ABOUT τ-CHAIN

OHAD ASOR

ABSTRACT. τ-chain (pronounced tau-chain) is a decentralized peer-to-peer network having three unified faces: Rules, Proofs, and Computer Programs, allowing a generalization of virtually any centralized or decentralized P2P network, together with many new abilities, as we present on this note.

"If law-making is a game, then it is a game in which changing the rules is a move." - Peter Suber presenting Nomic [10].

Document versions:

   0.2 Feb 6 2015

   0.21 Feb 9 2015

   0.22 Feb 12 2015

   0.23 Feb 21 2015

   0.24 Feb 25 2015

1. INTRODUCTION

1.1. Overview. We [1] propose taking the language of Ontologies to unify languages of:

- Knowledge
- Rules
- Logic
- Computer Programs
- Network Protocols

Ontologies are expressed in RDF language family (Resource Description Framework).

We propose a software client that stores an ontology of local rules, and determines its actions using a Reasoner. Reasoners are tools that infer new rules or conclusions given old ones. They do it intelligently - using pure logical reasoning, and they also supply proofs for their results. τ-chain node is therefore an intelligent agent able to communicate with other agents, at the very same language they're written with, which is quite human-readable.

It can communicate with other languages as well, like HTTP, once implemented in RDF.

Arrow of time is brought into the network using the Blockchain algorithm. Items can get into a Merkle tree that will be signed by a miner, roughly speaking. The network will also function an RDF-speaking distributed storage, namely Kademlia DHT, letting hashes of items to be time-stamped in a mechanism which is up to the rules.

---

[1] My Logic teacher HunterMinerCrafter `https://bitcointalk.org/index.php?action=profile;u=245263`, who introduced this idea and design to me, and myself, Ohad Asor.

The rules of the network are determined by its users. Conversly, many independent "universes" can be created over τ-chain, that may or may not share τ's timestamping. They can also reference to each other, allowing code-reuse: recall that rules are code, since we have a unified language. Moreover: we give an ability to implement Decidable computer programs with RDF, namely DTLC languages rather Turing Complete ones. The implications of those languages will be described later on.

1.2. **Brief History: What and Why Do We Need to Decentralize?** Humanity is not wild, but engineered. It is, and always was, engineered and manipulated by relatively small groups of people. Our media, education, economy, law, politics, ethics, are all shaped mainly according to the views of small groups. Those groups tell us we are englightened, but in fact, the situation is very much of opposite nature. Immanuel Kant began his famous essay "What is Enlightment?", and he defines:

> "Enlightenment is man's emergence from his self-imposed nonage. Nonage is the inability to use one's own understanding without another's guidance. This nonage is self-imposed if its cause lies not in lack of understanding but in indecision and lack of courage to use one's own mind without another's guidance. Dare to know! (Sapere aude.) "Have the courage to use your own understanding," is therefore the motto of the enlightenment."

Those 1784 (1984?) words are in fact relevant for today more than ever: today we can break the chains of opression and let our own voice emerge.

About a century later, Franz Kafka described the inability of the individual to stand against irrational and moraless bureaucratic systems. While the laws are never actually given to the citizens, they still must obey them or otherwise suffer the wrath of society acting in violence against them, while the system contradicts itself: on one hand they claim they can never define law in a closed form, and on the other hand, they always find a way to justify their actions according to the law. Today, they also blame their computers and the records or decisions it allows to keep or modify. So maybe law cannot be formalized for us, but can be formalized for them?

Law can and should be formalized. The most important property a law system should have is a consistent ethical basis (e.g. constitution) and consistent implications from this basis to laws themselves. But who should formalize the laws?

Formalizing laws will pose even a greater danger if done by centralized hands. It is evident that democracy is incapable of assisting: the way in which voting is done today is far from assuring a consistent, moral, and functioning system.

Until recent times, this was indeed an inevitable situation. Now we have mathematical and technological ways to create laws collaboratively, while preserving on frames we set for ourselves, like consistency, votes, or minimal requirements.

Centralization of law is only one thing. What about centralization of information? Think of the following situation: Google has the most valuable information in the world, namely, "what people want". They do not give us access to their databases. But imagine, what if we could access their data: we would not only obtain this important information, but also have the ability to perform a much more sophisticated search. We could, for example, query about topics related to

a given topic, or automatically build new aggregated data from the database, and basically have endless additional uses.

Communication is also centralized. When communicating by electronic means, what we say is often intercepted by unintended parties. Our privacy is deeply vulnerable. We do not tend to have our own website and post our thoughts there, but we do it on centralized locations like Facebook. We do it because centralized hands give us better technology, with long term support and less bugs. We pay for it with our privacy, endless ads and marketing junk, and we even let them manipulate who our friends will be. Can we keep all benefits and have a moral high-end software?

1.3. **The Vision.** τ-chain's goal is to unite, yet keeping decentralization, humanity's knowledge and thinking, know-hows and communication, laws and opinions, all into one giant shared database that is able to be coherent and consistent, to be queried meaningfully, to reuse information/code/data efficiently, to allow all kind of social operations and communications to be done with no unwanted guests, to allow every user set their own specific rules and to be part of communities sharing the same thoughts, or goals, or needs. τ-chain does not enforce anyone to subscribe or follow one or another ruleset. But it lets all mental and technological benefits to be combined for the sake of better humanity.

Obviously, not all questions are solvable, some better solved by humans and some better solved by machines, and some are solvable with many machines and a lot of knowledge. Once one is able to solve a puzzle, namely: to prove a given claim, it gets into the network. Until then, it stays out.

This vision is not new, and probably began with the Semantic Web as we will describe on section 5, but could not be fulfilled without the nature of decentralization, which also gives the ability to fairly incentivize participants. Moreover, this vision in its decentralized version is not new as well, but no serious attempts were made before to make it happen.

It is about time to begin doing to many issues bothering humanity what Satoshi Nakamoto began to do to the monetary system.

It should be noted that τ-chain isn't a coin as for itself, and will bootstrap as a decentralized network, where developers are just like everyone, without any kind of so-called premine or any currency in hand, since the system itself has to be bootstrapped practically without rules.

2. Abbreviations and Definitions

- db: Database
- P2P: Peer-to-Peer
- prop[s]: Proposition[s] - a claim or definition to be understood under strict logic [2]
- DHT: Distributed Hash Tables
- DTLC: Dependently Typed Lambda Calculus
- TFPL: Totally Functional Programming Language
- RDF: Resource Description Framework
- N3: Notation3 Language

---

[2]"Cats are either flowers or mammals" is a true prop under pure logical interpretation.

3. OVERVIEW

We present τ-chain, a fully decentralized P2P network being a generalization of many centralized and decentralized P2P networks, including the Blockchain. We aim to generalize the concept as much as we find, and give users the ability to implement virtually any P2P network over τ-chain. Its interpretations, uses, and consequences are far from being a P2P network only, and include software development, legal, gaming, mathematics and sciences, logic, crypto-economies, social networks, rule-making, democracy and votes, software repositories (like decentralized Github+Appstore/Google Play), decentralized storage, software approval and verification, even "doing your homework in History or Math" in some sense (stronger sense that search engines), and many more aspects [3] .

3.1. **Five Equivalent Definitions.** τ-chain can be defined by several equivalent definitions:

- A shared db of rules, with a client that is able to change the rules, obey the rules, infer new rules from given rules, and make sure rules are consistent.
- A shared db of computer programs' code being collaboratively composed with revision control and custom permissions (like GIT), with a client that is able to run code, reuse code from existing programs, and verify programs against formal specifications which are programs themselves.
- A shared db of props, with a client that is able to state new props, prove props with custom derivations rules which are props themselves, and verify proofs to given props.
- A shared db of ontologies, which are definitions of types (taxonomies) and their relations, with a client that is able to propose new ontologies, make sure that ontologies are consistent, and query the ontologies db various queries.
- A decentralized Nomic game.

FIGURE 3.1. τ-chains
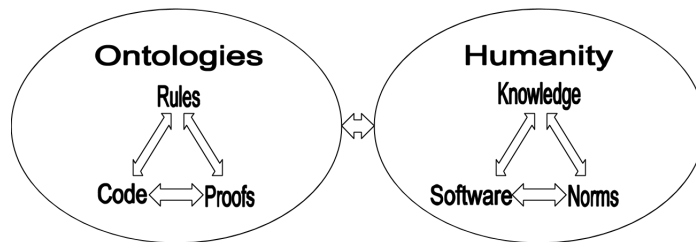
3.2. **The Essence of τ-chain node.** At its basis, τ-chain is nothing but a db of quads being tuples of four words each, namely: context, subject, predicate, object. On this note we show how rules, programs, proofs, and ontologies are all naturally representable by a unified way, and point to some of the far-reaching uses of such a system.

---

[3]Zennet Supercompter's design has now changed to work over τ-chain. BitAgoras will work over it as well.

Every node contains knowledge, at the very intelligent sense, namely the ability to make inference and implication using logic. They do not aim to represent a human intelligence, but their knowledge is first and foremost how to communicate and transfar liable information. Technically, this is can be done by formalizing with ontologies algorithms like DHT and Blockchain. But when looking at this in a unfied way as virtual creatures being able to ask rather retrieve, know rather store, talk rather comminucate, consider rather verify, having only mouth and ears and ontologies and a reasoner.

4. Basic Equivalence Relations

We give an informal explanation of rules as programs as proofs. For a formal derivations, we refer the reader to the literature of Type Theory.

By intuition, "running a program" can easily be seen as equivalent to "obeying the rules", while the rules are the code itself. The non-triviality lies on the other direction: can all rules be formalized as programs? The answer is positive, in a sense, and we will explain it through the proofs interpretation.

4.1. **What are Proofs?** From abstract logic point of view, a proof is a path taken from given hypotheses and axioms, and ends up with a result to be proved under the hypotheses, while the path has to be taken according to given derivation rules, and follows the notion of <u>implication</u> ($a \implies b$).

Example 1. Following are examples of derivation rules:

- Modus Ponens: if A implies B and A is true, then B is true:

$$((A \implies B) \wedge A) \implies B$$

- Cryptographic signature: it is impossible to generate a valid signature (up to assumed validation procedure) without access to the corresponding private keys.
- A mapping from data to its hash is one-to-one and noninvertible. Obviously, this is false under pure mathematical logic, but it is valid in cryptography being a practical science.
- Hilbert's formal inference rules and Gentzen's formal inference rules.

It should be noted that derivation rules can be stated as axioms, and axioms as hypotheses. Hence, a proof is inferring a statement from other statements, or equivalently, deriving a rule given other rules. Looking at definitions (taxonomy) as rules themselves, we see that rules are proofs and proofs are rules, and that proving is inferring.

4.2. **Proofs=Programs.** This non-trivial result is the celebrated Curry-Howard isomorphism . As there are many kinds of programs and many kinds of proofs, we are interested in a space where programs always halt. Roughly speaking, such programming language fall under the class of TFPL, e.g. Idris, AGDA, coq [4], which correspond to DTLC. Such and other class are isomorphic to the classes of proofs.

TFPL are not Turing complete. This lets us escape from many paradoxes arising from Turing machines formalism, promises us that programs always halt, and give

---

[4]Though COQ isn't really DTLC but Calculus of Constructions, yet those differenced doesn't really matter for our sake

FIGURE 4.1. Obeying Code
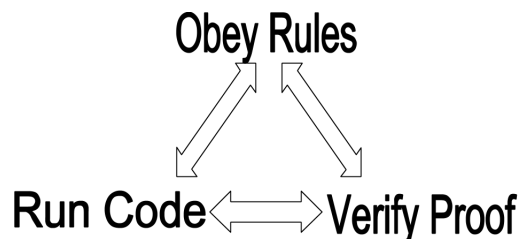
us strong abilities to claim and prove claims about the program [5] . By this, we can prove that a given code satisfies a given unit-test [6] . We can also prove the execution of the program [7].

It should be noted that practiaclly, not being Turing complete at this sense gives only advantages: any application one can come up with does not require Turing completeness, but DTLC is enough. Turing completene languages can do things that DTLC cannot only at extreme (mostly infinite) theoretical cases.

Corollary 2. *Logical proposition can be interpreted as rules, and vice versa. Proofs from axioms can be interpreted as inferring rules from other rules, and vice versa. Computer programs written in a totally functional programming language can be interpreted as constructive proofs, and vice versa.*

It should be stated that the correspondence between proofs and computer programs is so strong, that it gives rise to new foundation of mathematics, rather Cantor's/ZFC Set theory, namely: Category theory. It turns out that proofs and programs are isomorphic to category theory, at some sense. This is the Curry-Howard-Lambek isomorphism, aka Computational Trinitarianism [8] .

5. ONTOLOGIES AND THE SEMANTIC WEB

5.1. **Background.** This is really a long story, and we're going to make it very relatively short. It all began when the inventor of the World Wide Web, Tim Berners-Lee, came up with a vision that all data online will be machine readable, and build a web of relations between objects and types. That's the Semantic Web vision in a nutshell. For this goal, a vast amount of tools has been developed and still being developed.

Nowadays there's a huge and growing amount of ontologies from all aspects of life: basic web components and logical entities and relations (OWL), legal (cf. LKIF ontologies), software, medicine, e-commerce, cyptography, social media, geography, news, and many more [9][10]. Another notable project is *dbpedia*, that already formalized millions of concepts from Wikipedia with consistent ontologies. There is also a

---

[5]Sometimes like "the program is doing X and not doing Y in Z steps".

[6]Being a form of formal specification.

[7]cf. `http://amiller.github.io/lambda-auth/`

[8]`http://ncatlab.org/nlab/show/computational+trinitarianism`

[9]`http://www.w3.org/standards/semanticweb/ontology` `http://www.w3.org/wiki/Ontology_Dowsing#Lists_of_ontologies_and_services`

[10]Ontologies and search engines for ontologies from EulerGUI manual `http://eulergui.sourceforge.net/documentation.html##Finding`

vast amount of tools to manipulate RDF data, a list can be found on the Semantic Web Wiki[11] .

The same growth can be observed in mathematics and other sciences formalized for COQ, all naturally translatable to RDF, and containing thoushands of mathematical theorems with their full proofs.

5.2. **Semantic Web's Semantics.** The basic representation of concepts is standardized by RDF (Resource Description Framework). RDF is a language expressing Ontologies. An ontology contains definitions of types (taxonomy) like "dog is an animal", and relations between them like "all dogs has four legs". Of course, a definition of Four has to make sense as well, like the definition of "has".

Notation3 (N3) is a language makes it more convenient and human-readable to write RDF ontologies. It is logically powerful enough to represent DTLC. Moreover, it can always be converted into quads and vice versa, which is the natural format for τ-chains, as mentioned at the end of the introduction.

RDF being exactly pure logic can also be converted into English, in some limited sense, and vice versa. It is more machine-readable english than human-readable, but still very natural. We of course speak about the Attempto project.

5.3. **Quads.** τ-chain's storage contains nothing but a list of quads, i.e. four words being context, subject, predicate, object. Context gives separation between universes, since we obviously require to support distinct and non-mixing rulesets. So given the context we remain with a triple of subject, predicate, object. Triples can contain "all" possible knowledge, at the sense it can be proven that logical sentences (therefore also rules and computer programs) can be formulated this way and vice versa. To give a taste, let's write the sentence "the quick brown fox jumps over the lazy dog" using triples. To make it easier, we give names to objects. Let's call the fox Fred and the dog Dave. So:

```
Fred is fox.
Fred is quick.
Fred is brown.
Dave is dog.
Dave is lazy.
J is jump.
J over Dave.
J by Fred.
```

The last two sentences will more formally written as "Jump domain Fred. Jump range Dave" but of course any vocabulary is valid, since we do not concern about the actual meaning of the words but on their structural dependence, which itself gives meaning to them.

5.4. **State-of-the-Art.** EulerGUI is a veteran IDE for various reasoning engines, demonstrating the power of ontologies. Some of its supported formats are RDF, RDFS, N3, OWL, UML, SPARQL, Attempto (Controlled English), and even Java jars. It can output Java code that builds a UI according to an ontology, using Rule based SWING. It supports four different and powerful reasoning engines, logican queries, explained proofs, consistency checks, fuzzy logic, graphical visualization,

---

[11]http://semanticweb.org

and more. It can be integrated with the Deductions engine [12] which is itself written in ontologies. It is described on their website as "Artificial Intelligence techniques applied to common software tasks, using First Order Logic through N3 + OWL ontologies and rules". Both EulerGUI and Deductions are compatible with the powerful reasoning engine Drools. On Drools website one can find pros and cons of ontologies based development [13] . The reader is invited to get introduced to this world from the linked materials. A more recent tool is Protege, a professional open-source ontologies IDE.

5.5. **Ontologies of Rules.** Given we formalize our rules as an ontology. What are we going to do with them? We take an example tool called cwm[14]. We can ask it questions that are answerable from within the rules, even if not explicitly but by inference. We can also verify the consistency of rules [15] .

The reader is invited to take a look at its various tools[16], including the reasoner and the OWL Verbalizer (OWL is RDF with some basic ontologies defined, and is a W3C standard). See appendix for demonstrative screenshots.

5.6. **Programs as Ontologies.** DTLC based languages can implement almost everything, and in practical sense - everything. While Turing Complete languages suffer from undecidability, on DTLC languages we can look at the code as logic we can work with and prove various useful claims directly from the code, like that the program does not use the internet, or accesses only certain files, , show it halts, prove its execution path etc.

The transition from DTLC to RDF isn't trivial but two examples of how it can be done can be found at [14, 13]. The structure of the program (almost) doesn't change and namings can be preserved. The RDF format can be kept for being human readable, but for machine-proofs we do not stay on the RDF representation but use SMT solvers for reasoning.

6. Peer-to-Peer

6.1. **DHT.** As stated above, τ-chain can generalize any P2P network. A decentralized P2P client is a state machine that decides what to do given various inputs, according to some rules.

Example 3. DHT is an example of a P2P architecture. One of the products using it is BitTorrent. The most common flavor of DHT is Kademlia. Bitcoin's network is DHT itself, but with full replication per node, while on regular DHT the number of replicas can be controlled. A sample formal specification (namely: rules) of the Kademlia can be found at [11] . Note that this specification is given with a TFPL, hence can easily be translated into ontologies. We can see it has four primitives: Ping, Store, FindNode, and FindValue. The rules define what to do on each case.

---

[12]http://deductions.sourceforge.net

[13]http://docs.jboss.org/drools/release/5.4.0.Final/drools-expert-docs/html/ch01.html##d0e384

[14]http://www.w3.org/2000/10/swap/doc/cwm.html

[15]An important example for consistency check is, given a logical proof of some theorem represented as rules, verifying the proof is equivalent to veriying the rule's consistency.

[16]http://attempto.ifi.uzh.ch/site/tools

Example 4. Bitcoin is a decentralized P2P network having proofs as its main interpretation, while of course can be written as rules.

On τ-chain, every node stores three local ontologies: its routing table, its user's input, and the input from its peers. Those are combined with the ontologies this node subscribes to, from the shared db, together inferring what the client should do now, which can be local storage operations or sending information to peer(s). It is like: I tell you part of what I know, you think about it and tell me the conclusions you want to share with me.

Corollary 5. *Any P2P network is a ruleset defining what to send given what was received.*

A P2P network can set its own rules, and many contexts for many rules, and by this on-going implementing new networks over this one, while the user can decide on which contexts they want to participate. This is τ-chain.

6.2. **Temporal Logic: Blockchain.** How do we insert a notion of time into τ-chain? After all, useful programs or rules must be time-aware. Synchronization is even more a problem in a decentralized setting: how can P2P nodes reach concensus about timestamps? This hard problem was solved by Satoshi Nakamoto by inventing the Blockchain algorithm. Using the proof-of-work method can give meaning to time ticks as blocks. "Next moment" means next block. Ontologies can speak about time by interfacing temporal vocabulary to a blockchain, or even implement a blockchain as ontology.

The data is layered as a Merkle tree: the deepest level represent the DHT entries to be timestamped. The upper levels of the tree are formed in a block as a part of a blockchain. DHT items can therefore be timestamped by their hash.

7. τ-CHAIN

7.1. **Node Structure.** The client is essentialy a reasoner, like EYE, wrapped by a C++ program that asks it what to do. The wrapper exposes to the reasoner additional abilities of mainly networking, namely connecting and accepting connections, sending and receiving.

Events are written by the wrapper as an N3 ontology, then it calls the reasoner to unify the events against the rules. If the reasoner's output contains calls for action, the wrapper parses them and behaves accordingly.

By giving networking primitives to the ontologies, the reasoner can implement an HTTP server, and by this supply GUI to the users.

We begin with a C++ wrapper supplying primitives for DHT's syntax, while the semantics is up to the ontology.

7.2. **Bootstrapping.** The network will begin with the following "bare" client: it will contain an ontology that implements simple DHT, and will run actions by querying the ontology, using CWM, queries like "what should I do now", or more precisely, "what should I send to each of my peers" and "what should I do with my local storage". From this point we can implement everything by ontologies, in a collaborative work, over a decentralized network.

On this bootstrapping stage we plan to insert many readily-made and ontologies, and we plan to gather a round-table of professionals to set this network's global rules

FIGURE 7.1. τ-chain Structure
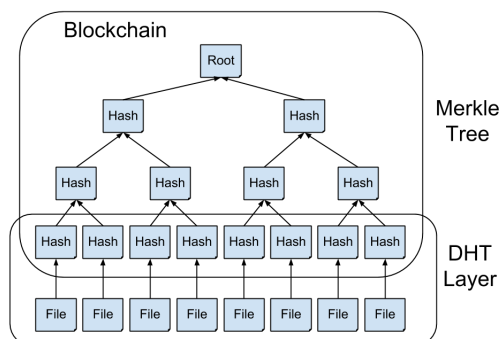
together. Here are some possible features to be implemented at the bootstrapping step:

- Rules to avoid malicious use of the network.
- Ability to create separate contexts, where each context has its own rules and they do not interfere. By this, users may create many programs and users can pick which programs they want to use. Like decentralized App-store/Google Play.
- Incentivizing every node for its work.
- Implementing Bitcoin as a proof of concept[17].
- Distributed large-scale storage, which is essential for the system itself, since it is planned to deal with large amount of data.
- Voting.
- How rules are going to be set from now on.

7.3. **Maturity.** Once the system is thought to have enough strength to go fully public and for everyday use, we can speak of what can be done over it then.

- Decentralized Source Repository (e.g. decentralized GITHUB).
- Decentralized Application Repository.
- Safely and automatically offer coins [18] to a human or machine proving a theorem or writing a software given formal specifications.
- Huge db of code fragments ready to be automatically reused by verifying formal requirements or by detecting isomorphisms [19] between ontologies.
- Any kind of votes for any purpose, like development team vote for a valid and authentic release, or decentralized democracy.
- Collaborative social/corporate rule-making, with the ability to find contradictions, to ask "what is missing in order to obtain X". And voting for those rules if wanted.

---

[17]Nothing intended to be used in real life. Of course, improved cryptocurrency is planned as further steps, especially at the scope of Agoras.

[18]τ-chain will not implement coins from day 1, but it is something that can and should be built upon the system.

[19]Those isomorphisms can connect ideas across science. It is possible that one writing some code that walks on graphs, will solve a problem that a biologist is dealing with.

- Fully-customizable, safe, decentralized and private social networks, as well as private clubs (even with membership fees, entrance test, acceptance rules etc.).
- Ask human-readable questions about virtually anything. Like "where did Aristotles live". CWM is able to answer such questions from ontologies, and specify a proof for its answer. Of course, other tools are both compatible and much more powerful, e.g. COQ.

Financial market over τ-chain will be built by us and will be called Agoras, implementing a cryptocurrency with an intelligent market of various services like banking, consulting, coding, or proving.

7.3.1. *Secure Software.*

7.3.2. *Code Reuse.*

7.3.3. *Shared Meaningful and Queriable Knowledge.*

8. Conclusions

We have shown how τ-chain is able to generalize any collaborative work, especially peer-to-peer networks. It provides ultimate information sharing capabilities, with rich ways to query data, infer new information, and act collaboratively. Existing P2P networks can be ported into it, e.g. Bitcoin's Blockchain, and make them being controlled by additional rules that can be changed on-the-fly with any rule for changing the rules. It may serve as a universal source of trustable information, as a collaborative source of knowledge, source code, and rules, in a form that is both machine and human readable and processable. It offers a ground in which sciences can be unified, and, more importantly, people's thoughts can be met and unified cleanly.

References

[1] "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008, Satoshi Nakamoto.
[2] "Intuitionistic Type Theory", Per Martin-Löf, 1980, `http://www.cip.ifi.lmu.de/~langeh/test/1984%20-%20Loef%20-%20Intuitionistic%20Type%20Theory.pdf`
[3] "Certified Programming with Dependent Types", Adam Chlipala, `http://adam.chlipala.net/cpdt/cpdt.pdf`
[4] "W3C RDF 1.1 Concepts and Abstract Syntax", W3C Recommendation, `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225`
[5] "RDF 1.1 Semantics", W3C Recommendation, 2014 `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225`
[6] "RDF Schema 1.1", W3C Recommendation, 2014 `http://www.w3.org/TR/2014/REC-rdf-schema-20140225`
[7] "Notation3 (N3): A readable RDF syntax" W3C Team Submission, 2014 `http://www.w3.org/TeamSubmission/n3`
[8] "The Understandability of OWL Statements in Controlled English, Semantic Web", Tobias Kuhn, 2013, `http://attempto.ifi.uzh.ch/site/pubs/papers/kuhn2013swj.pdf`
[9] "Programming in Martin-Löf's Type Theory", Nordström, Petersson and Smith, 1990, `http://www.cse.chalmers.se/research/group/logic/book/book.pdf`
[10] "Nomic: A Game of Self-Amendment", Peter Suber, 1990, `http://legacy.earlham.edu/~peters/writing/nomic.htm`
[11] "Formal specification of the Kademlia Routing table and the Kad Routing table in Maude", Pita and Camacho, 2012, `federwin.sip.ucm.es/sic/investigacion/publicaciones/pdfs/KademliaSpecificationSummary0212.pdf`
[12] "Category Theory for Scientists", Spivak, 2013, `http://math.mit.edu/~dspivak/teaching/sp13/CT4S--static.pdf`

[13] "A Type Theory for Typing, Meta-Typing, and Ontological Information in Discourse", Norihiro Ogata `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.3116&rep=rep1&type=pdf`

[14] "Towards a Conceptual Structure based on Type theory", Richard Dapoigny and Patrick Barlatier `http://ceur-ws.org/Vol-354/p63.pdf`