverse index: for each edge, we remember which contexts it appeared in. No lemmatization, no POS tags, no joining of particles with predicates — words go into the graph as they appear in the text. A node is a word. An edge is "this word stood next to this one". A context is "this is the sentence I saw it in". That's it.

A few operations live on this storage, but for our discussion one matters: classifying the relation between two words. When the system finds a path between two nodes, it looks at the sets of ctx_id in which the edges of that path live. If all edges of the path share a common context — this is an ASSERTION: there actually exists a sentence in the corpus where this connection is stated directly. If there's no common context — this is REASONING: the chain is assembled from independent sentences, and the transition between them passes through a node where the context changes. No thresholds, no heuristics, no labels: intersection of identifier sets and nothing more.

3. DEMONSTRATION

To check that negation requires no special handling, I loaded into a clean database four simple facts about a cat: "The cat is sleeping on the sofa", "The cat is sleeping during the day", "The cat is not sleeping at night", "The cat is not sleeping when hungry". The graph picked up nodes the, cat, is, sleeping, not, on, sofa, during, day, at, night, when, hungry — each word as a standalone node, without any joining of "not_sleeping" into a compound and without any lemmatization. The edge is↔sleeping acquired contexts {ctx0, ctx1} — sentences where the cat is sleeping. The edge is↔not acquired contexts {ctx2, ctx3} — sentences where the cat is not sleeping. The node "sleeping" in the graph is a single node; the distinction between "sleeping" and "not sleeping" doesn't live in nodes, but in the CONTEXT SETS OF EDGES around this node. Now the interesting part. I issued four queries and recorded the output. The first query — "is cat connected to sleeping".

```
cat ↔ sleeping: ASSERTION (2 paths)
  ✓ [assertion, strength=2] cat → is → sleeping
      ctx0: “The cat is sleeping on the sofa”
      ctx1: “The cat is sleeping during the day”
  ✓ [assertion, strength=2] cat → is → not → sleeping
      ctx2: “The cat is not sleeping at night”
      ctx3: “The cat is not sleeping when hungry”
```

The system doesn't choose between "the cat is sleeping" and "the cat is not sleeping". It finds BOTH facts, classifies BOTH as assertions, and for each one shows the sources in the corpus. From the standpoint of discovery, both are valid: the corpus indeed contains both; there's no contradiction here because the conditions differ (during the day / on the sofa vs at night / when hungry). If the system had a negation operator, it would have to choose; here there's no choice, because there's no binary polarity — there are two different paths from cat to sleeping, each with its own context set, and both are validly shown. The second query — "is cat connected to night".

```
cat ↔ night: ASSERTION (1 path) + 1 reasoning
  ~ [reasoning, strength=1] cat → is → sleeping → at → night
      switches at: sleeping
  ✓ [assertion, strength=1] cat → is → not → sleeping → at → night
      ctx2: “The cat is not sleeping at night”
```

Now the distinction between assertion and reasoning kicks in. The path cat→is→sleeping→at→night exists structurally (all edges are in the graph), but the edges come from different contexts: is↔sleeping lives in {ctx0, ctx1} — where the cat is sleeping — and sleeping↔at lives in {ctx2} — where it's not sleeping. Their intersection is empty, and the system marks this as reasoning through a context switch at the node "sleeping". In other words, "the cat is sleeping at night" is NOT AN ASSERTION OF THE CORPUS, but an inferred stitching, and the system sees this. Meanwhile the path cat→is→not→sleeping→at→night lives entirely within ctx2, where it's said directly — "The cat is not sleeping at night" — and this is an assertion. The distinction works without any knowledge that "not" means negation; it works because the edges of the path physically share a common sentence, while the alternative path's edges don't share one.

What hooked me here most of all is the complete absence in the code of anything that could be called "negation handling". There's no "not_sleeping" node. No rule "if a particle stands before a verb, glue them". No "if is_negative_query" branch. No polarity field on facts. The "not" node in the graph is no different from "on" or "when" — just a word that stood next to other words in particular sentences. The distinction between "the cat is sleeping" and "the cat is not sleeping" appears only when someone issues a query; before the query there's no distinction in the graph, just edges with attached contexts. The behavior that a classical system would call "negation handling" is here dissolved into the ordinary mechanics of path classification, and there simply isn't a separate place in the code for it.

A curious detail worth calling out separately. To the query not↔sleeping the system answers with an assertion, and that assertion rests on ctx2 and ctx3. The node "not" lives in the graph on its own, with its own contextual neighborhood — cat, is, sleeping. And this neighborhood is perfectly symmetric to that of any other "modifier" that could end up in the same position in a sentence. If tomorrow the corpus gains "The cat HARDLY sleeps at night", the nodes "hardly" and any others will be handled exactly the same way: they'll collect their contexts, form neighborhoods with cat and sleeping, and queries will be classified by the same rules. No special rules for "particles", "modal words", "negation" — nothing but edges and contexts.

4. FOUR FORMS OF NEGATION

The linguistic typology of negation [5] distinguishes several forms, and they're worth stating explicitly. Negation in natural language isn't only the particle "not" before a verb. There are antonyms: "hot" and "cold", with no grammatical link between them, but functionally close to the "sleeping" / "not sleeping" pair. There's implicit negation: "stayed silent", "refused", "ignored" — positive words in which the meaning of negation is lexically baked in, without any morphological marker. There are modal hedges — "hardly", "barely", "scarcely" — partial negations that don't fit in classical binary logic at all. And there are prefixed negations: "inattentive", "unclear", "disagreement" — negation hiding inside the word.

A system in which negation is implemented as a special NOT operator turns out to be hostage to one of these forms — the one it was set up to handle. If the operator works on the particle, antonyms fall through: they have no grammatical marker the operator could react to. Implicit negation falls through for the same reason. Modal hedges fall through because a boolean operator can't handle gradation between "truth" and "falsity". And with prefixes you get questions like "should 'inattentive' count as negation of 'attentive'", and any answer creates an inconsistency with how the particle is handled — if yes, then why don't we do the same for all words with the prefix "in-" regardless of meaning; if no, then why are particles and prefixes treated differently when they're often semantically close. To cover all four forms, you have to invent four mechanisms, each with its own rules, and then a fifth to reconcile them. The architecture balloons, and a significant portion of it ends up dedicated to handling different types of negation that were supposed to be one phenomenon. This fragmentation of approaches is well-documented in computational linguistics surveys [6].

In an adjacency graph with context-bound edges, all four forms receive the same treatment. Each word — a node. Each observation in a

sentence — a set of edges bound to that sentence. Nodes "sleeping" and "not" are neighbors — because the words "sleeping" and "not" stood next to each other in the text. Nodes "cold" and "warm" will be neighbors in some contexts, contrasted in others — this will manifest through different intersections of their context sets with other nodes. The node "refused" will accumulate its neighborhood, through which the system will discover that it's functionally similar to "decline", via chains to words like "didn't say", "didn't answer" — but these will be reasoning paths with the familiar context-switch marking, not assertions. The node "inattentive" will live as a standalone word, with its own neighbors, and if the corpus contains enough facts about inattentiveness, its neighborhood stabilizes. The route by which each of these words got into the graph is different — but at the level of traversal it's indistinguishable. A node is a node; it's traversed the same way; its contextual neighborhood accumulates the same way; the distinction between "this is stated directly" and "this is an inferred stitching" is made the same way. One mechanism, four forms covered, and the only forms not covered are the fifth, sixth, and seventh, which I haven't met yet, because I'm not going to specially handle them either.

5. THE METHODOLOGICAL LESSON BEYOND NEGATION

I'm saying all this about negation, but the methodological lesson is wider. It can be stated roughly like this: before coding observed behavior of a system as its internal primitive, it's worth asking whether this behavior could appear as a side effect of something simpler already in the system. Surprisingly often the answer is yes. And then the right move is not to add a mechanism, but to drop the idea that one is needed.

In the architecture of NSAI this same filter fired in several other places, and each time it also led to removal rather than addition. IDENTITY — there's no separate "this is the same entity" mechanism; a node in the graph is a node, and no identification operation above it is needed. CONFLICT BETWEEN FACTS — there's no "find a contradiction" operation inside the system; conflict arises as an observation from outside, via the same path classification that handles any query. SYLLOGISM — there's no separate modus ponens module; inference chains emerge through the same graph traversal that handles any query, and deduction dissolves into the traversal. DISTINGUISHING DIRECT ASSERTION FROM INDIRECT INFERENCE — there's no special validator component; the difference between "this is literally said in sentence N" and "this is stitched from independent sources" follows from whether the edges of the path share a common ctx_id.

Each of these cases follows the same template. A classical symbolic architecture posits a separate component for the corresponding behavior. NSAI gets by without it, and the behavior still appears — because the needed asymmetry is already in the data, in the graph distributions, in the context sets of edges. The architecture doesn't get simpler by accident; it was designed with this principle from the start, and every time I caught myself wanting to add another module, I forced myself to first check whether the existing machinery wasn't already doing what I needed. In half the cases, it was.

Among all these cases, negation is the most striking. The temptation to "implement" is especially great here, because thousands of years of logical tradition say that negation is a fundamental operation, and it's hard even to imagine a reasoning system without it. If it turns out that a separate mechanism isn't needed even here — it's reasonable to check every next "mandatory" assumption of the system you're building. Probably half of them aren't needed either. This is perhaps the most valuable architectural skill I developed over time working on NSAI: to suspect any component that justifies itself with "this is how it's done in the literature".

6. BEYOND NSAI: FORMALISM IN LANGUAGE PROCESSING

So far I've been talking about negation and a few internal cases in NSAI. But the same filter works one level up — at the level of the approach to language processing itself. I can show this shift clearly because the project has two versions: v1 and v2 sit side by side in the repository. v1 (also known as NSAI v0.1) used the entire standard formal apparatus considered necessary for working with Russian text: POS-tagging via natasha [14],

context-sensitive lemmatization via pymorphy3 [13], a tree of syntactic dependencies, typed facts of the form (predicate, [arguments]) with three classes — relation, entity, observation, a UD-POS [15] → pymorphy POS mapping for disambiguation. Each component seemed necessary: morphology — to reduce different forms of a word to a single lemma, syntax — to extract structural relations, fact types — to know how to handle a fact downstream. The standard picture: a formal pipeline through which text passes, with each stage adding its own layer of structure to the data.

In transitioning to v2, each of these stages went through the same test I described for negation: does this formal component give behavior that wouldn't otherwise exist — or does the behavior emerge anyway, with the component simply imposed by literary tradition of working with language?

The answer was the same everywhere. POS-tagging isn't needed: if two words stand next to each other in a sentence, their co-occurrence is recorded in the graph regardless of their parts of speech. Lemmatization isn't needed: different forms of a word are different graph nodes that accumulate their own contextual neighborhoods, and the fact that they're "variants of one word" either manifests as similarity of their neighborhoods, or doesn't — in either case this is an OBSERVATION FROM THE DATA, not an axiom from above. A syntactic parser isn't needed: to classify the relation between two words, it's enough to know whether the edges of the path lie in a common context, with no knowledge of roles. Fact types aren't needed: an edge is an edge, a context is a context, further differentiation is done via the same intersected/non-intersected check that handles any query.

Concretely it looks like this. v1 — about fifteen hundred lines with heavy dependencies (natasha, pymorphy3, sqlite indexes under predicate-argument format). v2 — about a hundred and fifty lines, pure Python with no external dependencies. The effect I find most telling is not the line count but the scope: v1 in principle could not work with non-Russian text, because its entire apparatus was tied to natasha and pymorphy3, specific to Russian. v2 works with any text in any language, because its nodes become sequences of characters separated by spaces, with no assumptions about their grammatical nature. This isn't an optimization. It's a change in what the system PROMISES to do — and this change became possible precisely because v1 was promising too much.

It's the same methodological move described in the case of negation, but applied at the scale of an entire pipeline. Each component — POS, morphology, syntax, typing — deserves its own analysis, and they'll be unfolded in subsequent notes. What matters here is to fix the fact that negation isn't an isolated case. It's the MOST VIVID example of a general pattern that, when I started applying it consistently, led to an architecture that abandoned a significant part of what the literature on language processing considers mandatory.

One more detail worth fixing, to make clear exactly what's been removed in v2. Lemmatization, POS-tagging, syntactic parsing — in v1 these were PREPROCESSORS: text passed through them and reached reasoning already transformed. The reasoning part of v1 never saw the original text — only the result of preprocessing. In other words, the formal apparatus of language in v1 was never part of reasoning; it was a TRANSLATION LAYER between text and reasoning, promising to simplify the input and instead baking in a set of assumptions about a specific language. Removing this layer means not removing a cognitive module, but letting reasoning work with language as DATA, not as a preprocessed set of assumptions about structure. This is what NSAI as a program is about: at each next step — whether POS, syntax, fact typing, or something else — to ask whether this is yet another translation layer that can be removed without loss of behavior; and each affirmative answer promises not only architectural simplification but also expansion of applicability.

Methodologically, this approach isn't new. Brooks [12] in robotics in the early 1990s argued for abandoning internal world models in favor of behavior-based architectures; "the world is its own best model" — a formulation methodologically close to what v2 does for language. Technically v2 inherits from the distributional hypothesis of Firth [7] and Harris [8]: knowledge about a word is extracted from its neighbors in the corpus, without formal description through grammar. Word2vec and its descendants [9] realize this idea in vector space; v2 does it in graph space, with the

difference that the graph preserves attribution to sources, while a vector representation loses that connection. At the level of the cognitive paradigm, Newell & Simon [2] formulated the physical symbol system hypothesis — a position that v2 implicitly opposes, not by denying it, but by showing that one of its typical components (explicit operators) can be removed without loss of behavior.

One point worth addressing separately. Bender & Koller [10], going back to the classical formulation of the symbol grounding problem [11], argue that a system trained only on form cannot learn meaning in the strict sense. NSAI v2 works precisely with form and doesn't claim semantics in their strict sense — it solves a different task: discovery of relations in a corpus with traceability to sources, where assertion and reasoning are explicitly distinguished. This isn't an objection to their position, but a clarification of scope: v2 doesn't try to step outside form, and therefore doesn't run into the grounding problem.

7. CONCLUSION

Code, tests, and architecture details live at github.com/varshchik/nsai. There's also principles_en.md (and a Russian version), a short document collecting the principles of the system's design, including the cases discussed here. It's more compact than the code and reads independently.

Subsequent notes in this series will unfold the remaining cases separately. At the level of NSAI's internal architecture — conflict without a contradiction module, syllogism as graph traversal, distinguishing assertion from reasoning through edge contexts. At the level of the formal apparatus of language processing — POS-tagging, lemmatization, syntactic parser, fact typing. I won't promise timelines; I write as the relevant parts of the system ripen to the point where it makes sense to describe them publicly.

If anywhere in this note the argument seems fragile or doesn't quite hold together — I'd be glad to hear from you at me@varshchik.dev, and I'll respond gladly. NSAI is a solo research project, and an outside perspective is especially valuable to me.

REFERENCES

[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.