EasyChair Preprint

№ 7848

Machine Trainable Software Models Towards a Cognitive Thinking AI with the Natural Language Processing Platform NLX

Felix Schaller

May 18, 2022

Machine trainable software models towards a cognitive thinking AI with the natural language processing platform NLX

Felix Schaller[1]

Abstract: Since the last decade, machine learning, especially with artificial neural networks, has triggered a new quantum leap in computer science. Despite the considerable achievements, these applications still lack a general purpose approach for artificial intelligence (AI). The main reason is the absence of the ability for cognitive reflection or self-awareness. They are mainly highly specialized trained patterns that can solve intricate problems but cannot describe themselves. I would like to contrast this with a new method of trainable software models that shall be capable for self-awareness. Implemented in the project Natural Language Platform NLX it shall be demonstrated that self-aware AI is key for human-like cognitive tasks. The hypothesis claims that to reach this goal, machines require to describe its system context semantically by a formal model. Neuronal networks are good at specific tasks, but the trained patterns cannot derive a reasoning for the trained solution. Only that it satisfies its intended functionality - but not why. Creating formal models instead of patterns has turned out, that the formal nature of natural language is the best to reach that goal of a self-aware AI. Certainly there are other AI's that do natural language processing with neuronal networks. But most of the models try to resolve the content with too rigid constraints and with little attention to the context. For this project context plays a key role to resolve the meaning of natural language. If the context is resolved correct such AI can be used for general purpose tasks resolving anything imaginable.

Keywords: Machine Learning; Software Models; AI; Cognitive thinking; Cybernetics and Systems Theory; Computer linguistics; Anthropology

1 Machine learning today

The last decade can be seen as the great breakthrough in artificial intelligence, where finally by the leading role of neuronal networks, computation solutions could be achieved which where not possible before by using primarily formal methods and algorithms. Meanwhile almost all groundbreaking improvements by AI go on the account and the subdomain of neuronal networks. They perform superior especially in domains where there are no formal solutions possible, because no formal rules are available. Such domain is in particular the computer vision domain where patterns play a major role. But also for other domains like natural language processing, neuronal networks meanwhile conquered a lot of computational domains where they now rule over formal methods and algorithms that attempt to do the same task. In the last few years the project openAI [Op21] with their newest language

[1] Validas AG, IT research, Arnulfstr 27, 80335 München, Germany schaller@validas.de

model GPT-3 [FC20] has achieved astonishing results, where this AI can even create intricate software code from natural language [Ch21] that is even executable. Also in other domains of natural language processing GPT-3 in conjunction with neural theorem prover has achieved astonishing results by solving some Math Olympics problems [Po22]. But yet when summarizing all these success stories the impression prevails, that these solutions can be good at a specific domain, but yet still require a human guided setup or operation frame to master the task. In other words it needs to be specialized for the domain to perform. Meaning that it would neither be capable to develop the solution by it's own nor to reflect it's result. Something you would expect by a general purpose tool. So it can be confidently stated it is yet a stupid - indeed a very efficient stupid - machine.

2 The nature of neuronal networks

What makes us confident in that claim? the reason lies in the nature of neuronal networks itself. Though if stacked together in huge networks they can even easily surpass a human on that specific domain, but still they can only perform tasks for what they are designed for. It can be anyway seen as a general purpose tool from that perspective, that the problem posed on can be general purpose indeed. But yet it processes only tasks for which it is specifically tailored for and not any task by one single interface like a HAL 9000, best known from Stanley Kubricks "A Space Odyssey". The reason is this networks does the task, but it does not know why. It doesn't even know if the result produced is correct, except that it satisfies an intended cross reference. But here lies the rub, that to develop unique solutions, the AI requires a formal reason.

Neuronal networks are not capable by nature for that task because they currently supersede on problems nobody knows a formal solution for, so you unleash a neuronal network on it to intuitively find a solution with heuristic methods, like humans develop a gut feeling for a certain domain by experience, a network achieves positive success rate over time. After achieving a finite amount of training the network became confident in estimating the result. But no matter how precise the network performs, it's yet an estimation - indeed a highly sophisticated estimation - of the solution. This solution is anyway just a product of the training set posed to the network and in an unluckily constellation in the selection of training material during training, the network later may show unintended behavior leading to false positives [FS10]. One reason are unknown patterns on a subliminal level or patterns that seem not obviously relevant by human interpretation being mixed in the training data which influences the final result, best known since decades from the 'tank problem' [Ri19]. Another example are adversarial attacks by noise. By nature a neuronal network is not so resistant against such type of signal jamming [NKP19]. Here the major reason is, that the network is applied to the raw data finding patterns on the atomic level. Would the signal instead be transposed by Fourier Transformation, a noise jamming could be isolated easily [ECK19].

2.1 Functional safety

Based on the fact, that no matter how sophisticated the network is, it is at the end still a highly sophisticated guessing, achieved through training experience by no formal rule. That makes it to the ultimate tool for unresolved problems but on the other hand makes those networks actually insufficient for functional safety domains [GB19]. Currently there are several safety standards under development which discuss criteria and evaluation methods under which circumstances they are acceptable [Ts20] [et20]

2.2 The solution is encrypted in the matrix

The final problem then is, that the trained solution is finally hidden in some patterns of the various matrices inside the network, which does by no way unveil its reasoning for the solution. Ironically by the reason that the network was the only available choice, because there was no other formal method accessible. Indirectly, in this problematic there lies also a huge potential towards a paradigm shift about the meaning of formal methods and machine learning. This paper also addresses the current paradigm of formal truth in later chapters.

2.3 Natural language processing and neuronal networks

As natural language has yet been seen as too ambiguous for formal methods the solution lays near to use that available tool that can handle problems with unknown formal rules. And as the success story of the GPT-3 model reveals, it does this job pretty well and precise although language is seen as something too ambiguous for deterministic tasks like creating software code. The reason is, that the hypothesis of the ambiguous nature of language is actually not true. It is only true if you look on literal terms as formal or imperative expressions. And that's why solving natural language was yet prevailed to the domain of neuronal networks. But language has indeed a very formal nature and can be very powerful if you look at natural language from a different point of view. This matter will be very essential for the topic of this paper and will be dedicated in particular in a later chapter.

3 The formal method and machine learning

As laid out in the chapters before, it is obvious that machine learning methods - in particular with neuronal networks - computation has penetrated into areas which were not available before with formal methods. But that alone would not make a system more "intelligent". Because yet it does not provide a reasoning, at least and most for itself to build formal conclusions from it. The simple reason is that to know what I am dealing with, I require to describe it semantically furthermore this semantics requires a context. The context becomes key in that matter, and in this context also lies the reason why formal methods fail where machine learning succeeds - because they do not consider a context.

3.1 Structural science and the Hilbert program

When looking today on the domains of math and information science it is all about reasoning. In particular universal validity of claims and their proof. This was believed till Alan Turing introduced its Turing Machine as answer to David Hilbert's Program for an axiomatic proof theory [Pl14]. Turing demonstrated with this machine theory, that there is no ultimate law, that any task can be solved by formal rules. This paradigm thus defines the foundation of all information theory - but actually it is only half true. It is only true when applied in the frame and to problems of the David Hilbert Program. Although the problems stated there gather all formal problems in information technology till today. Basically what all axiomatic theorems do, is to set up a claim that pretend to have an ultimate validity and the algorithm has to be just sophisticated enough to handle that problem. To break out from that paradigm it requires to look at it from a different perspective. If an algorithm does not resolve or the theorem does not prove it's not necessarily the algorithm, but the constraints that are set up that cannot be fulfilled. So if a Program does not return from it sub-states it does not automatically mean the problem is unsolvable, but the constraints are either specified wrong (misconception of the author) or are underspecified (missing information). Practically this is the result of bug-fixing where developers getting feedback about the misconception or suitable preconditions for the algorithms. It's the essence of all systems, that they require the right context to work in. Even in nature species can only exist in their suitable biotope. Axioms and theorems do not come with such a system context, thus the context must be created first and ensured it suits the formal method properly. Something like an immune system for theorems. That's mostly the job of functional safety.
The deepest insight i got in that problematic myself when designing a sophisticated DSL to parse natural language documents. It requires to have so many information on certain abstraction levels that it is almost impossible to cast this into a rigid abstract syntax tree of a DSL. But on the other hand it revealed such a deep insight how system contexts are made up and that certain information is required on a certain branch of abstraction to decide further in higher levels of abstractions. Alongside it provided also a rough impressions how neuronal network learn such proper tree structures by a heuristic approach. I suggest that such an approach could maybe one day help to find a solid theory how neuronal networks develop their decisions. Nevertheless due to the high amount of semantic information a DSL cast into formal code is an unsuitable approcah to create a solid DOM structure but sufficient to provide a proof of concept for the natural language processing. So far to say that whole structure problematic in the DSL alone holds enough material to occupy with it in a separate Paper.
Carrying on with the problematic of formal methods, the actual reason why they fail can be summarized:

1. Because they claim to have ultimate validity. All constructed models that reflect a certain aspect of a real problem are true only in a certain area. No model can claim from itself being valid for all circumstances.

2. If a Turing Machine fails because it does not return from its pushdown state it means the problem is underspecified and lacks constraints it requires to solve the posed problem.
3. Because of the self-entitlement of ultimate validity of axiomatic theorems they claim to work context free.
4. Due the negligence of meta structures in the data, those axioms are applied on raw and unstructured data on an atomic level. Thus lacking a structured semantic of the data and therefore any formal method will fail, because ambiguity is not resolved properly.

3.2 The key role of context

As already introduced in the last chapter, the problem with all formal methods is, that a theorem cannot stand by its own. All formal constraints require a certain frame where they can be seen as valid and that frame represents a suitable context.
EXAMPLE 1: Try to qualify a software for a functional safety domain. Here it is mandatory to draw a line around finite tasks, named use case, of within they can be declared as safe after undergoing a certain process of validation. Any contemporary program is intended to work in any context so the degree of freedom in that program is reduced to that amount of states to work safe under any circumstance the user can bring it to and what the safety case allows.
EXAMPLE 2: A further example what's closely related is the already mentioned challenge in the last chapter. The challenge to develop a grammar for a domain specific language that can be unleashed on any text to resolve the document structure from it. After that challenge is accomplished, it is cast in a document object model. DSL's will work reliably on formal languages, that's what DSL's where designed for, but there will be no ultimate valid and error-free grammar to solve the content of natural language documents - including the interpretation of the document structure - without a certain knowledge of the context of the text to resolve.
Further examples in other domains are like automating tasks in continuous integration (CI). It normally requires always someone who sets it up manually. The cannot be feed into an automated process, because those automation processes are lacking knowledge about what the given data represents and what shall be processed. Yet the maintenance of such CI pipelines is restricted to a developer, due to there is no additional information for the system to source in order to fix itself in case of an unexpected state or condition. Such systems thus require additional design information. And such information can only be provided by a proper knowledge of the context they process.

3.3 Fractals and meta structures of reality

By the theoretical proof through the Turing Problem someone may formulate a law, that reality is so complex it cannot be described by formal methods and therefore forget that formal methods though work but just within a certain frame. Applying it on a given context it reveals that there are so many ambiguities to resolve properly. The very reason for that is, that the reality has a self similar nature. I make a bold claim that this circumstance is causing all the ambiguities when applying a certain theorem on a matter. Not that it is just self similar but also fractal, those self similar features build upon fractal meta structures which build the ground truth and are the backbone of any cybernetic system. As it can be shown that systems work within a certain frame, it also shows that parts of the system can be described formally. Therefore, when there exist rules for sub-systems based on the fractal axiom, the entire system can be formally described in systems of systems, where each super-system create conditions where subsystems can exist. Gregory Bateson a pioneer in cybernetics and systems theory shaped the term of "meta structures" in his book "Mind and Nature" [Ba79]. He sees those structures as immanent and as a implicit consequence of growing complexity in systems. This means for a practical application, that without a proper knowledge of the context you are unable to distinguish self similar features from each other on an atomic level. Many different problems appear identical on an atomic level and thus cannot be distinguished. This problem leads to an ambiguous situation how to distinguish and categorize the data features. Unfortunately most applications work solely on the atomic level without a structure semantics. Simple experiments with computer vision mimicking the human perception can be made to understand the need to create meta structures of the given data and relate them to the given context. Here may be given two distinct problems where this matters:

- Taking the halftone image as example in the Fig. 1, looking here in a closeup on the patterns would not reveal the content of the matter. Only when you zoom out, the human perception would resolve a pattern of a white horse. Which means the information is hidden when just looking on the atomic level of the patterns itself. They do not really distinguish from each other, because a single feature does not carry the information. The information is encoded deeper in the meta-structure.
- Same problem occurs when designing computer vision algorithms for autonomous driving or augmented reality. Resolving the observer position from a time sequence of a video-camera, it requires to find recurring patterns in the image sequence to track and locate them over time to render the motion of the observer in 3D-space. Such algorithms are know as SLAM algorithms (**S**imultaneous **L**ocalization **A**nd **M**apping). Earlier Algorithms have solved it by adding markers on very distinct features in the images like the PTAM algorithm in Fig. 2. But such an algorithm gets confused if it is opposed to redundant features in the image like on a checker board. In this case it is unable to distinguish to which of those features it shall assign, given by the last image frame. The result is, that trackers are jumping between the

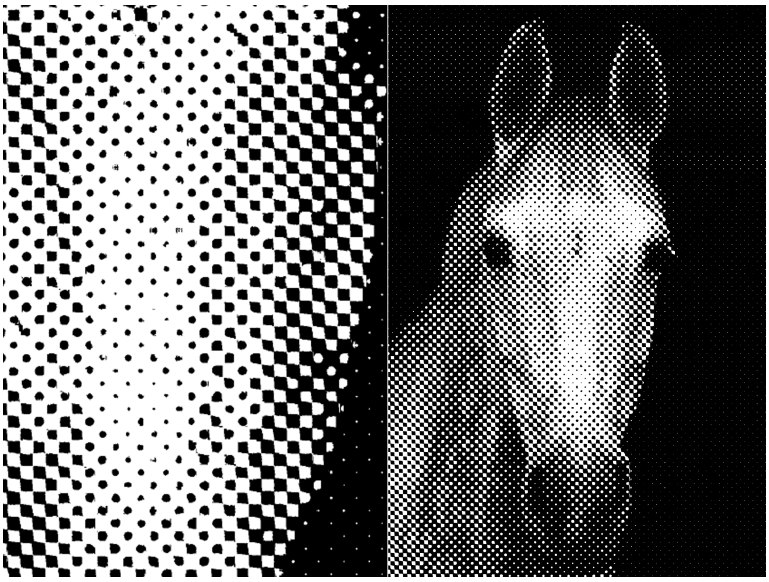

Fig. 1: (left) a closeup view on the raster does not reveal the motive, while on the right picture the motive can be clearly percepted

Fig. 2: Problems in older computer vision algorithms like PTAM: Feature-trackers get confused by redundant patterns in the camera image which makes the result unusable

redundant features. This creates a jittering path, which makes a further resolution of the observer position impossible.

Concluding from these examples, it seems obvious that for a proper interpretation of posed problems it will be essential to develop a structured semantic for them that they can be formally resolved. Otherwise the problem gets stuck in ambiguity. So the next question arises here: how to build up meta structures on data to derive a context so it can be processed with formal methods.

4 Towards formal models in AI

Carrying on with the success story of neuronal networks from above I mentioned the circumstance that they penetrate in areas where there is no formal rule known by applying heuristic methods of intense training. By training the neuronal network derives hidden meta structures of the trained content and stores it in its patterns of several hidden layers. But those meta structures do not own a semantic. So they can not explain what they are doing and develop reasoning for that. Thus it requires to develop a semantic to interpret those meta structures.

4.1 Introducing software models to reduce complexity

Even when working with software models which promise by structuring the matter to reduce complexity, like models from the SysML and UML domain, they can help to sustain an overview for the user and if done by experienced system engineers it can speed up the development process and in some cases it even can be generated software code from. But as the modeling languages UML or SysML often want to reserve the flexibility to describe systems, users can create huge mess. Many modeling automation projects in the industry, starting with great ambitions aiming to automate the development process and to leverage systems and software development to the next level not seldom died an inglorious death. Some of them are still riding a dead horse because the companies already poured millions of their budged into the project fearing all that investment to be lost if they redesign their concept from the bottom up, so they insisted to carry on and exacerbate. I have consulted some of them and was not reluctant to provide them with an honest analysis of the tricky situation. The main reason in those projects were that there was no proper modeling semantic given that maintained a certain methodology hygiene. Especially because in many modeling tools available on the market there is neither type checking nor other methodology checking automatism implemented. With the final result that the created models were useless for the further processing in downstream system domains and can finally only served as a form of a documentation of the process. This circumstance arises the need for a proper modeling theory. For that purpose the FOCUS theory of distributed systems by Broy et.al.

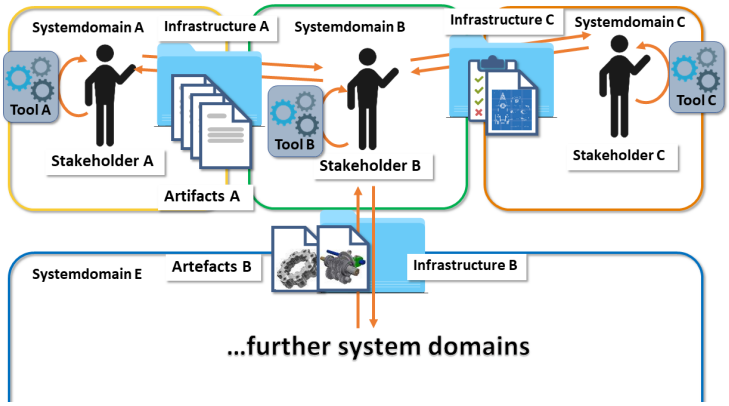

Fig. 3: Exchanging data artifacts across system domains require intense manual administration

was developed [BS01] This modeling theory is currently implemented in the SPES modeling framework which is constantly being developed further in various research projects and industrial applications [Bö16]. Having a proper concept of various viewpoints and the concept of granularity levels, the framework does a good job in improving the quality of model based systems. Meanwhile it has been widely established in the industry - and added with a proper tooling, it has the potential becoming the new standard in systems engineering.

4.2 Getting the developers knowledge into the model

As shown in the last chapter that a proper methodology accompanied with a powerful framework is key for a good success story in model based software engineering, and if such a proper methodology is properly implemented it can avoid nightmares in systems engineering. But although the model based approach improved the development in software and systems, those models still requires the interpretation of the developer because the models do not own its own semantics. To the computer all software models are simply a finite collection of nodes and edges - nothing more. Yet that is nothing that could live up with the fancy claim of a cognitive thinking AI as promised in the title, but we are not there yet. This is the case simply due to the lack of knowledge for the machine - and that remains with all data artifacts in information technology: They create meaning for the developer or expert, but not to the machine itself, simply because they hold no meta information for the machine about what they are and how they have to be treated. First and most, literals that are used by the developer to name identifiers give meaning to the developer and to understand what it is about, but the machine lacks an interpretation of the literals to create its own semantic.

4.3 Introduction of Meta Information

A first step to help the machine resolving the context would be to implement a semantic by introducing meta information as an overlaying layer that tells the machine how to handle the model elements. This meta information introduce a classification system with a proper taxonomy to describe a model element - lets say we have a model that describes artifact flow within a tool chain and contained within a model element that shall represent a specific tool e.g. a GCC compiler. Then we assign a class hierarchy which derives that specific compiler tool from a generic compiler up to higher super classes: tool, executable etc. Further it is defined that this tool has certain interfaces to other tools. Now when we then assign that class to a tool in a tool chain the modeling tool knows what kind of data that tool can interchange with other tools. If we now want to define a artifact flow from one tool to another, the modeling tool already can do some automatic assignment because of the semantic given in that meta-description. By that it can derive constraints of what kind of data can be interchanged between those tools. By building up such meta-information the system engineer no longer need to explicitly assign any data connections by himself manually but the software already knows what connections are possible and can assign all appropriate connection by themselves. Here at that stage it would be already a semi automatic and proactive tool. Such proactive actions can be a great help, but frequently proactive behavior in programs create a huge nightmare.

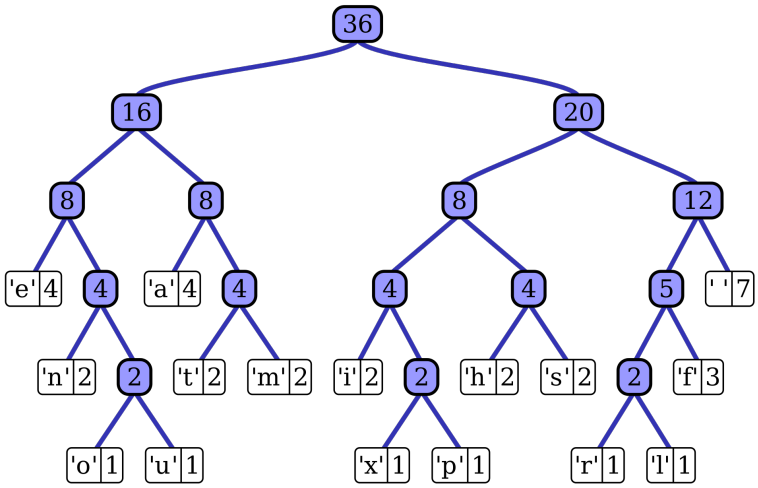

Fig. 4: A Huffman Tree describes a hierarchy of redundancies and is widely used for data compression and image-processing. it can be seen as a meta-structure describing redundant features in a context

4.4 Interactive modeling and machine trained models

Many professional computer users who where already using Computers with Microsoft Software around 1995 are very likely to have come in touch with Microsoft®Word™back then. At that time Word had an digital assistant who tried to guess the users current actions. So if you were typing a postal address in a document, the digital assistant appeared from nowhere saying “Ah obviously you want to write a letter”. In some rare cases the guess would have been right. But honestly in most cases this little lad was more than an annoying nag. The reason was very obviously, that in most cases the unordered and spontaneous interventions in form of an interactive assistant where inappropriate for the current situation. The main reason that seems most obvious here was, that the solutions offered were too stenciled that it could have been addressing real individual needs for the user. This originated from the cause that you could not interact with that assistant like with a real human person and respond: “sorry you misinterpreted, this is not going to become a letter but an address list”. Then next time the assistant would have behave different. But because it just imitated basic tasks, based on a stencil this would have not lead to a real semantic to source the specific needs of the user. The Clippy assistant from Microsoft®Word™had a precessor: Microsoft®Bob that originated from an research project Seductive Interfaces by Clifford Nass and Byron Reeves [Sk94]. Based on the research of what can improve human computer interaction those stencils for the assistant are created. But human computer interaction cannot be satisfied by stencils. It requires real interactions with learning effect. So this means that if the computer want to respond to the need of the user, they require a model that can be trained. How such models can be trained? following approaches are possible:

- macro training by recording tasks in a guided way. By that the computer learns by observing user interactions with a GUI and then repeats it on similar problems. All accomplished by an interactive approach. In the case of conflicts the model can further improve to learn to handle special cases.
- Back projection of error. By reverse analyzing logged action the user gives feedback about what was made right and what wrong thereby proactive processes can interactively adapt on the user’s need.

Although this can improve the annoying behavior of proactive assistant, this already opens up an interface for supervised learning based on software models that can be trained. The interface has much in common with declarative programming language but with interactive additions to improve by failure. The interactive approach bypasses the problem that programming implements the constructed worldview of the developer into the solution, which is often incorrect, while an interactive approach is validated by reality itself. And last but not least many conflicts arise by a different interpretation of a topic between different parties. Through own studies with complex software tools like professional 3D programs I made the early experience that supervised training of models to speed up redundant workflows works already on the level of GUI interaction, but still comes on the cost of

manual work because pure interaction lack a fully scalable semantic [fr12]. During that interactive training it turned out that the context created is in its structure very close of what emerges from a conversation. Also inspired by another 3D tool called Wordseye [CS01]. The solution therefore lied near, directly using natural language as an interface.

4.5 A theory about natural language and its potential origin

When discussing the topic of artificial intelligence the performance of artificial intelligent systems is often tried to be benchmarked with human intelligence in that degree how far this system would be away from a real cognitive thinking and human-alike system. Yet many artificial intelligent systems surpass humans on a certain discipline or domain with ease, but overall they could be anyway just compared rather to a certain animal intelligence stage. The main reason is not so much the power of the artificial system itself, but the ability to self reflect, analyze and explain. Human consciousness is superior to the animal not solely of the bigger brain, but because they developed a tool of cognitive perception of the environment and itself. This tool is language. Only by the tool providing a semantic to a given context, the cognitive achievement for the human kind was possible. And this ability can be seen to be the only thing that makes them superior to the animal.

So how could language have been evolved? Studies with primates and kids in the field of cooperability showed that human children intuitively cooperate while primates rather compete against each other. A primate knows that the stronger partner does not share the reward with the other, while kids instinctively shared the reward [HT06]. This arise the thought that the success of the human race might lie in the ability to cooperate and form enterprises to accomplish tasks to what a single individual would not be capable to. For example hunt a mammoth or buffaloes, build pyramids or a temple. For such a venture it requires to synchronize the strategy and to coordinate the tasks among the members of the venture. Therefore a tool is required to transfer the vision and strategy of the leader to all the members. In the same context the cave paintings may also have been arose to support the verbal expressions of the vision. Something that still prevails in contemporary business meetings through charts and pictograms.

4.6 Language as a modeling tool

Talking about natural language processing in the previous chapters, the current practiced approach is achieved by neuronal networks. Language expressions are yet widely understood as too ambiguous to formalize expressions and theorems. But this is only half true. Language is a tool that can be very precise in expressing. Looking at language as formal and imperative commands doesn't work. The nature of language is not imperative but rather cooperative. Language explicitly requires a context to work. At the beginning of a conversation the actual statement is often unclear. By its declarative nature, semantic in language is created by

interaction with the other party. The interaction goes on until all uncertainties are clarified between the parties. This is important because:

- The background knowledge of both parties is mostly different, so common knowledge about the matter must be established.
- Terms in language are not of axiomatic nature and thus do not have a fixed meaning. So the real semantic of a term is either constrained by the context or requires to be negotiated with the other party, till both parties reached an consensus.
- It has a fractal nature and that makes language as a very efficient and information dense modeling tool, where it only requires detailed descriptions where it does not derive automatically by the given context
- A further fractal feature of language is a proper taxonomy of the terms where terms derive from a hierarchic taxonomy structure. This analogy is already successfully instrumented by object oriented programming (OOP) scheme and thus a big step towards natural language processing of instructions. It has turned out since its invention, when the OOP paradigm is understood properly it's a great help in designing applications.

This and more features of natural language makes it very efficient in the expression and communication of intentions because many things can be implicitly derived from the context and do not have to be explained in a redundant manner. Through its truly declarative nature, language has the ability to build formal models that can be used for cognitive human like problems and tasks.

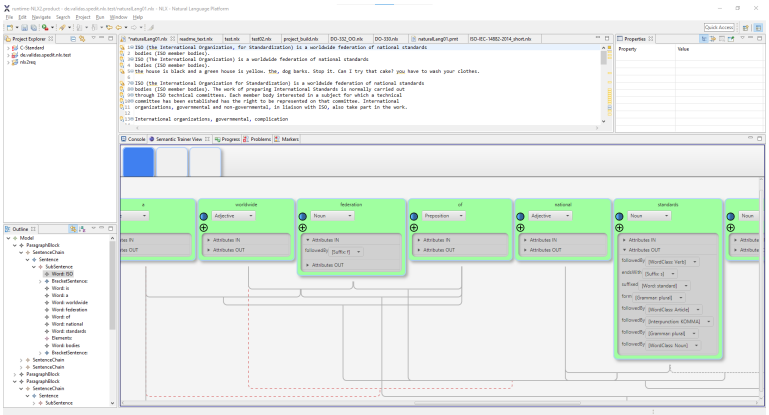

Fig. 5: Overview of the NLX tool. (top) Text document, (bottom) interactive grammar trainer, (bottom left) DOM-tree

5 Experiment: Proof of concept with the natural language project NLX

Based on the prior considerations about natural language processing in a truly formal manner with machine trainable software models, the NLX project was set up. Currently aimed on the goal to provide a proof of concept for the hypothesis of natural language based on machine trained models. Also to show that such models are capable to resolve cognitive tasks and provide an argued reasoning for its conclusions. Something that to our knowledge was not yet published or proven in the field of artificial intelligence. Going more in detail of the architecture, the prototype is built up in two parts. A front-end application, written in Java with the Eclipse Modeling Framework. As back-end i am using the Neo4j spatial graph database to formalize the semantic structure and to resolve patterns with the graphical querying language Cypher.

As textual pre-processor for language processing, the front-end currently uses a DSL, where the grammar is tailored in that way, that it can parse natural language documents containing various structure components like:

- Paragraphs
- Chapters with chapter number
- Sentences
- Tables
- Bullet point and numbered lists
- IT-Words like paths, emails, urls, camel case
- Source code
- Footnotes

Based on these structures it can not only resolve natural language content but also interprets the structure of the document itself. The DSL transfers the textual input into a document object model tree (DOM-tree). On top of this tree all kinds of other generators can be adapted to generate e.g. XMI-Models from that structure. So apart from the unfinished prototype of natural language processing the tool yet offers the working DSL platform that can be used to generate all kind of other products on top that can make use of the structure derived by the DOM-tree. This offers already a rich selection of ready to use use cases which can be set on top of that core platform. At Validas AG we e.g. generate requirements models from industrial standards that integrate the requirements from those standards into other software models.

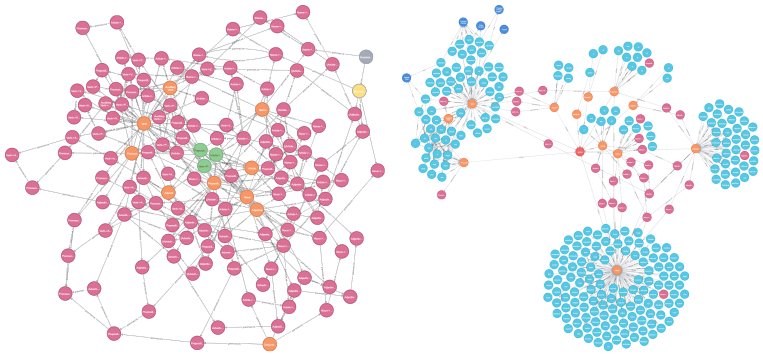

Fig. 6: (left) grammar rules network, (right) rules related with lexical dictionary and other attributes

5.1 Current state of project: Grammar tree and trainer

Currently the development of the natural language processing is in the state of developing a grammar trainer which resolves the sentences in a grammar model. At the moment it trains a grammar structure interactively on a given context of text samples. By working very close with a context it has implemented several common NLP moules like:

- Structure trainer
- Exception pattern trainer
- Lemmantizer
- Tense extraction
- Build-in implicit rules
- Implicit rules-builder

With these features it can be trained in which context certain words types are allowed and what words could be subtyped by providing extra patterns through training input of right and wrong detection. This helps to cluster words and their features in a model with high granularity. Yet not all intended features are working stable, but when this all works reliably then in the future an interlinked ontology model and constraint provers will be built on top. Those provers then would have the role to validate the created context of the statements and derive formal processes from it like analysis tasks, process automation, code generation and many more. The grammar trainer already can be seen as a first proof of concept of machine trainable models. It attempts to resolve the sentence structure like finding an exit through a maze. If the entire sentence has found one root and all branches cover the entire sentence the sentence is resolved. The resolved sentence will then be transferred into the overall ontology of the document. With the grammar trainer it is intended to train a model that is capable

to resolve a sentence structure and separates the parts of subject predicate and objects an further determine time and mode like passive, active, conjunctive. What it basically does is that it intends to create a tree structure of a sentence. Starting with the detection of the right word type where it determines whether the word is a noun, verb, adjective and so on it accesses the database to derive the trained rules in which words can stand in. As a word can have different types, like the word “use” could be a noun or a verb and thus according to the given context it can detect in which type this word acts at that position of the context. The sentence structure resolves then in more and more higher hierarchies, until a final root for the entire sentence is found. When such a hierarchy of a sentence is resolved it can be parsed for entities to build an ontology via entity relationship models. The solution of a

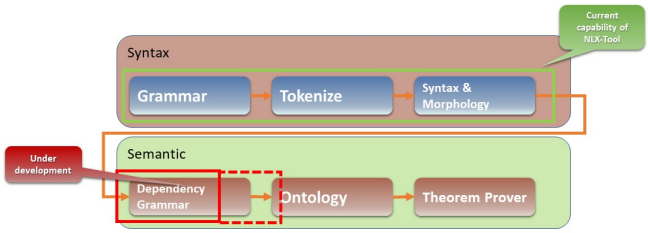

Fig. 7: Current development status of the project

trainable grammar is chosen by the reason, that constructed rules would be too unflexible for the almost infinite variances of sentence structures, thus the structures shall be trained with a training interface to improve the grammar model constantly. Currently the extension of the training capabilities are ongoing and are estimated to be complete soon.

5.2 Future tasks

Before the ontology model can be started the grammar trainer has yet to become more versatile. When this is finished the entities can be parsed into an ontology to derive its functional relationships and its attributes. This document-internal ontology shall then be linked to other related ontologies outside the document or with a database of ontologies acting as the background knowledge of the system. All this is then fed into a kind of “constraint prover” which does the validation of the statements on one hand and resolves the logic on the other hand. Using this prover can formulate all kinds of logical problems converted into software models that can be interpreted or turned into executable code.

6 Conclusion

Based on the general idea of this paper and the experiment findings it can be said: A machine is not automatically intelligent because it is trained excessively and it will finally reach human intelligence if the AI is just big enough. That humans have developed cognitive capabilities is rather exceptional and is able because the human consciousness uses a formal model to understand and explain its environment and himself. This is achieved with the tool of natural language that acts like a meta-model for our self-awareness. Only with such requirements cognitive AI's may be feasible.

References

[Ba79] Bateson, G.: Mind and nature: A necessary unity. New York: EP Dutton. Bateson, MC (1994). Peripheral visions: Learning along the way, 1979.

[Bö16] Böhm, W.; Daun, M.; Koutsoumpas, V.; Vogelsang, A.; Weyer, T.: SPES XT Modeling Framework. In (Pohl, K.; Broy, M.; Daembkes, H.; Hönninger, H., eds.): Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES 2020 Methodology. Springer International Publishing, Cham, pp. 29–42, 2016, ISBN: 978-3-319-48003-9, URL: https://doi.org/10.1007/978-3-319-48003-9_3.

[BS01] Broy, M.; Stølen, K. In: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer New York, New York, NY, 2001, ISBN: 978-1-4613-0091-5, URL: https://doi.org/10.1007/978-1-4613-0091-5_2.

[Ch21] Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. d. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; Zaremba, W.: Evaluating Large Language Models Trained on Code, 2021, URL: https://arxiv.org/abs/2107.03374.

[CS01] Coyne, B.; Sproat, R.: WordsEye: An automatic text-to-scene conversion system. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. Pp. 487–496, 2001.

[ECK19] Esmaeilpour, M.; Cardinal, P.; Koerich, A. L.: A robust approach for securing audio classification against adversarial attacks. IEEE Transactions on Information Forensics and Security 15/, pp. 2147–2159, 2019.

[et20] et.al., J. M. C.: Concepts of Design Assurance for neural networks (CoDANN) - easa.europa.eu, Mar. 2020, URL: https://www.easa.europa.eu/downloads/112151/en.

[FC20] Floridi, L.; Chiriatti, M.: GPT-3: Its nature, scope, limits, and consequences. Minds and Machines 30/4, pp. 681–694, 2020.

[fr12] freshNfunky: Freshnfunky/Intelligentmaya: Cross Breeding Artificial Intelligence with 3D application, 2012, URL: https://github.com/freshNfunky/intelligentMaya.

[FS10] Forman, G.; Scholz, M.: Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. Acm Sigkdd Explorations Newsletter 12/1, pp. 49–57, 2010.

[GB19] Gharib, M.; Bondavalli, A.: On the evaluation measures for machine learning algorithms for safety-critical systems. In: 2019 15th European Dependable Computing Conference (EDCC). IEEE, pp. 141–144, 2019.

[HT06] Herrmann, E.; Tomasello, M.: Apes' and children's understanding of cooperative and competitive motives in a communicative situation. Developmental Science 9/5, pp. 518–529, 2006.

[NKP19] Naseer, M.; Khan, S.; Porikli, F.: Local gradients smoothing: Defense against localized adversarial attacks. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, pp. 1300–1307, 2019.

[Op21] OpenAI, June 2021, URL: https://openai.com/.

[Pl14] von Plato, J.: The development of proof theory, Oct. 2014, URL: https://plato.stanford.edu/entries/proof-theory-development/.

[Po22] Polu, S.; Han, J. M.; Zheng, K.; Baksys, M.; Babuschkin, I.; Sutskever, I.: Formal mathematics statement curriculum learning. arXiv preprint arXiv:2202.01344/, 2022.

[Ri19] Riley, P.: Three pitfalls to avoid in machine learning, 2019.

[Sk94] Skelly, T.; Fries, K.; Linnett, B.; Nass, C.; Reeves, B.: Seductive interfaces: Satisfying a mass audience. In: Conference companion on Human factors in computing systems. Pp. 359–360, 1994.

[Ts20] Tsukiyama, A.: ISO/PAS 21448:2019, June 2020, URL: https://www.iso.org/standard/70939.html.