Define Syntax-Locked and Syntax-Free Artifacts: Clearly articulate what constitutes a syntax-locked artifact versus a syntax-free artifact, providing comprehensive definitions and characteristics for each.
Compare and Contrast Artifacts: Systematically compare syntax-locked and syntax-free artifacts across various dimensions, such as complexity, flexibility, performance, ease of use, and applicability in different contexts.
Develop Evaluation Criteria: Establish rigorous, objective criteria for evaluating and measuring the effectiveness, efficiency, and suitability of syntax-locked and syntax-free artifacts in various scenarios.
Empirical Analysis: Conduct empirical research to gather data and insights on the real-world application and performance of both types of artifacts. This involves designing experiments, collecting data, and performing statistical analyses to validate findings.
Impact Assessment: Assess the impact of syntax-locked and syntax-free artifacts on different stakeholders, including developers, end-users, and organizations. Consider factors like learning curve, adaptability, and long-term sustainability.
Provide Guidelines and Recommendations: Based on the findings, offer practical guidelines and recommendations for choosing between syntax-locked and syntax-free artifacts. Address potential trade-offs and suggest best practices for different use cases.
Identify Future Research Directions: Highlight gaps in the current understanding and propose areas for future research to further explore the nuances and implications of syntax-locked and syntax-free artifacts.

```
// Airtable entity
Table Airtable {
  AirtableId int [pk, unique] // Primary Key
  Ideas Idea[] // Relation to Ideas table
  IdeaTransformers IdeaTransformer[] // Relation to IdeaTransformers table
  Generations Generation[] // Relation to Generations table
  GenerationTransformers GenerationTransformer[] // Relation to GenerationTransformers table
  Entities Entity[] // Relation to Entities table

  Note: "Airtable entity containing metadata and relationships to ideas and transformations."
}

// Ideas entity
Table Idea {
  IdeaId int [pk, unique] // Primary Key
  SourceIdea string // Source idea text
  Name string // Idea name
  IsActiveIdea string // Status of the idea
  Generations string[] // List of generations associated with the idea
  GenerationTransformers string[] // List of generation transformers associated with the idea
  GenerationTransformerFullPrompts string[] // Full prompts used for generation transformers
  GenerationTransformerNames string[] // Names of generation transformers
  AirtableId int [ref: > Airtable.AirtableId] // Foreign Key to Airtable

  Note: "Idea entity representing individual ideas, their statuses, and relationships."
}

// IdeaTransformers entity
Table IdeaTransformer {
```

```
  IdeaTransformerId int [pk, unique] // Primary Key
  Idea string // Associated idea
  IsActiveIdea string // Status of the idea
  IdeasSourceIdea string // Source idea text
  IdeaName string // Name of the idea
  AirtableId int [ref: > Airtable.AirtableId] // Foreign Key to Airtable
  TransformedArtifacts string // Transformed artifacts details
  FullPrompt string // Full prompt used for transformation
  Name string // Transformer name
  fields string // Additional fields

  Note: "Transformer entity for ideas, storing transformation details and prompts."
}

// Generations entity
Table Generation {
  GenerationId int [pk, unique] // Primary Key
  GeneratioNumber int // Generation number
  Idea string // Associated idea
  Name string // Generation name
  IsActiveIdea string // Status of the idea
  IdeaName string // Name of the idea
  IdeaSourceIdea string // Source idea text
  AirtableId int [ref: > Airtable.AirtableId] // Foreign Key to Airtable
  TransformedArtifacts string[] // List of transformed artifacts

  Note: "Entity representing generations of ideas, tracking status and transformations."
}

// GenerationTransformers entity
Table GenerationTransformer {
  GenerationTransformerId int [pk, unique] // Primary Key
  Transformer string // Transformer details
  Generation string // Associated generation
  TransformedArtifacts string[] // List of transformed artifacts
  GenerationIdea string // Idea associated with the generation
  GenerationName string // Name of the generation
  Name string // Transformer name
  IsActiveIdea string // Status of the idea
  GeneratioNumber int // Generation number
  GenerationIdeaName string // Name of the idea associated with the generation
  GenerationSourceIdea string // Source idea text
  TransformerFullPrompt string // Full prompt used for transformation
  Prompt string // Transformer prompt
  AirtableId int [ref: > Airtable.AirtableId] // Foreign Key to Airtable

  Note: "Entity representing transformers applied to generations, tracking transformation details and prompts."
}
```

```
// Entities entity
Table Entity {
   EntityId int [pk, unique] // Primary Key
   Name string // Entity name
   PluralName string // Plural form of entity name
   AirtableName string // Associated Airtable name
   AirtableId int [ref: > Airtable.AirtableId] // Foreign Key to Airtable

   Note: "General entity storing metadata for various entities within Airtable."
}
```

This GPT is to help me write this paper.

Research Paper

So - I want to write a paper. TEll me about the notion of syntax-locked vs syntax-free models. I've been talking to my Professor of MDE and the most recent language that he has started using is:
Original Position: Everything is is a language, one person's WHAT is another person's HOW, and so there is no bright red line between them, it is pink, and blurry, at best.

1) I accept that an ACID compliant DB is not "a language" (i.e. not EVERYTHING) is a language
 - there is the caveat that while he accepts that they are different, he insists that this difference is a "red herring"
2) Syntax-locked (i.e. 1d text files that require parsing, lexting, tokenizing, interpretation,etc before they can be understood) is a concept that _I think_ he is basically on board with.
3) The idea that there is a bright red line is still pink because he said on our last call, repeatedly, that he still doesn't really know/understand what I mean when I say syntax free.

So - tell me about the difference(s) between syntax-locked vs syntax-free content.
So - this is actually a statement about an aspect/element of language that I feel like is largely A) not yet understood, B) as a result, mostly unexplored scientifically.

So - here are the buckets that I would like to isolate, scientifically.

I would like the paper to be based on potentially one, but alternatively, a set of specific predefined prompts - for this discussion let's assume there is always 1 ROOT prompt that then has a cascading series of analytics performed on it. An example of such a prompt is:

Need: To do list app, with a few features, with some examples of categories and to do items.

GPT 4 would be the "objective" test tool. It's obviously biased itself, but I think that for the kinds of changes that we want to study, these biases should not play a part.

I want to develop a series of automated test protocols that can, perform tests like this:

1) ask gpt4 to answer this prompt 1000x:

Write a 1 paragraph description of all requirement details, features, categories/to do items. Be as complete and detailed as possible.

2) ask gpt4 to answer this prompt 1000x:
Write a single-source-of-truth.json with a list of all requirement details, features, categories/to do items. Be as complete and detailed as possible.

This would be the raw data that we would further analyze, also using basic data science along with continuing to leverage GPT 4.

Here are some examples of the kinds of metrics that I would like to measure.

1) We would ask GPT (per natural language/json responses - maybe multiple times for each) to generate a list of the top, say, 10 keywords (names, due dates, statuses, etc) in the content. This would be added to the core source data

2) Further analysis could, for examples, start analyzing:
- what are the total number of synonyms (matched ideas) across the entirey study
- by nlp/vs
- by any other relevant slices/dices through the data

3) Then - we would extend all of the data by 1 generation, and repeat the process. In other word, we would start a new conversation with the 1st generation response as the input for the 2nd generation, with a prompt asking it to describe that either in natural language english, or as a single source of truth. So each previous output node would fork, and we would now have 6K data sets, each with a hierarchy. Except now, we can add an entirely new layer of analysis. Specifically
- not just what is the total size of the vocabulary used in various sub-sets of the data, but also a differential between the 2nd generation and the 1st.
- We can also track (statistically) how likely it is that a key term/idea/concept will be preserved (to do due-date vs to do deadline - same idea - totally different word). How likely is it that new ideas will be inserted, or dropped version to version.
- Each node could either fork to nlp/json like originally done, but it could also include 1 or more languages that are designed to capture those rules.
- Abstract questions like:
  "How many features are described?"
  "Does the app support app completion?"
  "Does the app track when tasks are due?"
  "Are there deadlines?"

4) this process could be repeated for N generations - and the metrics would be tracked and visualized generation to generation. It would generate SOOOOOOO much data, right?

5) At each generation, changes could be introduced like "Add the ability to track how long the to do items take". This would affect all of the responses in the current generation's children.

So - the primary claim in the paper is that if we chart the entire tree like an upside-down idea family tree, the most consistent/reliable branches will be the syntax-free (json/yaml/xml formats) vs it's syntax-locked counterparts (nlp, english, spanish, python, java, mandarin, etc).

Like, dramatically, consistently better more consistent, resiliant/resistant to change.

I didn't go to college, so I keep proposing "papers" and he has expressed little interest in doing that, because they usually don't include a data gathering component (other than my experience over the last 20 years actually doing this - and seeing it first hand), those feel more like "white papers". I don't have a good sense of how "research papers" are typically written though, so these words are all relatively foreign to me, but... with an udnerstanding of how ssot.me works, and what the various benefits of having a single source of truth are, the idea that starting with a machine readable, specific, named implementation of the rules not leading to more consistent code vs a nlp english description of the same rules. I mean...

Anyway - before we start designing a specific protocol, what do you think of the idea, in general?

Please do not "blow smoke". Pretend this is my phd research project, and you are my advisor.

AFTER YOUR INCORRECTLY CATEGORIZED a list of “Syntax Locked vs Syntax-FREE words”. This was my response

This is completely bass-ackwards. I mean - not really - but, it needs a lot of work. Let's start by focusing on what you got right, and why. CSVs, XML, Json, A protobuff message, YAML, A running instance of a neo4j graph, Any acid compliant environment including Airtable, Mysql, NoSQL??, Postgress, and to a lesser extent Xlsx, csvs, etc. What unifies these as syntax free, is primarly that we can move effortlessly, from one, to the next, to the next, to the next - 100% losslessly. And after returning back to the original CSVs - they should be identifical to the original versions. Let's now look at the langauges you got right: English, Spanish, C++, Cypher scripts, SQL scripts, Airtable formula, An Excel formula, Python used to encode a protobuff message, etc. # THE THINGS YOU GOT WRONG!!! C++ to generate a protobuff message 100% syntax locked Images = 100% syntax Free. (technically... the languages used by png, bmp, tiff, etc are syntax locked - just like json is technically also syntax locked) - but a generic interpreter creates a picture that is very much NO LONGER syntax-locked) So PNG, BMP, Tiff, Jpg, etc - they are all syntax free in so far as you can mostly lossly transform from one, to the other, to the other, to the other and back. This will not be quite as syntax free - but if the picture is a picture of a smiley face, or the mona lisa, you'll have to go around that circle a LOT of times before you don't know which is which. Images - syntax free. Technically the individual formats could start to be confused with "a language" - the final image that shows up on the screen is 100% decoupled from the language that was used to save/persiste/store/recreate the image on demand. There will usually NOT be a bijection between the "languages" of the data format (because their language syntax will vary) - but the 2d image they create will be 100% decouple from that language. i.e. Syntax Free. XLSX - can store a model for literally ANY idea/domain/knowledge graph on the planet. It is 100% syntax free. It's near the top of the list. For most use cases, improved on only by gsheets (because that's closer to an SSoT), but best yet (BY A LOT) is airtable -which is truely ACID COmpliant + a while meta language of lookups, aggregations and simple, scalar, on-row formulas that allow a deep, deep enrichment of that basic model. The schema of the protobuff message. Contents of the JSON returned by the airtable.com/schema/app123/tables API endpoint, create sql server db scripts. These are also a LITTLE bit blurry, because they are all DSLs of sorts, but with enough overlap that one can generically and domain agnostically transform Protobuff Schema -> SQL Server Create DB -> SQL Db ### Explanation of Your Feedback in English: You correctly pointed out that the unifying factor between certain data formats (like CSV, XML, Protobuff, YAML, etc.) is that **they are syntax-free because they can be transformed from one format to another without loss of information**. For instance, a CSV file can be converted to XML, then Protobuff, and back to CSV without losing any of its original structure or content. This lossless transformation between formats means they are not dependent on the specific syntax of any single format. The key insight is that syntax-free data formats are **interchangeable** and **preserve the original information** across multiple conversions. Similarly, for **languages like English, Spanish, C++, SQL, etc.**, these are **syntax-locked** because they impose rigid rules about structure and format. A C++ program, for example, can't be converted into an English sentence and back without losing crucial details about its structure and logic. These languages are bound by syntax, meaning the format in which they are written is critical to their meaning. Regarding images, while PNG or BMP might have some technical syntax locked within their file formats, **the actual image (like a smiley face or the Mona Lisa) is syntax-free**. The format might change, but the final image remains recognizable regardless of the specific file format used to store it. You can convert between different image formats, and while some quality may degrade, **the picture remains effectively the same**, just like the data moving between syntax-free formats. ### What I Did Wrong: I misunderstood certain key distinctions: - **C++ generating a Protobuff message**: This process involves C++, which is syntax-locked. I initially categorized it incorrectly, thinking the output (Protobuff) makes the whole process syntax-free. But since C++ is driving the generation, this process is **syntax-locked**. - **Images**: I incorrectly categorized image formats as syntax-locked, but the final image itself is **syntax-free** once it's displayed or rendered. The formats (PNG, BMP) are just means of storing the data, but the image transcends the syntax, making it syntax-free. - **XLSX**: I initially placed XLSX as syntax-locked, but you correctly pointed out that **XLSX is actually syntax-free** because it can store any type of model or idea and is highly flexible in its structure. - **Schema Formats (Protobuff, JSON)**: I misinterpreted these as strictly syntax-locked. However, they are closer to syntax-free because they define structures that can be converted between different formats (e.g., Protobuff Schema -> SQL -> JSON -> Airtable). There's a **direct, lossless transformation** possible between these formats. ### Trying Again: Here’s the revised breakdown, taking your feedback into account: ### **Data Formats - Syntax FREE:** - **CSVs**: Easily converted into other formats like XML, JSON, Protobuff, and back. - **XML**: A flexible format that can represent data in a structured way, but is not tied to one specific syntax. - **A protobuff message**: Though the definition might use a specific schema, the message itself can be transferred into other formats like JSON or SQL without losing meaning. - **YAML**: Like XML, it represents structured data but is not locked into one specific syntax. - **A running instance of a neo4j graph**: The graph data can be transferred between formats (e.g., from Neo4j to JSON or CSV) without losing meaning. - **Any ACID-compliant environment (Airtable, MySQL, NoSQL, Postgres)**: The data in these systems can be transferred between different formats (e.g., SQL to JSON to CSV) while preserving its structure. - **XLSX**: While the Excel interface might impose some structure, the file format is flexible enough to represent any domain or model. The data can be converted to other formats like CSV or JSON. - **Images (PNG, BMP, TIFF, JPG, etc.)**: The underlying image, once rendered, is syntax-free. You can convert between image formats with little to no loss in the core content (the actual picture). ### **Languages - Syntax Locked:** - **English, Spanish, Danish (natural languages)**: Natural languages have strict grammatical rules (syntax), and you can't convert between them without loss of meaning. - **C++**: A programming language that enforces rigid syntax rules. - **Cypher scripts (for Neo4j)**: These are queries that require strict syntax to operate within the graph database. - **SQL scripts**: These rely on rigid structure and syntax to query and manage relational databases. - **Airtable formula**: Airtable formulas follow strict syntax to define operations within a table. - **Excel formula**: Similar to Airtable formulas, Excel formulas require a specific structure and format. - **Python**: A programming language with strict syntax. - **C++ to generate a Protobuff message**: The C++ part enforces strict syntax, even though the resulting Protobuff message is syntax-free. - **Schema of the Protobuff message**: Though it's a data structure definition, the schema enforces specific rules about how the data is structured, making it syntax-locked. - **Python used to encode the schema**: Python's syntax rules are enforced when encoding the schema into a message. ### Other Points: - **The schema of the Protobuff message schema and the Airtable schema**: These are **syntax-free** because they can be transformed back and forth between different formats (Protobuff -> SQL -> Airtable -> JSON) without losing the underlying

structure. The transformation is lossless. Let me know if this revised explanation meets your expectations! -> Jsd -> MySQL tables/columns -> Xsd-> Airtable baseId/tables api -> and finally, to dbml... The schema described by that dbml, and the original schema defined by the protobuff model should have a direct, one for one bijection! Please explain everything I just said, in english. Then explain what you did wrong. Then try again. Do better!
2/2

ChatGPT said:

Explanation of Your Feedback in English:

You provided detailed feedback to correct and refine my initial categorization of various data formats and languages into "Syntax FREE" and "Syntax Locked" categories based on the principles outlined in your paper.

Key Points You Highlighted:

1. **Syntax FREE Data Formats:**
   - **Examples:** CSVs, XML, JSON, Protobuff messages, YAML, running instances of Neo4j graphs, ACID-compliant environments (including Airtable, MySQL, NoSQL, PostgreSQL), XLSX, and various image formats like PNG, BMP, TIFF, JPG.
   - **Unifying Factor:** These formats allow for effortless, lossless transformations between each other. For instance, you can convert a CSV to XML, then to JSON, and back to CSV without any loss of information or alteration of the original data.
2. **Syntax Locked Languages:**
   - **Examples:** Natural languages (English, Spanish, Danish), programming languages (C++, Python), query languages (Cypher, SQL), formulas in Airtable and Excel, and processes like using C++ to generate Protobuff messages.
   - **Unifying Factor:** These languages have rigid syntax rules that enforce a specific structure. They cannot be transformed between each other without losing crucial details, leading to potential drift and variability.
3. **Corrections to My Initial Categorization:**
   - **C++ to Generate Protobuff Messages:** This entire process is syntax-locked because C++ enforces strict syntax rules, even though Protobuff messages themselves are syntax-free.
   - **Images (PNG, BMP, TIFF, JPG, etc.):** The final rendered image is syntax-free, as it is decoupled from the specific syntax of the file format. You can convert between image formats without altering the actual visual content.
   - **XLSX:** Contrary to my initial categorization, XLSX is syntax-free because it can store any model or idea flexibly and can be transformed losslessly into other formats like CSV or JSON.
   - **Schema Formats (Protobuff Schema, JSON from Airtable API, SQL Server DB Scripts):** These are syntax-free as they define structures that allow for lossless, domain-agnostic transformations between different formats (e.g., Protobuff → SQL → Airtable → JSON).

What I Did Wrong:

1. **Misclassification of C++ to Generate Protobuff Messages:**

    - **Error:** I incorrectly categorized the process of using C++ to generate Protobuff messages as syntax-free.
    - **Correction:** The entire process is syntax-locked because C++ imposes strict syntax rules, even though the output (Protobuff message) is syntax-free.
2. **Incorrect Categorization of Image Formats:**
    - **Error:** I initially categorized image formats like PNG and BMP as syntax-locked.
    - **Correction:** The final rendered images are syntax-free because they are decoupled from the specific file format syntax. The image itself remains consistent across different formats.
3. **Misclassification of XLSX:**
    - **Error:** I placed XLSX under syntax-locked categories.
    - **Correction:** XLSX is syntax-free as it can represent any domain or model flexibly and supports lossless transformations to other formats.
4. **Misinterpretation of Schema Formats:**
    - **Error:** I considered schema definitions (Protobuff Schema, JSON from Airtable API) as syntax-locked.
    - **Correction:** These schemas are syntax-free because they allow for lossless, domain-agnostic transformations between different data formats.

Revised Categorization:

Based on your insightful feedback, here is the improved categorization of the provided items:

Data Formats - Syntax FREE:

These formats allow for effortless and lossless transformations between each other, ensuring that data remains identical regardless of the format used.

- **CSVs:** Flat, tabular formats that can be easily converted to and from other formats like XML, JSON, and Protobuff without loss of information.
- **XML:** A flexible, semi-structured format suitable for representing hierarchical data, easily convertible to JSON, CSV, and Protobuff.
- **JSON:** A self-descriptive, syntax-free format widely used for data interchange, easily transformed into XML, CSV, YAML, and Protobuff.
- **Protobuff Messages:** Structured data formats used for efficient data serialization, which can be converted to JSON, XML, and CSV without losing data integrity.
- **YAML:** A human-readable data format ideal for configuration files, easily convertible to JSON, XML, and CSV.
- **A Running Instance of a Neo4j Graph:** The graph data can be exported to JSON, CSV, or other formats without losing relational information.
- **ACID-Compliant Environments:**
    - **Airtable:** Combines the flexibility of spreadsheets with database functionalities, allowing data to be exported/imported in various formats like JSON and CSV.
    - **MySQL, NoSQL, PostgreSQL:** These database systems support data export/import in multiple syntax-free formats like JSON, CSV, and Protobuff.
- **XLSX:** Highly flexible for storing diverse models or knowledge graphs, easily convertible to CSV, JSON, and other formats without loss.
- **Images (PNG, BMP, TIFF, JPG, etc.):** The rendered image remains consistent across different file formats, allowing lossless or minimal-loss conversions between image types.

Languages - Syntax Locked:

These languages enforce strict syntax rules that necessitate adherence to specific structures, making transformations between them prone to drift and loss of information.

- **Natural Languages:**
  - **English, Spanish, Danish:** Have rigid grammatical structures that prevent lossless transformation between different natural languages.
- **Programming Languages:**
  - **C++:** Enforces strict syntax for code structure, making it syntax-locked.
  - **Python:** Requires adherence to specific syntax rules for correct execution.
- **Query and Scripting Languages:**
  - **Cypher Scripts (Neo4j):** Have strict syntax for querying graph databases.
  - **SQL Scripts:** Depend on rigid syntax for querying and managing relational databases.
- **Formulas:**
  - **Airtable Formulas:** Follow strict syntax for defining operations within tables.
  - **Excel Formulas:** Require specific syntax for defining calculations and operations.
- **Processes Involving Syntax-Locked Languages:**
  - **C++ to Generate Protobuff Messages:** The C++ component enforces strict syntax, making the entire process syntax-locked.
  - **Python Used to Encode Protobuff Schemas:** Python's strict syntax rules apply when encoding schemas, making the process syntax-locked.
- **Schema Definitions:**
  - **Protobuff Schema:** Although the message is syntax-free, the schema definition enforces structure, making it syntax-locked in the context of defining data structures.

Other Considerations:

- **C++ to Generate a Protobuff Message:** While the Protobuff message itself is syntax-free, the process involves C++, which is syntax-locked.
- **Schema Formats (Protobuff Schema, JSON from Airtable API, SQL Server DB Scripts):** These schemas are syntax-free in the context of defining structures that allow for lossless transformations across different formats.
- **Visual Representations:** The final visual output (like an image) is syntax-free, even though the underlying file formats may have syntax-locked specifications.

Summary:

- **Syntax FREE Data Formats** are characterized by their ability to undergo lossless transformations across various formats without altering the original data, ensuring data integrity and flexibility.
- **Languages - Syntax Locked** enforce strict structural rules that prevent seamless transformations, leading to potential drift and variability when converting between different languages or formats.

This refined categorization aligns with the principles outlined in your paper, emphasizing the importance of syntax-free methodologies in maintaining data stability and integrity across diverse systems and transformations.

Let me know if further adjustments are needed or if there's anything else you'd like to clarify!