Tau-Chain and Agoras

Ohad Asor
(ohad@idni.org)
Editor: Prof. Dr. Avishy Carmi
(avishy@idni.org)

August 20, 2020

Draft for Community Review

Abstract

We describe the systems Tau-Chain and Agoras, the former being a peer-to-peer network which is fully and effectively defined by its users, and the latter being an economy built on those capabilities, facilitating economics of knowledge among other supporting aspects. Tau is a next-generation intelligent social network and discussion platform based on a newly proposed paradigm, termed here Human-Machine-Human Communication, offering a logic-based set of solutions to problems related to large-scale discussions, decision making, software development, artificial intelligence, philosophy of law, and more.

www.idni.org

Editor's Note

Intelligence is largely, perhaps entirely, a matter of appropriate selections. Intelligent beings occasionally seek to augment their potency in making profitable decisions in diverse settings, sometimes under weighty constrains. Nothing truly stands in the way of a sufficiently intelligent entity but Nature's limits on the manipulation and storage of information. This observation renders intelligence as the only valuable resource.

When one speaks of intelligence one often distinguishes between humans and machines. It seems that in each case intelligence bears a different meaning, and indeed, the ultimate goal of general artificial intelligence is to escalate machine intelligence to human level. Intelligence is a rather complex term, encompassing many categories pertaining to mental faculties and the intellect. One thing is, however, clear: it evolves with time. Here, we may ask two questions. The first one is why should intelligence at all evolve? and the second one is by what means. The answer to the first question may be that as all other processes in nature, so does intelligence strive to flourish by constantly confronting new conditions that carry the imperishable promise of new problems. The answer to the second question is that in most part intelligence evolves by means of communication between individuals, cultures, and species.

Humans have excelled in communications. They have developed a diverse language designed to discuss problems and their potential solutions. And yet it seems that the most pressing problems, those which concern large social constructs such as societies and nations, cannot be solved via discussions. If we follow the above reasoning the limits on the scale of discussions makes a major barrier in the course of human evolution.

Tau-chain, or Tau, is a system designed to ultimately solve the problems inherent in scaling up discussions. It does so by the aid of machines. This, however, comes with a significant cost. The machines in the Tau network should have access to meaning. In other words, humans and machines speak in languages that they both understand. This paper provides justifications why should formal logic be used in this kind of human-machine-human communication. Importantly, it postulates there is no single language adequate for all means and hence suggests a (Tau) meta-language, TML, designed for constructing languages and translators among them. This underlies the concept termed here *the internet of languages*.

Tau truly is a self-amending system governed by the process of social choice. On

Tau users discuss and decide on various issues and that includes Tau's own code. Tau hence evolves based on the collective decisions of its users. In the paper this aspect is paralleled with the process of legislation. Indeed, society constantly changes its rulers, its laws and behaviours, by casting votes as part of various electoral systems. The key difference from Tau is that in Tau users can equally discuss what to choose from rather than just choosing. Again, this is made possible as Tau is designed to retain those precious traits of small scale discussions, such as sharing opinions and choice about choice, on a large scale settings.

To deal with the paradoxes of self-reference, TML is designed having in mind the three laws of laws. They are characterizations of a formal logic of law, for as mentioned above the Tau process is akin to legislation. The three laws, decidability, closure under Boolean operations, and self-interpretation, furnish the coexistence of decidability and unrestricted recursion -- two highly desired properties in the process of legislation. They also point at specific logics that adhere to the three laws.

It does not end here. To sustain a social construct one needs an economy underneath. The knowledge formalized on Tau is suited for sustaining an economical infrastructure, a knowledge economy. To quote from the paper:

> *Formalizing knowledge in a machine-accessible format offers an advantage over books and search engines: the data isn't a stream of bits anymore but the meaning behind the bits, and therefore allows to perform an automatic semantic lookup of small pieces of knowledge even in a large compilation of writings.*

On Tau the knowledge economy is implemented via Agoras, a network aided with a cryptocurrency. Agoras will leverage Tau's knowledge representation and collaborative formalization features in order to allow knowledge owners and creators to offer it for paid usage.

The paper describes in depth these and plenty other features of Tau and Agoras.

Contents

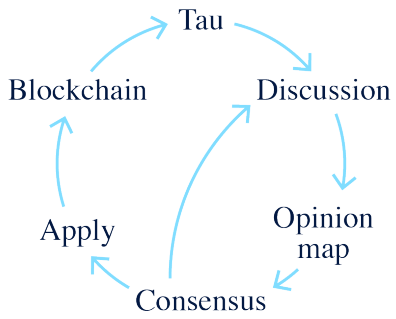
1 Introduction

Tau-chain, or simply Tau, is a decentralized blockchain network offering innovative social-choice methods in order to determine its own next version, and by that being controlled solely and effectively by its many users. Tau's users specify what they want the system to be like, and subsequently Tau auto-updates itself according to the users' consensus.

Software being updated and auto-updated is now a matter of every day. But who gets to decide how the next version will be like? Usually only a development team decides so, possibly taking into consideration the users' inputs, but how can we have software that is solely controlled by its many users? This is what Tau is about. Over Tau, the developers are no one else but the users themselves, in a way that can actually work on a large scale. Furthermore, on Tau, the software's requirements and specification are identified with its code. By that users don't need to describe the "how" but only the "what".

Our focus is how to make this work in practice on a large scale with many users. Achieving such a goal turns out to involve addressing certain difficult problems, in particular the problem of very poor scalability of discussions, avoidance of logical paradoxes, and impossibilities that typically arise from self-amendment. The proposed solutions are interesting in their own right and can be used on many other similar settings, e.g. for modern democracy, sound legislation, collaborative software development, and managing a large body of knowledge such as that which arises in academic research or large organizations. One crucial ingredient in this enterprise is the usage of *formal languages* as shall be described below.

Figure 1.1: Tau's Workflow



We argue that the process of users collaboratively deciding on Tau's next code is the same as the process of legislation. Laws have not only the aspect of just being laws but also the process of reaching an agreement for what the laws should be, and more importantly, how laws change with time. Tau is a process of legislation where the law is nothing but Tau's code. We shall therefore significantly focus on law and legislation,

and the reader may bear in mind that those processes are just another way of looking at the Tau process defined in the next paragraph.

We now give a succinct definition of Tau to be unpacked in the course of this paper: Consider a process, denoted by X, of [many] people, forming and following, by the way of discussion in formal languages, another process, denoted by Y. Tau is the case where X=Y.

To put it even more shortly: Tau, is a discussion about Tau. In particular, Tau is a machine-aided process of *understanding each other*.

Agoras is a network with a cryptocurrency[1] to be implemented over Tau allowing the following features:

- **Self-Definition**: Built over the Tau technology, Agoras is a software effectively controlled by its many users.
- **Knowledge Economy**: Leveraging Tau's knowledge representation and collaborative formalization features in order to allow knowledge owners and creators to offer it for paid usage, as well as for knowledge seekers to efficiently locate and purchase knowledge.
- **Computational Resources Market**: The ability to rent and rent-out forms of computational resources (CPU, disk, etc) allowing large scale computations to take place under cost and computational efficiency.
- **Derivatives Market**: Agoras will have a no-middleman peer-to-peer derivatives market and by that implement risk-free interest without any creation of new coins, using the concept of zero-delta portfolios.

It should be remarked that the design presented here subsumes an old design presented at [11] which was found to be mistaken, as well as neglecting vital aspects related to self-amending systems e.g. how to scale up social choice while avoiding self-amendment paradoxes. The old design has to be completely discarded and there's no benefit for the reader in trying to complement the picture by revisiting it. Additionally, the computational resources market is different from Zennet's which is described in [10], for the surrounding capabilities (mainly logic, contracts, and proofs) give rise to a more sophisticated design.

[1] In contrast to Tau which has no currency or any other monetary aspect.

Another remark is that this paper sometimes refers to Tau and Agoras in a present tense, e.g. "Agoras does...", but Tau and Agoras are still not ready as for the time of writing these lines.

This paper is organized as follows. In the first section we discuss Tau. First, starting from presenting the paradigm of human-machine-human communication and the main features of the system, as well as how to scale discussions. Then we continue with aspects of decision making. Afterwards we speak about collaborative software development, and then we move to describe our main theoretical contribution being the laws of laws and logics that admit them. We conclude with the internet of languages.

The second section is about Agoras, first describing contracts and then moving on to economics of knowledge, and concluding with computational resources market and derivatives market.

On the last section we present a taste of the long-term impact of the system.

Five appendices follow: The first is a summary of the main problems and questions answered on this paper. The second describes a bit of classical choice theory and compares it to our setting. The third corresponds with Suber's view on self-amendment. The fourth sketches the P-DATALOG langauge, and the fifth presents a proof that P-DATALOG with evaluation operator collapses to P-DATALOG.

2 Tau-Chain

2.1 Human-Machine-Human Communication

How can a very large group of people reach a collective decision? Tau offers a new paradigm to address this long-standing problem, a paradigm which we refer to as *Human-Machine-Human Communication* that solves this problem to a very large and generic extent. Tau is therefore useful for all kinds of collaborative theory formation and decision making and not only regarding its own code, as we shall emphasize later on. We can even argue that our solution is a complete resolution of the problem: x10 more participants will indeed yield at least x10 more outcome, in sharp contrast to how discussions currently scale[2]. This set of solutions is rooted in two concepts which will be explained shortly: the first is the usage of *machine-comprehensible* languages, and the second is the *ability to maintain small scale aspects* of discussions and decision making on a large scale. But before we touch those two points let us discuss a bit legislation and voting.

When considering a large scale collaborative effort of legislation (or any other theory formation), voting commonly comes to mind. Indeed legislation of state laws typically involves forms of voting. However there is a certain major scaling limitation related to voting: even though it's possible to let everyone an effective and equal right to vote, it is still impossible to let everyone an effective and equal right to propose what to vote over, since even if everyone got an equal right to propose, how one is even going to read millions of proposals a day? This makes the equal proposing right ineffective. We conclude that *voting cannot effectively scale and stay fair at the same time*.

Furthermore, this difficulty of discussion scaling holds even if we assume that all

[2]Discussions scale even more poorly than other settings arising in economics. Typically economical settings scale under the "Law of Diminishing Returns": twice more workers typically don't yield twice the outcome, but less. This is commonly formalized by stating that the second derivative of the utility function is negative, or in other words, convexity. This is one of the main reasons that convexity plays an important role in economics, and one of the main assumptions on *any* economical utility function is convexity. However discussions scale much worse: their utility (e.g. the amount of knowledge, or agreements, becoming accessible and useful to the participants) does not merely exhibit a negative second derivative w.r.t. the number of participants, but commonly even a negative first derivative, since having too many participants in a discussion commonly reduces the outcome, not only not increasing it proportionally.

Note that differentiability and convexity here are in a generalized sense: for discrete variables, taking the differences would serve the same goals and arguments for the law of diminishing returns.

participants are polite, smart, open minded, love each other, and behave perfectly. It should be noted that other projects that deal with helping discussions indeed tend to deal with managing the absence of such reality, of everyone being "perfect". However we deal with a much more fundamental bottleneck which remains in-place even if the "perfectness" assumption holds.

In everyday life, large scale decision making is commonly addressed by creating hierarchies of decision makers, who in particular get to decide what people may vote over. But then the vast amounts of knowledge and the varying preferences of the voters are not reflected in the proposals, because, in the better case, the data fails to propagate up the hierarchies due to information-handling bottlenecks, preventing humans from processing large amounts of information in *any* form[3]. Voting is therefore carried out over alternatives almost void of truly relevant considerations.

In small groups we rarely even consider voting. People simply say what they have to say and this by itself is enough for the participants to understand each other's opinion and where they agree or disagree with each other. The participants don't need to vote because they already know what the voting result would be, since all the information is already presented, implicitly or explicitly, in what they say. Put it differently, the *opinion map* naturally arises from things said during *discussion*. How can we achieve this in the larger scale, in which we have a discussion among a large amount of participants yet we are still able to recover the opinion map?

This is made possible with the help of machines, but it comes with a significant cost: participants in a discussion will have to speak in a language that is understood by machines[4]. When this is the case, the machine is then able to calculate the opinion map (and the agreements/disagreements/consensus). We refer to such languages as formal languages, and more specifically, we consider languages which are, or translatable into, *logical formulas*. More on the language aspect of Tau will be emphasized on subsections 2.5 and 2.6, but we first move on to describe other elements of the system.

[3]Let alone other bottlenecks which may or may not be solved by imposing certain flows and behaviors.

[4]In fact not all such languages are adequate for this case, and we shall emphasize later on about the logical aspects required in order to make this work.

2.2 Large-Scale Discussion Platform

Maintaining aspects of small-scale discussions in the large scale, we'd like users to simply say their opinions without any need to organize them, yet all viewpoints and the relations between them should be automatically inferred.

Tau takes the form of a discussion platform and a social network similar to existing ones, with friends and posts and comments. The main difference is that users write in formal languages and further aspects derived from that. The main components of the system are therefore:

- **Discussions:** Ability to have discussions using formal languages. Discussions include opinions, questions, and answers. Discussions may also be implicit: sphoradic posts and comment across the network, yet about the same subject, may be counted as such.
- **Worldviews:** Being user profiles and representing the totality of all the user's opinions expressed over the system.
- **Teams:** A group of users collaborating towards a common goal.
- **Permissions:** Allow only certain groups of people (e.g. friends) to have access to the users' worldview or parts of it. Similarly for teams.
- **Query:** Users are able to run semantic queries over worldviews and discussions, and virtually any data present on the platform. This amounts to machines answering questions in a precise way.
- **Agree/Disagree:** Users are able to click "agree" or "disagree" on other users' posts. This is considered as merely a shorthand as if the user posted the same opinion (or its negation, respectively). Similarly, users may agree/disagree on which question is interesting, see below for questions vs. answers over the system.
- **Trust:** Automatically agree with certain users about certain topics.
- **Consensus:** Calculate what all or most of the discussion's participants agree on. Users are able to configure much broader definitions of what "consensus" means.

- **Opinion Map:** A more detailed report of all opinions expressed in a discussion, ordered by implication in the logical sense[5]. Observe that although we can expect a discussion of four people to contain four different opinions, we do not expect a discussion of a million people to contain a million different opinions, but many participants will hold the same (or overlapping) opinions and/or agree with each other.

- **Contradiction Resolution:** When a contradiction arises inside a worldview or between users (whether in the same discussion or not), the system will report the contradiction and offer ways to resolve it.

- **Autocomment:** Tau will be able to automatically comment on topics, optionally with the user's review and approval, based on previous posts from the same user regarding that topic.[6]

- **Synthesize:** In the case of a discussion about a specification of some desired software[7], to synthesize code that meets the consensus over this specification.

In addition we have the following features which arise from the Internet of Languages, a design allowing to define new languages and translate between them, discussed in detail at 2.6:

- **Add Language:** Support a new language over the network, or a new version of an existing language, by submitting a translator guiding how to translate documents in that language into an already-existing language.

- **Translate:** Translate (compile) a document from one language defined over the Internet of Languages, into another.

2.2.1 Worldviews and Teams

A user's worldview is the collection of all opinions expressed by that user in the form of posts and discussions and agreeing/disagreeing. Similarly, a worldview contains which

[5]e.g. "X and Y" implies both X and Y, or in other words, "X intersect Y" is a subset of both X and Y, or, "A is a special case of B".

[6]Tau will never guess your opinion, not even an educated guess. This demonstrates both the power and the necessity of logic-based AI in contrast to machine learning which is probabilistic in nature.

[7]Including the specification of the system itself.

questions the user has marked as "interesting"[8]. Users can then query not only their own worldview but also other users' worldview, provided that they gave permissions for other users to view their worldview or part of it. By that users can find other users with similar ideas or common interesting questions. One can only imagine the implications of this for recruiting, research, business partnerships, and even dating.

Teams are groups of people with a common cause or common questions to discuss and answer. A team creator may arbitrarily choose its initial rules. One special team would be the one that determines the next Tau code. Who will participate in this team and how, is left for this very team to decide, as we shall describe more later on.

2.2.2 Truths vs. Opinions

It should be noted that, inevitably, Tau has no access to truths, only to opinions. One might argue whether humans have access to truths, and we're not going to get into this point, we only state the obvious that machines have no access to what we intuitively consider as physical truth[9]. Tau has no way to verify the correctness of statements like "the Sun is made of hydrogen" or "the Sun is made of cheese"[10]. By that, Tau can never say who is right and which opinion is correct[11]. However it can infer certain kinds of truths and falsehoods in the case of tautologies and contradictions. A contradictory opinion is always false, and is a case where we can indeed point to a physical falsehood. The equivalent holds for tautologies.

The more opinions stated about a certain topic, the higher the probability that a contradiction will arise[12]. We can therefore approximate the truth by the way of elaboration, and choose to accept certain opinions as truths or falsehoods by the way of debate, challenging each side to give more statements on the subject and by that

[8]The nature of questions over Tau will be discussed below.

[9]In contrast to mathematical truths, which are no different than tautologies, and by that say nothing nontrivial about the "real world" other than "a contradiction may never exist".

[10]Even if we equip machines with sensors, e.g. camera and microphone, it is always possible to "fool" it by letting it see any predetermined inputs, and it will never know whether its sensory inputs are "real", "fake", or "compromised".

[11]In particular, Tau should not be seen as a tool for avoiding "fake news", except in special cases as we explain right away. We mention "fake news" because a question commonly arising among people hearing about Tau and its logic and truth aspects, is whether Tau can solve this problem, and the answer is mostly negative indeed.

[12]In fact, this probability increases in an exponential manner because of the multiplicative nature of conjunction.

increase the probability of contradiction for each side. This can also have a monetary aspect over Agoras.

2.2.3 Questions vs. Answers

We start with philosophical observations regarding questions vs. answers. We have the notion of a correct answer, and we can even program computers to tell whether a given answer to a given question, is correct. We are interested in correct answers to our questions.

Just like answers may be correct or incorrect, at the same way questions may be interesting or not. How can we tell which question is interesting? This is not a question which machines can answer, as this is purely a human thing.

Questions being interesting is not only subjective, but inherently always stems from the logically-arbitrary preferences of the asker[13]. There isn't such a thing as "an interesting question", but "a question interesting for certain beings in certain times". Our questions come from our human nature and our personal nature. Our personality is defined by the questions we find interesting, so much more than by the answers we give. Similarly, cooperation between people interested in the same questions makes much more sense compared to cooperation between people agreeing on the same answers. Many would be much more interested in finding people asking the same questions as them than finding people giving same answers.

Questions also guide some theoretical considerations in knowledge representation, e.g. the setting of open world assumption vs. the one of closed world assumption[14], since a question may be seen as a query under the open world assumption, while answers are more related to the closed world assumption. Questions are therefore a major aspect of Tau.

For clarity (and for this subsection only) we can distinguish between "questions" and "queries". By "queries" we refer to questions in which we expect to have an immediate answer to, e.g. the case where we feed a machine with information and then query that

[13]To get convinced about the arbitrariness and non-logicalness of our desires, one can acknowledge that we don't even know what we'll want to eat for dinner. No logical formula can entail that. It has nothing to do with the nature of truth but with the human nature.

[14]Open/Closed world assumptions are fundamental notions in the fields of databases and knowledge representation.

information. The machine should not return any new information, it will only use the information that we gave it[15].

In contrast, by "questions" we refer to questions in which we don't expect to have an available answer yet. A question is a tool to define which knowledge is desired. Questions usually come chronologically before the knowledge, not the other way around, and even in case they don't, that knowledge would frequently be ignored and dismissed as "not interesting". Questions are a tool to scope a discussion or an exploration into certain areas of knowledge. A tool which the machine will never be able to simulate, but only given the human input of which questions are interesting, it can greatly help us in finding correct answers by organizing discussions about those questions and by performing various automatic inference tasks, in particular, locating correct answers to interesting questions disregarding who or how many gave the answer, that, across the whole network, up to visibility permissions set by the users. By that no good answers and ideas will be wasted and neglected anymore.

The fact that questions do not have a "truth value" similar to answers, but "interest value"[16], guides us to conclude with the role of the people vs. the role of machines over Tau after becoming mature with enough formalized knowledge: humans are for questions, and machines are for answers. More broadly, we see this as a philosophical truth that should guide any AI aspirations.

2.2.4 Understanding Each Other

Over Tau the machine is not an equal part in the conversation: it is only a machine. It merely organizes what we say and is able to do so since we encode our information in a way accessible to it. A user can broadcast an idea to another user, and at this narrow scope of communicating an idea between people we can already enjoy three benefits: easy explaining, easy understanding, and formalizing knowledge as a byproduct. We justify this claim heuristically by first presenting a working definition for "understand-

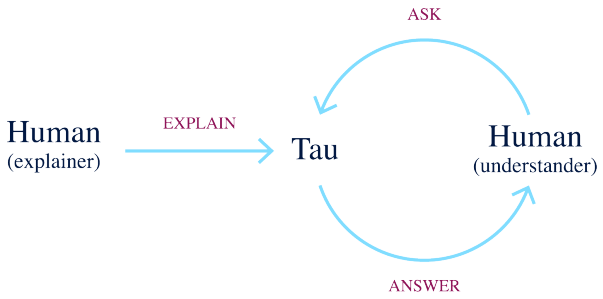
[15]This in sharp contrast to machine-learning kind of AI. Tau is about logical AI, while machine learning is about statistics under zero knowledge about the measured population, and therefore frequently returns wrong answers even for simple questions like addition of integers. There's much more to say about the relatively-little expressiveness of machine learning compared to logic. From a descriptive complexity point of view, machine learning is known to lie in the complexity class P, while common logics very frequently subsume this class by far.

[16]Giving rise to ordering of questions. One correct answer is not "more correct" than other correct answer, but one question may be more interesting than other interesting question.

ing": fully understanding a point means the theoretical ability to answer all questions relevant to that point[17].

Over Tau the explainer doesn't need to make other users understand (the target users), but only making the machine understand. This task might be simpler on some aspects and more complex on others, as machines are certainly less bound to organization and scale than humans and further can assist the formalization process, but on the other hand machines have a language barrier aside their inaccessibility to the human nature. Having achieved an idea formalized in a formal language, the target users can now not only translate it to other knowledge representation languages, or to organize it as they see fit, or to compare it to other formalized ideas, but they can also ask the machine all questions they got. Since the machine understood the subject completely by our definition of "understanding" above, it can help the target users understand it as well because it can answer all the users' questions, without the need to refer the question to the original idea's author. Tau is therefore a trusted ambassador of ideas.

Figure 2.1: Understanding Each Other



2.3 Large-Scale Decentralized Social Choice

2.3.1 Overview of the Self-Amendment Process

We now sketch the flow of Tau as a self-amending platform. Users discuss over Tau, using formal languages that admit the Laws of Laws (defined below), what should be Tau's next version. Tau then calculates the possible alternatives for its next version. On 2.3.2 and 2.3.3 we will deal with which alternative from those to choose. Those alternatives, whose aspects are discussed below, are next-block candidates for Tau's blockchain. Once an alternative is chosen and added to the blockchain, Tau then downloads that next block and invokes itself under its new definition. As we shall see,

[17] In particular, due to Gödel's incompleteness, we can safely claim that no one will ever fully understand arithmetic.

our logic of choice will include a self-interpreter and built-in quoting and evaluation operators, and by that Tau need not replace its code and externally run it, but it can simply call the evaluation operator over the quoted code inside the blockchain and by that replay the blockchain from the genesis block to its current block, resulting, among other desired outcomes, with a proof-carrying execution that justifies the whole blockchain's history up to the last block[18].

Furthermore, a special step between consensus and the next version is introdced, call it "application". After users have said their opinion and a consensus is reached, the change doesn't occur right away, but the users have to also agree to apply the change, and if they don't, the discussion continues and may or may not reach a different consensus. This "agree to apply" step is used as a safeguard and as another opportunity to review the consensus before it's being applied. The reason is that the consensus is always floating - it is the statements that all users agree on at a certain point in time. If a consensus is reached, it doesn't mean it'll stay there for the next minute. We therefore have to consciously accept the consensus in the form that it happens to emerge at. Moreover, since the consensus is a nontrivial aggregate of many people's opinions, it might be that the users see the proposed consensus (and hence the proposed next version) for the first time. They'll then need to review it and to agree that the complete picture is desirable indeed.

In addition, a consensus may not be decisive in the sense that it does not entail a unique action for each state of the program, and by that there's no way to execute it as a program. Refining the consensus is therefore a vital aspect in the process.

Finally we remark that users are able to configure many more tests and conditions in the application stage before accepting a consensus, especially as a filter against deadlocks as discussed on the next section. We mention here one possibility: one way to weigh a set of rules against another, in case where agreements on each are of the same strength (e.g. 50-50), is to consider which questions that are marked as "interesting" are being answered by each set of rules, and the one which answers more interesting questions is favored.

[18] A point which Suber's Nomic does not support without extra-logical components as emphasized in the subsection "Tau vs. Nomic".

2.3.2 Choice Dilemmas

"Should two courses be judged equal, then the will cannot break the deadlock, all it can do is to suspend judgment until the circumstances change, and the right course of action is clear."

— Jean Buridan, c. 1340

We explained how Tau is nothing but a software that is effectively determined by its users. For that to happen users will have to engage in large scale discussions about what Tau's next code is required to do. Subsequently, the consensus will be automatically calculated and all clients will be updated with the new specification of what Tau is defined to be at that point of time. But which kind of consensus should be used? Unanimity, majority, or any other?

The main technical problem with majority vote in a decentralized software network is the inability to detect multiple votes by the same person. This can be circumvented if we soften the requirement of decentralization by requiring to give some amount of trust and voting power to some people. Another method would be to let users prove their uniqueness of identity via centralized 3rd parties ("electronic notaries").

Unanimity also raises a security risk in a decentralized setting because a participant may, honestly or dishonestly, prevent the whole network from reaching consensus, and putting the network in the dangerous Buridan's Mule situation.

We have mentioned above a mechanism for contradiction-resolution, as well as debates, and testing which questions that are marked as "interesting" were answered. Those are not fully decisive methods. In what follows we therefore assume the case where the contradiction[19] was not resolved.

One rule of thumb is to not limit a user from doing anything that doesn't affect other users. If some group of people wants the system to behave in a certain way, and another group wants it behave in a different way, we can then check whether it's possible that both groups' preferences coexist and each gets a different view of the system. But indeed it is not always possible to make the two changes coexist as they might contain

[19] Ambiguity is referred to as a contradiction here, because the requirement of being a well-formed program or function is in contradiction to the case where we have non-unique actions for each state of the program.

some contradiction on the very foundation of the system, which may turn the system incompatible between the groups.

One way to deal with the case of two (or more) contradicting alternatives which cannot compatibly coexist, is to require the groups representing each alternative to compute hashes of their proposal (mining) until the hash is small enough. This gives preference to users with more computational (or other mining) resources, and is the Bitcoin's mechanism: the chain longer in terms of computational power, is the one preferred by the network. Many similar blockchain-based algorithms are possible each with its own advantages and disadvantages. A similar way is to simply randomly choose between the alternatives. This is of course the solution that we wish to avoid, however in a pure Buridan's Mule situation, randomness is the only way out[20].

The subject we touch here is actually the laws of changing the laws, a topic of central importance to be addressed in the next section. Many other ways of calculating the consensus are possible, none of which is perfect. It therefore requires a discussion among the users in which they raise possibilities and highlighting their advantages and disadvantages, evolving into an initial community-backed set of rules. We therefore suggest starting the discussion about Tau in a semi-centralized setting (in contrast to a blockchain network) and by that letting the users decide the initial governance mechanism of the decentralized network, thus temporarily circumventing the disastrous consequences of a Buridan's Mule situation.

2.3.3 Blockchain

As for the time of writing these lines it is important to explain what blockchains are for, since currently the world is largely full with misconceptions for what blockchains can do which other best-practices and cryptographic primitives cannot do. The answer is that blockchain solves the problem of decentralized timestamping. Many forms of securing data can be done without any blockchain but with the use of public key infrastructure, hashes, and other cryptographic means. However the problem of securing an ordering of events in time was shown to be unsolvable in an environment that does not assume trust, cf. the Byzantine Generals Problem.

Satoshi Nakamoto [8] was the first to come up with an approximate solution which

[20]Waiting is also a possible solution as proposed by Buridan, but is not guaranteed to work.

is based on cryptographic assumptions. This solution is very expensive (and not only in financial terms) and therefore is highly unrecommended for cases where decentralized timestamp is simply not required. However, it is required at the scope of Tau.

Tau's blockchain is a ledger of its own code. Each block contains the current Tau code and the next block contains the next code, in a fashion similar to "auto-update"[21]. The time-ordering of those codes is vital: if clients are not aligned regarding which code version to choose, they may break down the ability to even communicate over the network with one another, e.g. in case that the network protocol has been modifided in a non-backward-compatible way.

One might ask: which kind of blockchain algorithm will Tau use? And the answer is trivial: it'll use whatever its users want it to use. If one day a better decentralized timestamp algorithm will be invented, it can then be incorporated in the next auto-update.

2.4 Collaborative Software Development

Tau has four main guidelines relevant for collaborative software development: *correct-by-construction* software, *knowledge-oriented* programming, *consensus-oriented* programming, and *code reuse*.

Correct-by-construction software is obtained by the use of declarative programming, a paradigm where programmers define only the "what" and not the "how". As an example, suppose we have a body of code and we would like to add additional constraints to it, e.g. "never send over the network data which is marked as private". In non-declarative programming the programmer will have to analyze the control flow of the code and see where data is sent, and then make sure that the data is not private. This, of course, opens the door to a lot of human mistakes[22].

But what if we could just append a clause to the program expressing the above constraint? In case where that constraint is well-defined, it should be possible for a machine that gets to understand it to track the program's flow and prevent sending private data. For this we'll need the mathematical possibiliy of reasoning over programs, hence decidability of the language over such tasks is required.

[21] This is the process of "changing the law", as we mentioned above, and will emphasize more below.

[22] The amount of human mistakes and bugs during software development, is so huge, that arguably non-programmers highly underestimate them.

We therefore observe that declarative programming largely removes the need of taking care of the program's control flow, and we can focus on the program's *specification* rather *implementation*. The field of taking a program's specification and converting it into an implementation is called Software Synthesis[23]. Programmers therefore need to care only about the specification, and the rest can and should be automatic.

This specification, or in other words: the software's requirements, is mostly not defined by the programmer, but by the programmer's client. The clients need to say which software they want, and programmers code them. Once, by declarative programming, we identify the requirements with the code, the borders between programmers and users (or entities that order programming work from programmers), are blurred. However, a specification has to be *complete* in the sense that the machine can always deduce the unique action to perform next. To go back to our example, we will need a complete specification of what "private data" and "send over the network" means.

This brings us to knowledge-oriented programming. Over Tau a large amount of knowledge will be formalized, either directly for the sake of having a body of formalizing knowledge, or as a byproduct of discussions. Once enough formalized knowledge becomes available, even non-programmers will be able to say "I'd like this software to be secure, by this and that definition of security which I adapted from an expert that I trust", and a software will simply become secure by the mere utterance of this sentence.

We now move to the collaborative aspect which is consensus-oriented programming. This is quite straight forward given everything we explained so far: people will only need to state what they want the software to do, and their consensus is the agreed specification. This helps not only to agree with each other but also for different people to take responsibility of different (or even same) aspects of the software without stepping on each other's toes, a painful and expensive difficulty in real life software development.

Code reuse is also a very useful concept. Virtually all common algorithms were already implemented many many times. This is because using existing code comes with a lot of difficulties, so very frequently it's simply easier and cheaper to rewrite

[23]cf. page 11. Also observe that given a full specification, one can execute the program by querying at each step for what should be the next step, and by that execute it without the need for any conversion into code. This approach may work very slowly in practice, to an impractical extent. Synthesis is therefore about a certain kind of optimization. That said, there may be more aspects for such optimizations that may not involve code synthesis methods.

the code[24]. But since we reduce the coding task into that of knowledge representation, knowledge may then be very easily reused. No one will have to define the same thing twice anymore, and no integration efforts will be required anymore.

In addition, and this is relevant not only to code but also to knowledge formalization in general, we obtain a characteristic which is so good to the extent that it is an economical anomaly: the law of diminishing returns becomes the law of increasing returns, because writing new code or formalizing new knowledge becomes easier as more code and knoweldge already exist in the system.

2.5 Logics for Laws

"Before the law sits a gatekeeper."

– Franz Kafka, "Before the Law", 1915

We live in a world in which no one knows the law. Except trivial cases, you cannot know what is legal and what is not, all you can do is try and see what is the judge's or policeman's opinion after you've taken your actions, an opinion which of course differs from one case to another. Or you can consult a lawyer that will tell you that there are no absolute answers, and at the end of the day it's a matter of probability which unfortunately no one knows how to calculate. You can do your best to have a productive life and follow the rules as you understand them or as lawyers guide you, but no one can guarantee that you will not be considered a criminal, or that legal actions will not be taken against you. Similarly one can live a life of harming so many people and no legal system will stop it even if the system is aware of the actions taken. Such pessimistic situation is not new and is not local, and to my taste was best described by Franz Kafka.

Playing with words and taking them into any desired direction, consciously or not, with good or bad intentions, was always there since humanity acquired language skills. The worst lies contain only truths, and the worst crimes are licensed, and arguments can be given to justify almost anything. This "crisis of truth" is the foundation of the post-modern stream in philosophy, notably the Deconstructivist approach which

[24]Yet another peculiar fact in software development which non-programmers might not fully appreciate. Indeed rewrite is most commonly cheaper than maintaining existing code, if written by different people, or even by the same person after some time, a time which is usually surprisingly short.

demonstrates how texts may be interpreted in many contradicting ways. "There is no one truth" is the basis of post-modernism. But can we at least have some island of truth in which social contracts can be useful and make sense?

In what follows we derive a logical setting for laws and legislation and thus propose a sound way of implementing laws and legislation. Recall that this applies not only to laws in general but also to Tau's own code.

2.5.1 The Laws of Laws

For our purposes we define a law to be a function that takes a description of a situation and returns either "legal", "illegal", or "undefined"[25]. This might be surprising: how can something nontrivial be said about laws in such a generality that does not assume anything further about the nature or purpose or scope of those laws? We derive nontrivial statements indeed by considering the effectiveness of the representation of law, and the case of changing the law.

Moreover, we say absolutly nothing about the law itself, but only ask which language can be considred as adequate for serving as a language for law, or more accurately, since the language specifics don't matter, we seek for a logic that may soundly represent laws. We present three requirements from a logic for law (the "three laws of laws") which will lead us to a certain logical formalism.

We seek for a class of expressions $\mathcal{L}$ that is suitable for representing laws, and we now list some requirements from $\mathcal{L}$[26]. The first two requirements are very natural but pose a significant restriction on the required language:

- **Decidability**: It shouldn't take infinite time in order to answer whether something is legal or not. We require to always have the ability to get answers, and

[25]The necessity of the "undefined" value will be clear below. "Undefined" does not mean "unspecified". It is more like a machine that never halts and therefore has no return value at all, and all machines that may call it will also never halt therefore also take the status of "undefined". Another way to look at it: division by zero is not "unknown", but is truly undefined. Any formula that contains division by zero, is not true or false, but is simply undefined.

[26]We shall begin with treating $\mathcal{L}$ as a set of Turing machines, a view which is justified by the decidability requirement below, however, equivalently, $\mathcal{L}$ may also be considered as a programming language (and the set of Turing machines it represents is the set of all programs on that language), or, as a decidable (hence computable) logic. Our goal is to find a logical language that will capture this set of machines.

in particular, within finite time. This leads us to consider $\mathcal{L}$ as a *set of Turing machines*.

- **Closure under Boolean Operations:** The language should be closed under if-then-else with equality-to-zero conditionals, so if f, g, h are programs in $\mathcal{L}$, then so is "if $f(x) = 0$ then $g(x)$ else $h(x)$". In other words, $\mathcal{L}$ should be closed under union, intersection, and complementation[27].

The next requirement arises when considering the case of changing the law. Suppose we have a law and now we'd like to allow modifying it over time. For that we better come up with laws of changing the laws, as otherwise, the law may be changed under no condition whatsoever, rendering it meaningless and useless, since in order to break the law one only needs to first change it, and in the absence of laws of changing the laws nothing would prevent one from doing so.

We therefore need not only laws but laws of changing the laws. But then we'll also need laws of changing the laws of changing the laws, as otherwise one may change the laws of changing the laws, and by that be able to change the law, again rendering the law useless and meaningless.

Is it therefore the case that we must have infinitely many laws (laws of changing the laws of changing the laws of changing... ad infinitum), and if not, a dictator lawmaker which can change the law into anything any time, just to begin with the simplest law? As we shall see right away, there's a way out of this conundrum.

Consider the following law: "all laws, including this one, can be changed given majority vote". Is it a law, or law of changing the law, or law of changing the law of changing the law? Well, it is all at once, because it refers to itself. It refers to how this law itself may be changed, and by that it's a law of changing the law ad infinitum indeed. It is important to note that this law makes perfect sense and raises no paradoxes, a concern that commonly arises in self-referential statements[28], cf. also Appendix C.

[27] Another way to state it: the language of law should allow the use of the logical connectives "and", "or", and "not".

[28] Self-referential definitions have an unjustified negative reputation. If we open a dictionary and see a definition of some word where the definition contains and relies on the very same word, one might commonly dismiss this definition as either contradictory or meaningless. However we know from the theory of computability and mathematical logic that sometimes recursive definitions don't only make sense, but are sometimes essential. Consider for example the definition of the factorial function.

We were able to obtain laws of changing the laws ad infinitum in a single finite sentence only because we used self-reference, or in other words, recursion. We have just demonstrated that if the law doesn't refer to itself then we'll never be able to soundly protect the law from undesired change since we'll need infinitely many laws indeed, as above.

In fact we require from the logic of law more than just recursion: we actually require self-interpretation[29]. A language supporting self-interpretation must also support unrestricted recursion[30]. Self-interpretation is required because the current law must be able to interpret a proposed law in order to have access to its semantics, and then decide whether this proposed law is accepted or rejected.

This leads us to formalize the third requirement from the laws of laws:

- **Self-Interpretation**: $\mathcal{L}$ should contain a self-interpreter, which in particular implies that $\mathcal{L}$ is closed under recursion and self-reference in an unrestricted way [31].

Formally:

Definition 1. A set of programs $\mathcal{L}$ taking as input values from an arbitrary set X is said to be *closed under self-interpretation* if there exist:

1. A quoting function $Q : \mathcal{L} \to X$ that takes a each program in $\mathcal{L}$ to its source code, or any source code that represents that program. Q does not necessarily belong to $\mathcal{L}$.

2. An evaluation function $\text{eval} \in \mathcal{L}$ such that for all programs $p \in \mathcal{L}$ and all inputs $i \in X$, we have $\text{eval}\left(Q\left(p\right), i\right) = p\left(i\right)$. On all other cases, namely $\text{eval}\left(x, y\right)$ where x is not of the form $x = Q\left(p\right)$ (i.e. x is not a quoted program), we require $\text{eval}\left(x, y\right) = 0$. Formally:

$$\left[\forall p \in \mathcal{L} \forall i \in X . \text{eval}\left(Q\left(p\right), i\right) = p\left(i\right)\right] \bigwedge$$

[29] Self-interpretation is at the same fashion as Universal Turing Machines, while those of course don't satisfy the decidability requirement. cf. e.g. the Halting problem and Rice's theorem.

[30] Namely there's no restriction for the forms that the recursion may take, or in other words, can recurse any machine in $\mathcal{L}$. Note that unrestricted recursion is implied from self-interpretation since any self-interpreter may be used in order to simulate recursion, as we shall demonstrate below.

[31] It also implies non-monotonicity, cf. footnote in Appendix C regarding Datalog+EVAL.

$$[(\neg\exists p.x = Q(p)) \rightarrow (\text{eval}(x, y) = 0)]$$

2.5.2 Derivation of the Logic

We first show how nontrivial the self-interpretation requirement is. The following theorem and proof are slightly modified versions of those in [4, 5, 7] and demonstrate the well-known result stating that no total language may self interpret:

Theorem 1. *Let $\mathcal{L}$ be any set of programs that is closed under self-interpretation and under Boolean operations. Then $\mathcal{L}$ is not total, namely it is not defined on all its possible inputs, or in other words, $\mathcal{L}$ contains machines that do not halt on some elements of X.*

Proof. Given the assumptions we construct a nonhalting program in $\mathcal{L}$. By assumption we can write eval as

$$\text{eval}(x, i) := \text{if } x = Q(p) \text{ then } p(i) \text{ else } 0 \tag{2.1}$$

Consider the function

$$\text{evil}(x) := \text{if eval}(x, x) = 0 \text{ then } 1 \text{ else } 0$$

By the closure assumption and by the assumption that eval $\in \mathcal{L}$, clearly evil $\in \mathcal{L}$. By definition,

$$\text{evil}(Q(\text{evil})) = \text{if eval}(Q(\text{evil}), Q(\text{evil})) = 0 \text{ then } 1 \text{ else } 0 \tag{2.2}$$

but by (2.1) we have:

$$\text{eval}(Q(\text{evil}), Q(\text{evil})) = \text{evil}(Q(\text{evil}))$$

therefore (2.2) reduces into

$$\text{evil}(Q(\text{evil})) = \text{if evil}(Q(\text{evil})) = 0 \text{ then } 1 \text{ else } 0$$

which is a contradiction unless evil $(Q(\text{evil}))$ never halts[32]. □

[32]Equivalently, its output is undefined.

Corollary 1. *All logics having the satisfaction relation $\models$ defined over all formulas and all structures, do not admit a self-interpreter.*

This rules out virtually almost all common logics, including FO, SO, and HO[33].

Corollary 2. *Exactly one of the following two alternatives may take place: either the law is unprotected against undesired law change, or,* ***not everything is judgeable****.*

Proof. We have pointed out that without self-interpretation the law cannot interpret a newly proposed law and by that cannot protect itself under undesired law change. If the language of law doesn't support self-interpretation, then in particular any specific law cannot support it. But if we support self-interpretation, we can construct a nonhalting machine as in Theorem 1, which means that there are situations in which no legal/illegal value may be assigned. □

To demonstrate the "undefined" situation in a legal setting we recall the ancient Greek paradox of Protagoras, also called "the Paradox of the Court". Quoting the paradox from Wikipedia:

> "It is said that the famous sophist Protagoras took on a promising pupil, Euathlus, on the understanding that the student pay Protagoras for his instruction after he wins his first court case. After instruction, Euathlus decided to not enter the profession of law, but to enter politics instead, and Protagoras decided to sue Euathlus for the amount owed.
>
> Protagoras argued that if he won the case, he would be paid his money. If Euathlus won the case, Protagoras would still be paid according to the original contract, because Euathlus would have won his first case. Euathlus, however, claimed that if he won, then by the court's decision he would not have to pay Protagoras. If, on the other hand, Protagoras won, then Euathlus would still not have won a case and would therefore not be obliged to pay. The question is then, which of the two men is in the right?"

Indeed we observe a situation which is undefined and can be modeled as an infinite loop: if Protagoras won then Euathlus lost, but then Protagoras lost so Euathlus won, and so on ad infinitum.

[33]From here and to the rest of the paper our terminology is of Finite Model Theory, cf. [1]. A good starting point would be the Wikipedia article named Descriptive Complexity.

Corollary 3. *If a class of machines $\mathcal{L}$ satisfies the requirements from law, then it must contain non-halting machines and also must possess a decidable halting problem for all its machines.*

This demonstrates the difficulty in finding a class $\mathcal{L}$ which is both decidable and has a self-interpreter, as well as the dissonance between halting machines and machines with decidable halting problem.

A function that is defined on all its inputs (namely has no "undefined" value) is called *total*. Clearly, plain first and higher order logic over finite structures, are total. We are not aware of enhancements to classical or intuitionistic logic that support the undefined value, except three cases: undecidable classes, the deterministic transitive closure operator[34], or, partial fixed-point logics which is the case discussed below.

Time-bounded machines are therefore ruled out because they never loop forever. The situation with space-bounded machines is different. A machine may loop indefinitely even under constant space (memory), still such classes do have a decidable halting problem. More precisely, a class of Turing machines which never takes more than $f(n)$ bits of memory, where n is the length of the input and $f : \mathbb{N} \to \mathbb{N}$ is any computable function, has a decidable halting problem, and whether it halts or not can be decided in no more than $2^{f(n)}$ steps. This gives rise to the well-known (and almost trivial) result that

$$SPACE\left(\mathcal{O}\left(f\left(n\right)\right)\right) \subseteq TIME\left(\mathcal{O}\left(2^{f(n)}\right)\right)$$

although time and space complexity classes are of asymptotic nature and refer to problems rather machines, while our argument refers for all inputs not only asymptotically, and refers to machines rather problems. We are therefore hinted and guided to look into space complexity classes[35].

The decision procedure for the halting problem for space-bounded machine is as

[34] We will not discuss this operator here but it goes along the same lines as PFP for the matter of capturing space-bounded computation, PFP for PSPACE and det-TC for LOGSPACE.

[35] Space complexity classes enjoy several highly desirable features: they are closed under complementation (Immerman–Szelepcsényi theorem), and are also equal to their nodeterministic counterpart by Savitch's theorem, which suggests that they are also closed under inverse image. Both theorems are in case the class is expressive enough, and both hold for PSPACE and above. PSPACE, which is a space complexity class to be discussed below, is also closed under if-then-else and Kleene star, and we also know that PSPACE=IP by Shamir's theorem. In addition, space complexity classes (above some level) correspond to alternating time complexity classes which logically corresponds to quantifier alternation.

follows: since the machine's configuration at each point of time can be described with at most $f(n)$ bits, there cannot be more than $2^{f(n)}$ configurations. The machine then loops indefinitely if and only if it visits the same configuration twice, a situation that can be detected in no more than $2^{f(n)}$ steps.

Before we continue, we demonstrate how recursion can be simulated using a self-interpreter. We show how to implement the factorial function $g(n) = n!$ using the above E and Q functions (E is above's eval). We do so by defining an auxiliary function h:

$$h(x, y) := \text{if } y = 0 \text{ then } 1 \text{ else } y \times E(x, y - 1)$$

and then:

$$g(y) := h(Q(h), y)$$

for example:

$$g(1) = h(Q(h), 1) = \text{if } 1 = 0 \text{ then } 1 \text{ else } 1 \times E(Q(h), 0, 0)$$

and

$$E(Q(h), 0, 0) = h(0, 0) = 1$$

Observe that g, h are not recursive nor mutually-recursive.

We now proceed constructively. Consider a language with the primitives if-then-else and Q, E being the quoting and eval operators. In order for this class to have a decidable halting problem, we'd like the number of configurations it may take to be finite[36]. A logical characterization of the non-visiting-twice condition is precisely the PFP (partial fixed point) operator originally introduced by Abiteboul and Vianu and is well-known in the field of Finite Model Theory. Indeed, over oredered structures, it corresponds to a space complexity class: FO[PFP]=PSPACE [1, 2]. Denote by FO[PFP]+EVAL the class FO[PFP] enhanced with the quoting and evaluation operators Q, E above. We shall prove the following theorem in Appendix E:

Theorem 2. *For any FO[PFP]+EVAL formula there exists an equivalent FO[PFP] formula.*

This result can intuitively be seen as follows: since the set of all possible k-ary

[36]This doesn't mean that this is the only class with a decidable halting problem.

relations over a finite universe, is finite, we can therefore use the PFP halting condition with or without the quoting and evaluation predicates. The program may still take only finitely many configurations, all still in polynomial space. As a corollary we have:

Corollary 4. *The [equivalent] languages P-DATALOG, FO[PFP] and FO[PFP]+EVAL satisfy the laws of laws.*

The equivalence of FO[PFP] with P-DATALOG is derived in [1], and a sketch of P-DATALOG appears in Appendix D.

Furthermore, it is easy to see that HO^i[PFP]+EVAL also satisfy the laws of laws. However this is not the case for HO[PFP]+EVAL. The reason is that one may edit the quoted program and raise its order and then cause it to consume exponentially larger space (in case of raising the maximum order by one) and by that we lose the PFP halting condition[37].

More about the paradox of self amendment discussed in [6] cf. Appendix C .

2.6 The Internet of Languages

After establishing the logics that support laws, we have to come down to a concrete language. However we reject the concept of "universal language" and postulate that no single language is adequate for all purposes. We acknowledge that many languages should not only coexist and be mutually interchangeable, but they should also have the ability to evolve with time. We therefore come up with a meta-language that is capable of defining new languages, but then, one might claim that we solved nothing, and even though we reject the idea of universal language, we still came up with a universal meta-language. To this end we require the meta-language to be able to self-interpret and by that redefine itself and change with time. We therefore achieve a situation in which the choice of language doesn't matter while even the meta-language is not fixed.

The internet of languages is therefore a set of translators. It is important to note that we do not at all consider translations or even processing of natural languages, and our scope is restricted to formal languages[38]. Once a translator[39] from language X to

[37] So one might argue that HO[PFP]+EVAL doesn't even meaningfully exist.

[38] Surely it might be the case that one day we'll figure out how to support natural language to the desired nontrivial extent that Tau requires, but we don't count on such an event.

[39] All translations we refer to are assumed to be semantic-preserving translators.

language Y is written and submitted into the internet of languages, and similarly a translator from language Y to language Z, we then get a translator from X to Z "for free". This gives rise to calling it an "internet" of languages.

Formally:

Definition 2. Let $L \subset \mathcal{P}\left(\{0,1\}^*\right)$ be a set of sets of bitstrings where $\mathcal{P}$ denotes the powerset and the star is Kleene's. Denote by $\mathcal{L}$ a labeling function $\mathcal{L} : L \to X$ where $X \subset \{0,1\}^*$ is a set of labels and $\mathcal{L}$ is bijective. Assume we are given an equivalence relation $\sim_\ell$ for each $\ell \in X$ under the provision that two bitstrings are considered "equivalent" if they're semantically equivalent, and the specifics of this semantic equivalence is abstracted by $\sim_\ell$. Call the elements of L the "languages". An *internet of languages* is a function

$$F : X \times X \times \{0,1\}^* \to \{\bot\} \cup \{0,1\}^*$$

taking as arguments two labels of languages called the source and target languages, and a bitstring called the input document, and returns either $\bot$ or a document in the target language such that for all $\{\ell_1, \ell_2\} \subset X$ and $x \in \{0,1\}^*$ it satisfies

$$\forall \{\ell_1, \ell_2\} \subset X . \forall x \in \{0,1\}^* .$$

$$x \notin \mathcal{L}^{-1}(\ell_1) \bigvee$$

$$F(\ell_1, \ell_2, x) = \bot \bigvee$$

$$[F(\ell_1, \ell_2, x) \sim_{\ell_2} y] \leftrightarrow [F(\ell_2, \ell_1, y) \sim_{\ell_1} x]$$

Furthermore, we require that $F \in L$ when F is seen as a subset of $\{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \times \{\bot\} \cup \{0,1\}^*$, namely as a set of bitstrings each being a concatenation of four bitstrings (this is the self-interpretation requirement).

Loosely speaking, F has its transpose being its inverse. Observe that F defines a "global" equivalence relation $\sim$ over $\dot{\bigcup}_{\ell \in L} \ell$ (we use a topological notation here for "explicit" disjoint union, namely tagging each bitstring with the language it belongs to) which defines semantic equivalence over bitstrings from different languages.

2.6.1 Tau Meta-Language

In order to define a new language one defines how it translates to an existing language. Writing such a translator is done using TML, the Tau Meta-Language, an implementation of FO[PFP] (or more precisely of what [1] refers to as P-DATALOG)[40] which therefore admits the above requirement of self-interpretation. Another important requirement is the ability to go back and forth, namely given a translator from language X to language Y we wish to have a translator from Y to X "for free". This is not mathematically possible in Turing complete languages, but TML is capable of supporting that[41].

By TML being an implementation of FO[PFP] it fulfills the requirements from the languages of law in addition to the requirements from the meta-language. However TML is intended to be a meta-language indeed and does not come with KR&R (knowledge representation and reasoning) features. It is intended to be a tool for implementing KR&R languages. In other words, TML is a compiler-compiler intended for logic, knowledge, and other formal languages. But since TML fulfills the laws of laws we can then write a translator from some KR&R language into TML and then run the resulted TML code. The usage of TML in the system is therefore two-fold.

TML's translation workflow is as follows. Given an input document D in some language X, we wish to translate it a semantics-preserving translation into a document D' in the target language Y. A TML program consists of a grammar[42] and logical rules. Denote our translator from language X to language Y by P, where P is a TML program implementing this translation[43]. TML's backend will then run D through the grammar supplied in P and construct a parse forest. The logical rules in P will then edit this parse forest by adding and deleting nodes, eventually obtaining a tree in which its yield is D' in language Y.

We remark that this internet of languages may be used for other text conversion tasks beyond translation between KR&R languages, as well as building correct-by-

[40] To possibly be enhanced to HO^i[PFP]+EVAL.

[41] It is possible to use nondeterministic Turing machines in order to compute the inverse image for any machine, given it is guaranteed that the inverse exists. But on our case, NPSPACE=PSPACE by Savitch's theorem.

[42] For now TML supports context-free grammar extended with some non-context-free features, though ultimately it can support all context sensitive languages because they're both PSPACE-Complete.

[43] The specifics of the translation are of course up to the TML programmer.

construction compilers. As an example, for the sake of convenience and organization one can write a translator that creates a Wiki based on a body of formalized knowledge. In its full generality, TML is a generic compiler-compiler[44].

Further we remark regarding the existence and possibility of use of so-called Controlled Languages[45]. This concept is at the fashion of "simple enough English that machines can understand". Sentences written in such languages are valid (or almost valid) natural language sentences, but are forced to take a certain shape e.g. subject-verb-object. Those languages won't cover nearly all of valid natural language sentences, but they still might be more approachable than languages which don't take any form that resembles natural language.

2.6.2 Futamura's Projections

In a seminal paper, Futamura[5] has shown how to use a partial evaluator in order to automatically generate a compiler out of a given interpreter, as well as to automatically obtain a program that generates a compiler out of any interpreter. Those would be two out of four Futamura's projections. Such an ability is of course very appealing at the scope of the internet of languages. It is therefore planned to create a partial evaluator for TML and by that enhance the internet of languages with such important abilities. A partial evaluator may be used in more cases, especially for optimization purposes. In fact Futamura's projections and partial evaluation themselves essentially mainly offer optimization. Indeed given a source code of length M and a partial input of length N, the naively (i.e. under mere substitution without any optimization) partially evaluated program has maximal size of MN, therefore computable even in quadratic space, since $(M+N)^2 \geq MN$. Further optimizations of the partially evaluated code can be anything else up to TML's expressive power.

[44] Up to tasks which are out of its complexity class, e.g. certain optimizations and typechecks

[45] cf. e.g. Attempto Controlled English (ACE).

3 Agoras

3.1 Contracts

Contracts in a blockchain network, also referred to as "smart contracts"[46], have been a subject of interest. Allowing predefined actions to be taken on the blockchain as a function of events occurring on the blockchain as well, is not very different than the case of the language of law. Inability to reason over Turing-complete "smart" contracts has proven itself time and again to cause major financial losses to innocent parties. Not only decidability is required but also self-interpretation, surprisingly or not. To quote from [6] section 20:

> "Self-amendment appears in many contexts other than constitutional amending clauses. If a written contract stipulates that only written modifications will be effective, may that provision be modified orally? It is reflexive either way: if it is orally modifiable, then self-excepting, and if not, then self-applicable. Can a contract's no-waiver clause be waived? Why can a will's no-revocation clause be revoked? Why can a will's nocontest (*in terrorem*) clause be contested?"

We therefore observe how contracts share logical properties with laws, in particular when it comes to self-reference and amendment, which is not a surprising result. As we saw, coming up with an adequate logic is not an easy task and we presented here such a logic for the first time. In Agoras we therefore consider contracts formalized in logical formulas which admit the laws of laws. Special cases of such contracts are considered below: trading knowledge, renting computational resources, derivatives over financial assets, and more.

In addition, such contracts must be able to be settled and enforced on the blockchain, namely the network has to have the ability to verify the conditions for coin transfers.

Contracts in a decidable language are of interest by their own. It allows to ask all relevant questions about the contract, in particular its outcomes in certain scenarios, and to have a guarantee that whatever formalized negative result may never take place.

[46] Although currently contracts over blockchains which are referred to as "smart contracts" do not have any justification for bearing this name.

3.2 Economics of Knowledge

In our current world one can rarely monetize knowledge in a direct manner. We almost never buy or sell (or price) a single piece of knowledge. Seeking knowledge in an economical setting is currently mostly done by appealing to people who we assume to be involved in a field where the knowledge of interest may exist, just like there isn't a book for each single answer, but we look up answers in books that deal with apparently relevant fields. Knowledge is a major part of our economy, and here we seek for cases in which we can make the economics of knowledge more efficient and more advanced.

Formalizing knowledge in a machine-accessible format offers an advantage over books and search engines: the data isn't a stream of bits anymore but the meaning behind the bits, and therefore allows to perform an automatic semantic lookup of small pieces of knowledge even in a large compilation of writings.

We mentioned how a shared knowledge base will evolve over Tau and its logic-based discussions. In a framework where knowledge is constantly formalized and shared between users, it's only natural to consider an economical infrastructure to facilitate all aspects of the economics of formal knowledge. This is the topic of the current subsection.

3.2.1 Production, Supply, Demand, and Pricing

Let us first consider how knowledge is generated, or "mined". If we treat knowledge as something beyond [natural or artificial] sensory inputs, one can argue that the act of reasoning is the one that generates new knowledge from existing one. Such a process can also be done automatically. But how can we guide this automatic search process such that it'll produce interesting results?

In subsection 2.2.3 we spoke about the concept of "interesting questions". A piece of knowledge is valuable if it's interesting, and its value should depend both on the level of interest and on the hardness of answering the question. Observe that being interesting is relevant mainly for the asker (the "buyer") while hardness of answering is relevant mainly on the answerer (the "seller"). Questions and answers therefore [partially] correspond to demand and supply in the economics of knowledge.

Unlike the concept of being interesting which is purely a human thing, hardness of answering is something slightly more accessible to machines, since they can measure

steps and resources, and besides there is a rich theory about complexities of various reasoning tasks. This gives rise to more accurate pricing.

3.2.2 Knowledge-Cash Transactions

In a broad sense, users interested in certain questions may offer a reward for an answer. Verification of answers may be done in several ways. In some cases, like certain common mathematical questions, the answerer may supply a proof for the answer and no dispute arises over whether or not the answer is correct. We might even have information about whether such a verification process is expected to be efficient[47], again by using considerations from complexity theory[48]. There's more to add to this point from a cryptographic point of view (e.g. zero-knowledge proofs and homomorphic encryption), but this is out of the scope of the current paper.

But sometimes asking for a mathematical proof is too much and not only in cases where it is computationally expensive. Sometimes one might trust an expert in the traditional way, simply by impression or recommendation or advertisement etc. as common, and then automatically trust their answers. A simple example would be to trust a medical doctor which you already know well[49], and not require them to supply a mathematical proof for each and every medical advice they give, as this will render the whole thing impractical[50].

We proceed with additional use cases for knowledge economics over Agoras. Consider some reputable body, like a university or a trusted expert, which takes the hard task of formalizing some large and useful body of knowledge. They can then offer a subscription to users for automatic participation in their discussions. As explained above, Tau will allow "autocomment" so it can automatically participate in a discussion on your behalf based on your Worldview. That subscription lets subscribers enjoy automatic participation in discussions where the data comes from a trusted source (from

[47]Namely we want the computational complexity of verifying the proof to be significantly lower than of finding the proof. This is captured, for example, by the complexity class NP, and in a different fashion by the complexity class IP, which happens to equal to PSPACE by Shamir's theorem.

[48]Even more refined measures of complexity are given under the concept of "Fixed-Parameter Tractable" (FPT).

[49]Or, trusted by an entity which you also trust, to the extent of "if this entity says that this user is a real and good doctor, then I take their word for it".

[50]However when Tau accumulates enough knowledge we can certainly expect an automated well-justified medical advice.

the single user's perspective). For a specific instance, a law firm offers their knowledge-base to automatically participate in corporate discussions and possibly autocomment on certain ideas to be legal or illegal.

Another form of subscription may be pay-per-query, allowing subscribers to ask questions and get answers, with or without revealing the whole knowledgebase.

Furthermore, thanks to Tau's collaborative knowledge formation aspects, a group may formalize knowledge and monetize it together.

3.2.3 Freestyle Knowledge

That all emphasized, clearly a human touch is crucial for many forms of knowledge transfer, and not all knowledge may or is suitable to be formalized under all circumstances. Seeking quick advice from a doctor by means of exchanging formalized knowledge is not always the preferred way to go, same for taking a private tutor, and so many more examples. We therefore intend to also have a freestyle form of trading knowledge, in the form of text, audio, and video, based on the micropayments protocol described in [9], in a fashion similar to [14]. This final touch gives Agoras everything it needs in order to be a fully functional knowledge economy.

3.3 Computational Resources Market

Agoras will fascilitate a market for renting and renting-out computational resources. They can be used, aside the generic use case of computation, to perform reasoning tasks that occur during the normal operation of the system. We will not cover this subject now but we did so to a large extent on Zennet's materials, cf. e.g. [10]. Zennet's pricing formula appears on [13] which eliminates the risk of mispricing[51].

In contrast to Zennet's design, renting computational resources is done by a contract in the fashion of section 3.1.

We will however mention one point which we shall use later on: mitigating the risk of wrong computation (intentionally or not). We already spoke in subsection 3.2.2 about efficient proofs, and in such cases, they may as well be utilized here. In other cases, one way to probabilistically verify unverifiable computations is to calculate the same thing more than once (by randomly choosing more hardware providers), so increasing

[51] Non-precise pricing of computational resources is a security risk, because it can then be exploited.

the cost linearly, decreases the risk exponentially (e.g. x10 more cost decreases the risk to the power of 10).

3.4 Derivatives and Risk-Free Interest

Agoras will offer an innovative derivative market (as in put-call options), featuring the ability to receive risk free interest for deposits without the issuance of any new coins, or in other words, without inflation.

Before we get into details we remark that the proposed derivatives market is completely peer-to-peer with no man-in-the-middle whatsoever. More generally, there is no central entity or a toll-taker across the whole design of Tau and Agoras.

The underlying assets in this derivatives market are of a very generic nature: they can be virtually any [semi-]martingale, or more specifically, the price of goods over the network such that its transactions can be settled on the blockchain (e.g. knowledge or answers to hard computational tasks, or virtual assets encoded over the chain), or even exogenous to the network by introduction of so-called "oracles". This, as long as the pricing function and the contract are both in logic that satisfy the laws of laws, as we emphasized above.

A common question arising with options trading is how to price the them. For that there's a standard, Nobel-winning formula, called the Black and Scholes model. In real life, the actual premium prices that people pay or charge for entering a contract typically varies from the exact theoretical (Black&Scholes) price, depending on how the market participants foresee the future.

The Black and Scholes model offers important information beyond only pricing an option. It can also measure the sensitivity of the option's price to changes in the value of the underlying asset (the Delta), and several more important indicators, called the "Greeks". Taking it a step forward, Black and Scholes showed that certain combination of options can yield a risk-free interest. It is risk-free because the sum of the deltas of the options, is zero, cf. Zero-Delta Portfolio. It also yields an interest due to the time value of money being taken into account in the price of the option: an option with expiration date in two months should be priced higher than an option expiring in one, because the uncertainty is typically larger, and because of the higher opportunity cost incurred by having to lock a collateral.

The use of derivatives markets is typically two-fold: hedging and speculation. Common derivatives markets in the world are leveraged, and leverage is just a laundered term for taking a loan, allowing very risky speculations, and was widely criticized (at the scope of derivatives) as endangering the world economy. Therefore Agoras' derivatives market will not support any kind of native leverage. The more genuine need for options, which is why they were invented in the first place, is hedging. Suppose we have a client of a European company exporting to the China, such that it gets EUR and so the Chinese client holding CNY has to convert CNY to EUR. If the EUR/CNY ratio goes up, the client may suffer loses.

To avoid that, the client can hedge by buying options in a derivatives market. The client will pay a premium but will reduce the risk implied by currency fluctuations. A counterpart for that option may be a client of a company getting paid in CNY but the client is holding EUR. To give a cryptocurrency-related hedging example, suppose someone buys a lot of Bitcoin mining hardware and pays for that in CNY. The revenue from mining in terms of bitcoins is more or less known, but if BTC/CNY price goes down, the mining operation might incur losses. Therefore they might want to buy an "put" option, namely an option to sell BTC at a fixed price on a fixed date, and by that reduce (or even eliminate) the risk of curency rate fluctuation.

Getting back to Agoras' risk-free interest without inflation, the answer to the question "so where does the interest money come from if not newly printed?" would be that it comes from the hedging needs of the players in the economy. Note that this is not an arbitrage and indeed Black&Scholes assume that the market has no arbitrage opportunities[52]. The income from a zero-delta portfolio will reflect the time value of money, or in other words, the reward for locking a collateral, in addition to the opportunity cost.

3.5 Applications

3.5.1 Decentralized Search Engine

To make the ideas even more concrete let's consider a decentralized search engine, which was one of Agoras' original goals that was later generalized into the concept of knowledge economy. Google has at lease one million physical servers which need to crawl, index, and search the internet. So to compete with Google's search engine one

[52]Namely the "Efficient Market Hypothesis".

would expect to require at least the same order of magnitude of physical hardware. This magnitude is not so big comparing to the computational resources held by households: once city may have several times of Google's resources[53]. But in a decentralized network featuring a decentralized search engine, who is going to pay for all that computation?

The task of maintaining a web search engine depends on data which is completely unknown until someone goes to the internet and discovers it. Therefore such a task cannot be completely trustless, as one cannot prove that they downloaded and indexed into the search engine the correct data, and didn't modify or omit it. But although this problem isn't completely solvable without elements of trust, it is still solvable to some probabilistic extent with risks that can be lowered arbitrarily, theoretically, as explained in 3.3.

So having a decentralized search engine requires the ability to fairly rent and rent out computational resources, under acceptable risk as a function of cost. Continuing further, given a hardware renting market, what would be the next steps towards a decentralized search engine?

A web search engine consists of users and maintainers. The users supply search queries and the maintainers answer the queries, and to do so quickly they must have the whole web already indexed. Naturally, users have to pay to maintainers a payment that depends on the amount of usage and the cost of maintenance. But in a decentralized network the users and the maintainers are really the same entity.

To maintain the network all one has to do is to run the client and by that participate in indexing and searching. A user might query a certain amount of queries per day, while their computer can answer another certain amount of queries per day[54]. The computer can answer orders of magnitudes more queries than a single average user can manually provide. So a home users who uses tens or hundreds of "googlings" per day, and also run a client that supports the network, are expected not only to not have to pay but even to earn, as they'll serve others more than they consume. But we will also have heavier-usage users that may consume the search services in large amounts (e.g. automatically). Such users will have to pay and will not end up break even. This

[53]Moreover, the world's strongest supercomputers are only as strong as several thousands of commodity GPUs. Also, much of Google's computers are old and weak ones, weaker than a common smartphone. Heavily utilizing cheap old hardware was in fact one of Google's innovations.

[54]Answer to other users, as no single user will store the whole internet, while one can't predict tomorrow's queries.

points to a characteristic of an economy built to be fair in the sense that the money flows in the "right" rather the "wrong" direction: it's a zero-sum game, yet the money flows from the big and rich entities to the smaller ones.

3.5.2 Semantic Search

If stopping here, we didn't actually contribute much: so we'll have a better and decentralized search engine that gives many users some income, but life will continue more or less the same. Searching, as we know it, is no different than searching a web page with Ctrl-F together with a thesaurus. That's what Google is doing, more or less: it Ctrl-F's the internet under an open thesaurus. But making an economy out of knowledge has many more possibilities. Importantly, we'd like to incorporate into our knowledge economy also deeper and more meaningful knowledge than "those words or their equivalents are mentioned on that website".

Ultimately, we'd like to upload many (if not most) of our thoughts, opinions, and intellect, to the internet. We already do it. But all our search engines know to do is to use Ctrl-F and a thesaurus, namely they operate on a very shallow level, not even "shallow understanding". But knowledge that people actually seek for never comes in this form. We don't seek for documents that contain certain words but for documents that actually answer our questions. Similarly, we don't seek for professionals in which all they know is to search the internet, but we seek for professionals from which we expect a deep understanding of their field of expertise, and not their ability to mention a bag of words. Otherwise we'd do good enough with Google and wouldn't need professionals anymore.

Thanks to Tau's knowledge formalization capabilities, and with time, the dream of semantic web search engine can come true on top of the mentioned decentralized search engine.

3.5.3 Automatic Businessman

Users have their own local assets, being computational resources, knowledge, coins, and possibly other assets that they allow their formalized worldview to take into consideration. Agoras will then be able to tailor a deal by looking at the available assets and contracts offered out there, and by publishing bids for certain assets or contracts, all

from a coherent and logically proven plan of combining the assets and opportunities into a good deal. One can even reach an extent that is unheard of in common automatic planners: the user will be able to ask Agoras for deals that don't break laws and regulations, once law is formalized over the system. This is just a small example of what a logical reasoner may perform in an economy.

As we explained, Agoras will also contain a computational resources market and a future contracts market. The Automatic Businessman is therefore a holistic application that culminates all parts of Tau and Agoras capabilities. It should also be remarked that as for now, only Tau and Agoras offer a real method for achieving the [relatively] old concept of DAOs (Decentralized Autonomous Organizations).

4 Evolution and Impact

4.1 More Fair Legislation and Economy

A major assumption in economics is the one of efficient markets (the "Efficient Market Hypothesis") which, roughly speaking, demonstrates that the speed of information propagation in the economy is tied to the lack of arbitrage, rendering the market "efficient". People with privileged knowledge, e.g. insider trading, have higher chances of making profit, which is of course unfair and mostly illegal. The hypothesis also point that in an efficient economy the only way to make profit is by taking risks. Therefore better propagation of knowledge is a matter of fairness, because it entails that people cannot make profit without taking the associated risk. This doesn't mean that a fair economy is just like a casino: in a casino the expected profit is negative while in a healthy economy the expected profit is positive. This also doesn't mean that an efficient economy is automatically fair. It only means that a priviledged ability to make profits under no or less risks, is unfair.

Tau and Agoras take the information-propagation levels to new orders of magnitude, and by that offer a much more efficient economy than ever existed. Another aspect of fairness is the deflationary nature of Agoras: Agoras will not print new coins. From all new fiat money printed over, say, the last decade, how much of it did you and I receive? Zero. It amounts to an implicit tax, given to parties which we never intended to personally financially support. It is remarkable that Agoras is still able to recover risk-free interest without any inflationary aspect.

An economy cannot exist without a society, and is a thing within society. Unsurprisingly, all economists consider themselves as a kind of social scientists. Unlike physics[55], economics depends, and cannot be defined without, concepts that exist purely in human imagination. The principle that yields this status of economics is simple but very deep: we cannot have an economy without some subjective valuation (or "utility function"), namely the ability to say "I prefer this over that". This can be thought of like taste: some prefer chocolate over ice cream, and some go the other way around. They might even be ready to pay different prices. In the bottom line it comes down to ethical value systems[56]. Such values are indeed not physical terms but exist purely on our

[55] At least apparently.

[56] Ethics, in its broad sense, is nothing but the definitions of "good" and "bad", "better" and "worse",

imagination. And this doesn't make them a single bit less important.

Fairness of economy depends very much on the social and legal climate. One cannot do business without knowing the legal implications of relevant scenarios, and one cannot do business in the absence of truth and trust. We offer, for the first time, a sound solution for reasonable laws and legislation, justice, ethics, and social contracts, one that can even be automatically embedded into business plans, cf. the above Automatic Businessman. Furthermore, the formalization of values gives rise to automatically avoiding what you define as "bad" and achieving what you define as "good".

4.2 The Critical Mass and the Tau Chain Reaction

We emphasized that the futuristic features enabling effective large scale discussions depend on the usage of formal languages. How can we expect Tau to have a large user base given those language barriers?

Indeed we expect the initial users to be familiar with such languages, e.g. by being involved in formal logic or in programming. But those initial users will deal with programming Tau itself and enriching its knowledgebase (and also adding and improving languages over the Internet of Languages), as Tau is nothing but a software controlled by its users. Having, say, one hundred users that program the system by merely stating (implicitly or explicitly, and possibly in a disorganized fashion) pieces of what they want Tau to do, is a very small user base on one hand, but on the other hand it is a large development team developing the system itself and formalizing knowledge, and by that they'll make it accessible to slightly more users, say, two hundred users. A critical mass of initial users should therefore be enough to start a chain reaction of advancing the system with time to fit larger and larger audiences. cf. also subsection 2.4.

4.3 The Singularity

Given one wish, what would be the best wish to choose? An answer that creeps in is infinitely more wishes. Nowadays, the scientific and technological equivalent of one wish which amounts to infinitely more wishes, is the creation of an AI, or in a more modern terminology, Artificial General Intelligence (AGI). So much so, that it is virtually not

and in this sense it is a valuation system.

worth putting serious efforts into anything else other than achieving an intelligence higher than ours, as this will let us fulfill infinitely more wishes. The future point in time in which an AGI becomes smarter than all of us is known as the Singularity.

The task of creating such an AGI is overwhelming. As experience shows, not even large corporations are capable of achieving it. The main difficulty is not creating an "artificial brain", but making it know everything it should know in order to really advance us as would a very smart person. Formalizing sufficient parts of human knowledge is a huge task that cannot be done by means which are currently reasonable, e.g., a million people voluntarily[57] contributing small parts of knowledge each. But in a discussion platform with many (millions) of participants writing in formal languages, such a knowledge base has a real chance to emerge for the first time in history.

Our brain has an order of magnitude of 10G neurons operating in frequency of 5-50Hz. Common laptops have memory of such 10G order of magnitude[58] but operate in about 3GHz[59], namely around a billion times faster. Common GPUs would have about x1000 more computational power than a common laptop, so we end up with about x1,000,000,000,000 (a trillion) speedup. So hardware-wise, we're already beyond the human brain (even if we'd count synapses rather neurons). However it turns out that our brain is even much weaker than the counting neurons argument suggests: our short term memory has a capacity of about 5-15 items. No such limitation is in place for machines. Over those 5-15 items we can perform logical reasoning and inference and derive new knowledge, this, in a speed of no more than few times per second (to be generous). Is it therefore the case that a network of machines are able to recover the logical consequences which humans inferred over the course of history in the form of math and philosophy and science, overnight?

There's no reason why they shouldn't.

Furthermore, humans make mistakes, many of them are of the kind that machines aren't susceptible to. Sometimes we're building on a mistaken result, holding the mistake for a short or long time. Machines don't even need to look back: once programmed correctly, a result is always valid without hesitation. Back-of-the-envelope estimation would compare this situation to a random walk, and suggest that merely the absence

[57] As otherwise this won't be possible due to being too expensive.

[58] And even much more if taking disks into account.

[59] Add circa x10 for multithreaded CPUs.

of mistakes should make knowledge advance quadratically faster.

Another quadratic speedup comes from connecting the human's minds, which is all about human-machine-human communication. Tau offers combining brain powers on a large scale. The number of all possible connections between nodes/users/worldviews also grows quadratically. We're therefore already at a fourth power of acceleration, times a large constant which is the speed of calculation.

But there's even more to add to this estimate, which is the case of knowledge reuse. People forget things, in fact we remember a very narrow part of reality. It occurs to all of us that we have an open question for a long time, and suddenly we realize that we already knew the answer, just never linked it to that question. This is not going to happen over Tau.

Yet another booster of knowledge creation and propagation are economical incentives, which is what Agoras' economy of knowledge is all about.

Humanity never experienced nearly such an acceleration of knowledge growth. Where will it take us? Well, it'll surely bring us to the point of Singularity more sooner than later.

References

[1] H.D. Ebbinghaus, J. Flum, 1999, "Finite Model Theory".

[2] S. Abiteboul, R. Hull, V. Vianu, 1995, "Foundations of Databases".

[3] Y. Futamura, 1999, "Partial Evaluation of Computation Process – An Approach to a Compiler-Compiler".

[4] A. Bauer, 2016, "On Self-Interpreters for System-T and Other Typed Lambda-Calculi".

[5] A. Bauer, 2014, Stackexchange answer, "A total language that only a Turing complete language can interpret".

[6] P. Suber, 1990, "The Paradox of Self-Amendment".

[7] C.T. McBride, 2003, Haskell mailing list, "On Termination".

[8] S. Nakamoto, 2008, “Bitcoin: A Peer-to-Peer Electronic Cash System”.

[9] Bitcoin Wiki, “Rapidly-adjusted (micro)payments to a pre-determined party”.

[10] F. Brandt, V. Conitzer, U. Endriss, J. Lang, A. D. Procaccia, 2016, “Handbook of Computational Social Choice”.

[11] O. Asor, 2015, “About Tau-Chain”

[12] O. Asor, 2014, “About Zennet”.

[13] O. Asor, 2014, “Zennet Pricing Algorithm”.

[14] O. Asor, 2014, “Bitagoras”.

A Summary of Problems, Questions, and Answers

We sketch here a list of problems and questions that are addressed in this paper together with a short description of the proposed solutions and answers:

Question How can many people effectively share opinions and reach a collaborative decision?

Answer By retaining features of small-scale discussions in large scale settings.

Question How may programs accurately detect what users agree on?

Answer By the users formalizing their opinions in a *decidable formal logic*.

Question How to avoid information loss and negligence when processing large amounts of information?

Answer By using formal logic and the aid of the machine, no piece of knowledge will be effectively lost.

Question Which languages exactly should the users use?

Answer Not a single language but a set of languages that may grow and evolve over time.

Question How to avoid paradoxes arising from self-amendment?

Answer Adhering to the *laws of laws* (presented below) and logics that admit them.

Question How to avoid running into a deadlock?

Answer This is not a solvable problem, however we offer some remedies.

Question What is the optimal governance mechanism for the Tau network?

Answer There is no optimal mechanism, each one has its own pros and cons. The first users will have to discuss and agree on which one to implement.

Question Can Tau solve the fake news problem?

Answer No. It cannot distinguish between truths and opinions.

Question Will Tau be usable by virtually all people or only by e.g. programmers?

Answer Not widely usable at first but will become so with time.

Question How is the time-ordering of the various Tau versions is determined over the network?

Answer Using a *blockchain*.

Question How to create an efficient knowledge economy?

Answer By introducing *knowledge-cash transactions*.

Question How to support inflation-free risk-free interest?

Answer By implementing *zero-delta portfolios*.

Question Can Tau bring to life the dream of Artificial General Intelligence?

Answer Yes, with time.

Question What is one's ultimate wish?

Answer Tau.

B Tau and Classical Social Choice Theory

For completeness we briefly review few elements from the classical theory of social choice and compare them to Tau's setting. For reference and more information cf. [10].

Classical social choice theory deals with a society choosing a course of action from some available alternatives. Normally, individuals within the society will cast their vote for this or that option[60]. At other times they may rank alternatives to form what is known as a preference profile[61]. In some situations, reasonable voting methods and rules will be fairly simple. For example we quote May's theorem:

Theorem (May's Theorem)**.** *For any 2-candidate election with an odd number of voters, the winning rule of simple majority vote is the only voting system which never ends with a tie, is anonymous (treating all voters equally), neutral (each candidate is treated equally), and monotone (if one candidate is chosen, and some voters change their vote to vote to this winning candidate, then that candidate remains the winner).*

Many voting contexts, however, do not satisfy all these assumptions, e.g. when three or more alternatives are considered (cf. Arrow's theorem below).

The process of aggregating individual preferences in a multi-candidate setting is abstracted within the framework of social choice theory by the so-called Social Welfare Function (SWF). The SWF is concerned with preference profiles which are assumed (in classical social choice theory) to form a linear order, namely there's a preference relation between any two alternatives, and this relation is transitive[62] and therefore acyclic. For the most part, the SWF is a mapping from a set of preference profiles (one per individual) to another preference profile representing the collective view within the society[63]. This framework leads to a number of surprising results. The first of these, observed by Marquis de Condorcet (1785), is the existence of majority cycles, where collective preference violates what one could expect from any rational individual, as we demonstrate now.

[60] We already pointed out in the introduction that voting cannot scale and stay fair at the same time.

[61] A preference profile is therefore a set of claims of the form "I prefer X over Y", as we shall detail below.

[62] Transitivity means that if x is preferred over y, and y is preferred over z, then x is preferred over z.

[63] In other words, is a function taking a set of linear orders, into a single linear order.

Consider the task of ranking three alternatives x, y, z, based on the preferences of three voters, A, B, and C: $x < y < z$, $y < z < x$, and $z < x < y$, respectively. Here, the binary relation $<$ is a linear order on the set of alternatives ($x < y$ reads: y is favored over x). Exercising a simple majority rule reveals that participants A and C prefer y over x, participants A and B prefer z over y, and participants B and C prefer x over z, so the majority (two in this case) decrees $x < y < z < x$, which is clearly a contradiction. This situation, known as the Condorcet paradox, may be averted by adding more opinions. For example, taking two more voters, D and E, whose ranking profiles are: $x < z < y$ and $y < x < z$ respectively, will turn the majority rule's conclusion to become $x < y < z$.

Arrow's impossibility theorem, another key result in social choice theory, states that under some fairness conditions the only SWF possible is a dictatorship, where the choice function mimics the preference of only a single individual, the dictator. Formally, Arrow's theorem states:

Theorem (Arrow's Impossibility Theorem)**.** *With three or more candidates and any number of voters, if the SWF always deterministically produces a single winner, and respects unanimity a.k.a. Pareto efficiency (if all voters prefer a certain candidate, then the SWF chooses this candidate), and satisfies the Independence of Irrelevant Alternatives (IIA) condition (the SWF orders each pair of candidates solely by the relations between those two candidates in each individual's preference profile), then this SWF is a dictatorship (there exists a single voter such that the SWF always chooses their profile as the final result).*

Researchers have criticized and in turn relaxed some of Arrow's assumptions, most prominently the IIA condition. Indeed voters may prefer a certain ordering between two alternatives depending on the ordering between other alternatives.

Tau's setting is strictly more general. Opinions may be any relation (alternatively any logical formula), not only a linear order. Tau's default SWF is as follows: if one user has two opinions A and B, and another user has two opinions C and D, all being logical formulas, then the consensus is $(A \vee B) \wedge (C \vee D)$. Observe that this SWF satisfy all requirements listed in May's theorem and also the requirements in Arrow's theorem except IIA and the uniqueness of the winning alternative. Lack of uniqueness may pose a Buridan's mule situation. This can be circumvented by waiting for more

opinions to narrow down the available options. Otherwise, we fall back to solutions described in 2.3.2.

C Tau vs. Nomic

This paper would not be complete without some appeal to philosophy of law. In what follows we quote from Peter Suber's exemplary exposition in [6]. We comment on his points as well as showing how our logics of choice solve the raised problems. The footnotes and bold markings in the quotes below do not appear in the original text but are our additions. Suber writes in section 21:

> "Because the paradox of self-amendment is a special case of the paradox of omnipotence[64], our central question is whether a deity or AC[65] can irrevocably limit its own power."

Our design allows an amendment clause to indeed irrevocably limit its own power, e.g. the law "all laws can be deleted given majority vote, including this law". Sketching it in FO[PFP]+EVAL is trivial. Suber continues:

> "...The logical or formalist view of law as it affects this problem I am calling the inference model of legal change and validity. Under the inference model, legal change is modeled by deductive inference. The authorizing rule of change (for example, the old AC) is one premise, the fact of enactment under the authorizing rule or procedure is another, and the validity of the new rule (new AC) is the conclusion.
>
> Rule: If act A is done, then rule B is valid.
>
> Fact: Act A is done.
>
> Conclusion: Therefore, rule B is valid.
>
> The theory is that the amendment is lawful if and only if this deduction is valid and its premises are true.
>
> Self-amendment occurs when the rule affirmed in the conclusion is meant to replace the rule in the premises, that is, when the rule in the premises refers to itself."

Suber demonstrates here that the logic of law should be both *recursive* and *non-monotonic*. We have already pointed out how our logics of choice are recursive, and we continue to discuss non-monotonicity:

[64] Is the famous old paradox of "Can God create a stone so heavy such that he cannot lift it?". Our personal take on this paradox, according to our current analysis of self-reference, is that this is indeed a good enough justification for rejecting the notion of something being omnipotent under all circumstances.

[65] Amendment Clause.

> "Irrevocable self-limitation is self-contradictory on this model because it requires an inconsistency between a premise and the conclusion of the inference, when the premises themselves are (or may as well be) internally consistent. The rule of change that sits in the premise is inconsistent with its own invalidation, or with the exclusive validity of its successor asserted in the conclusion. Alf Ross, the Danish logician and jurist, has argued in detail that by formal criteria such an inference must be invalid."

We have shown that these claims are in fact not true. Self-contradiction and inconsistency may be avoided indeed if we allow the conclusion to take the form: "conclusion: delete law number X". This ability to delete is a main characteristic of the PFP operator comparing to other logical fixed-point operators (e.g. TC/LFP/GFP/IFP), showing the rareness of a supporting logic indeed. And this is what non-monotonic logic refers to: the ability to retract assertions. Arguably, the easiest way to see a model for this case is space-limited Turing machines[66]. It is no different than the concept of "Self-Modifying code", and space-limited classes of machines under the halting condition of detecting same configuration twice, are indeed closed under self-modifying code[67]. Further:

> "...There may well be a way to dissolve the paradox for the inference model or for formal logic generally. (We will see that satisfying the inference model is more difficult than satisfying formal logic alone.) Such dissolution would take the form of removing the inconsistency between premise and conclusion in the inference that models the act of amendment. A dissolution of this kind would satisfy logicians even if it had absurd consequences for law; but I have not been able to find even one dissolution of this kind that stands up to analysis."

It is of no surprise that it is so hard to find a logic that supports the case of self-amendment: indeed almost all logics considered in the literature of mathematical logic, are monotonic. Moreover, almost all of them don't know to return the "undefined" value[68], a necessity demonstrated in Theorem 1 and by Protagoras' paradox. However we have shown that the self-amendment paradox may indeed be resolved under very mild assumptions being the laws of laws. In fact Suber doesn't consider decidability

[66] Supplying a model proves a logic to be consistent, and Turing machines are models just fine.

[67] This is trivial because allowing self-modification does not alter the halting condition a single bit. It is also trivial to implement self-modifying code in P-DATALOG by adding nullary relation symbols to enable/disable rules.

[68] Equivalently non-halting machines as in our derivation above.

here (while he should have), so any language with quoting and eval operators would resolve the paradox[69], in particular, the set of all Turing machines. Continuing:

> "We may distinguish two ways to "solve" a paradox. Paradoxes are not mere contradictions; they are not nearly as tame. But their bite lies in their way of making contradiction appear inescapable. One kind of solution is to escape contradiction by a path not seen by others. I am calling this a dissolution of the paradox."

Our solution is indeed a dissolution of the paradox, in contrast to:

> "Another way is to explain why contradiction is harmless, whether it is escapable or not. This will only work when contradiction is harmless, of course, and most of the argument in support of solutions of this kind will be spent in that cause. This will not dissolve the paradox, but will excuse and domesticate it, removing its sting and threat. It is much like what lawyers call an "affirmative defense": admit that you did it, but claim some excuse like insanity. The second kind of solution is more radical than it seems and requires, in effect, an insanity defense for law itself.
>
> In short, **my thesis is that no dissolution satisfies the terms of the inference model, but that when we reject that model we find a number of plausible domesticating strategies.** First I will consider various attempts to dissolve the paradox."

We exhibit here the difficulty of dealing with self-amendment, to such a great extent that philosophers even have to accept the existence of a contradiction, or in other words, to reject logic. Rejecting logic due to a crisis of inability to understand deep and important topics is not new[70], a situation which Suber rightly defines as insanity. Fortunately, on Tau, we do not have to appeal to insanity. Later on he continues to discuss the acceptance of a contradiction:

> "...In standard logic all propositions follow validly from a contradiction. Therefore, one solution might be deliberately to make the premises of the

[69] This is true even for non-monotonic logics since eval can recover non-monotonicity. Hint for the reason: Datalog+EVAL may delete by relying on its ability to have negated extensional symbols. In fact FO+EVAL equipped with the "not same state twice" halting condition is already equivalent to FO[PFP], as we have demonstrated how recursion may be simulated using eval.

[70] Even quantum mechanics rejects logic by coming up with a new and contradictory logic, cf. "Quantum Logic".

> inference that models the change internally inconsistent. This would certainly work to make the new AC a logically valid consequence of the old one. But it would work only at the cost of assuring that every AC is internally inconsistent. This result is inadmissible for the inference model, which cannot allow inconsistent rules simultaneous validity. However, if one likes, one may consider the total invalidation of the AC a solution satisfactory to logic that is repugnant only to law."

which is of course a case which we cannot allow. Suber does mention that he doesn't rule out the possibility of some logic to support self-amendment, but his unjustified pessimistic tone can easily be observed:

> "...Most philosophers who have approached the problem (primarily through the theological paradox of omnipotence) evidently hope that a solution can be found that does not violate or require the inapplicability of formal logic. I have not argued for any conclusion that should dampen that hope. **I have argued that formal logic has very limited application in law**, but not that a dissolution of the legal form of the paradox is strictly impossible by logical criteria. Hence I admit —without the same earnestness of hope— that there may well be a dissolution of the paradox satisfactory to formal logic. All I have argued is (1) that the obvious attempts at such dissolution fail, and (2) that in any case law can dispense with such a dissolution. The second point is the more important thesis, and stands (if it stands at all) even if a satisfactory logical dissolution should be discovered tomorrow.
>
> In this sense the state of self-amendment today is analogous to the state of calculus between Leibniz and Weierstrass. There may be a way to render it coherent and consistent, but so far we lack the theory to do so. Meantime we use it with good results. **The difference is that a coherent theory of self-amendment is unnecessary for law**, while a coherent theory of the calculus was vitally necessary for mathematics.
>
> If self-amendment would be logically impeccable if only certain niceties were observed in rewording the clause and in transacting its self-change, then a legal system may still ignore those niceties utterly. This remains the case whether the required changes are ingeniously simple or hideously complex. Today lawmakers listen to what logicians say they should do only to the extent that logicians form a weighty voting bloc in their constituencies. **There is no reason to think that their lawmaking acts will cease to be valid if the logicians should one day become correct, any more than those lawmaking acts are invalidated when the weightier loyal opposition is correct. Legal validity is a matter of power and social practice, not abstract correctness.**"

We again observe the theme of "let's live with the contradiction", but even worse: he claims that lawmakers' actions remain valid even in the presence of a contradiction! This explains a lot about the dystopia we live at. And this approach is provably dangerous: form a contradiction one can derive anything as Suber himself mentioned "In standard logic all propositions follow validly from a contradiction."[71]. Worse, the distinguishment between "what is good for logicians" vs. "what is good for lawmakers" is beyond the insanity which Suber referred to above, but one may even call it evilness. No one can reject logic, not quantum mechanics, not even the law, not even lawmakers and law enforcers. Let us not forget the many innocent lives destroyed due to incoherent judicial system.

> "...There will be no solution satisfactory to formal logic if formal logic bars all self-reference, say, by incorporating a theory of types, and if self-amendment unavoidably requires self-reference."

Indeed type-theory will recover the paradox of "laws of changing the laws of laws of changing the laws..." ad infinitum. This is one mistake in the old Tau design which was based on type theory and total languages. However here Suber is getting close to our solution:

> "Ross distinguishes between logical and legal contradictions. A logical contradiction exists between any statement and its negation. We don't have to assert one or both of the statements for the contradiction to exist. A legal contradiction exists between any two inconsistent laws that are both valid at the same time. If they are not both valid at the same time, they will be logically but not legally contradictory. Armed with this distinction, Ross has an answer to the most common attempt to dissolve the paradox of self-amendment."

Indeed a time aspect, which is PFP's staging aspect (which is not exactly "time" but is the same for this matter), solves the paradox. But this doesn't give rise to a distinction between "logical contradiction" and "legal contradiction", not a single bit. One only needs to find a logic that is suitable for law, one that can soundly model self-change, as our logics of choice. Suber continues on this point:

> "The most common attempt to dissolve the paradox has been to insure, or assume, that the old and new ACs are never valid and supreme at the same

[71] A.k.a. the "principle of explosion" or *ex falso quodlibet*.

> time. **Temporal overlap can be prevented; but preventing it merely avoids a legal contradiction, not a logical contradiction.** It keeps inconsistent rules from enjoying simultaneous legal validity, but thereby presupposes their logical inconsistency. That inconsistency invalidates the inference that models self-amendment because a logical contradiction between premise and conclusion[72] (when the premises are internally consistent) suffices to invalidate a deduction. As Ross often puts it, the invalidity consists in our attempt to derive from one norm a second norm inconsistent with the first."

which is indeed the correct point of view, and simultaneous write and delete in PFP amount to a contradiction (as in the Appendix D where we explain the halting condition). However Suber suggests that it doesn't prevent a logical contradiction, a statement which we do not agree with, as we have pointed out to a logic that soundly models the situation, and the inability to accept legal world which rejects logic. Suber continues for the case where legal and logical contradiction are not distinguished:

> "If one rejects the distinction between logical and legal contradiction, but still holds the inference model of legal change, then one is no better situated to overcome the invalidity of the inference that models the self-amendment. If the contradiction is supposed to disappear because the old AC loses validity at the moment the new one acquires validity, then in the inference that models this process the assertion of a key premise must be suspended in mid-inference. **Even if this successfully removes the inconsistency in the inference that models the self-amendment, it replaces it with a new fallacy. Logicians do not have a name for this fallacy because it cannot be performed in ordinary argument where the inference (conceived logically, as opposed to psychologically) is instantaneous or non-temporal**."

which basically argues that non-monotonic logic is illogical, a point which is of course not true. The mere fact that our logics of choice have a model as above, proves this.

We could have continued commenting on Suber's summary but for the sake of brevity we will save that opportunity for another time. For now let us compare Suber's Nomic game (appearing in the appendix of [6]) to our solution.

Nomic's approaches self-amendment by allowing the declaration of rules as either "mutable" or "immutable", together with the voters' ability to "transmute" a rule, namely

[72] X->not X.

to transform it from mutable to immutable and vice versa. In short, in every round, each participant may raise for voting either a new rule, or a transmutation of an existing rule, or an amendment of an existing mutable rule. However it does not account for:

- Large scale. What if we have a million participants, will we then have to wait a million turns in order for everyone to merely state their opinion?
- Decidability. How can we even infer whether one rule contradicts another in the absence of decidability?
- Logical coherence. At each point of time the current body of law may be logically coherent, but the process of changing the laws in Nomic is external to the laws themselves. Laws and lawmaking therefore don't come along under the same logical formalism, in Nomic.

and in fact Nomic solves nothing from the raised paradoxes. Indeed as Suber writes on the appendix:

> "...In Nomic situations may easily arise in which it is very hard[73] to determine whether a move is legal. Moreover, paradoxes may arise in Nomic that paralyze judgment. Occasionally this will be due to the poor drafting of a rule, but it may also arise from a rule that is well-drafted but mischievous. The variety of such paradoxes is truly impossible to anticipate. Rule 213 is designed to cope with them as well as possible without cluttering the Initial Set with too many legalistic qualifications. Note that Rule 213 allows a wily player **to create a paradox, get it passed (if the rule seems innocent enough to the other players), and thereby win**."

[73] Or even impossible in the absence decidability.

D P-DATALOG Language

We give a tutorial for how the proposed logic that fits the laws of laws looks like. As we mentioned, it's referred to as P-DATALOG on [1], and is equivalent to FO[PFP] over ordered structures. It also captures the complexity class PSPACE over ordered structures. It is also the core part of the language TML. TML is logically equivalent to P-DATALOG and contains P-DATALOG, while enhanced with conservative[74] extentions for convenience. But before we start describing this language, we first describe the language Datalog.

A Datalog program is a set of facts and rules, each rule comes with a head and a body. The head is the *consequence* and the bodies are the *antecedents*. A fact is a rule with no body, expressing that it holds disregarding other facts. Facts are also inputs to a program. So the rule "if it's raining then I should take an umbrella" has as body "it's raining" and as head "I should take an umbrella", and if the input contained the fact "it's raining" then the output will contain "I should take an umbrella".

Datalog operates over relational databases[75], namely the databases which are made of tables[76]. The order of columns on the tables matters while the order of the doesn't matter[77]. Each cell on the table can take values from some finite set, called the *Universe*, or *Domain of Discourse*. In principle the universe may be infinite but decidabllity is recovered only in the finite case.

Let us take the main example in the literature for a Datalog program, the transitive closure of a graph:

```
T(?x ?y) :- E(?x ?y); E(?x ?z), T(?y ?z).
```

This reads as follows: given a graph as a list of edges E(<vertex1> <vertex2>), then the resulted graph T is also a list of edges, where:

1. T(x y), namely there's an edge between x and y in the resulted graph, if E(x y), namely if this edge occurs in the input graph as well, or,

2. There's an edge from x to z on the original graph, for some vertex z, and we also already concluded that an edge from y to z appears on T.

[74] Namely such that keep the logical expressiveness intact.
[75] Which may be seen as a random-access machine with a multidimensional memory.
[76] A table with k columns is nothing but a k-ary relation.
[77] Namely a table, just like a relation, is a set of tuples.

The resulted graph T will therefore contain an edge between any two vertices that are reachable from one another on the input graph. Note that T and E are names of tables, also referred to as *relation symbols*.

An input to such a program may look like

```
E(1  2). E(2  3). E(3  4).
```

For this input, the output would be:

```
E(1  2). E(2  3). E(3  4).
T(1  2). T(2  3). T(3  4).
T(1  3). T(1  4). T(2  4).
```

Execution of a Datalog program is made by so-called stages in a manner known as "forward chaining"[78]. In each stage, each rule "fires" once and only once. A "firing" of a rule means that we substitute in the bodies of the rule all facts on the current database, and then we possibly infer new terms (given at the head of the rule) to add to the database. In the first stage, all facts (stated either in the program or in its input) are substituted in the bodies and we obtain a list of facts to add to the database. We add them, and only then we proceed to the next stage, and so on. The program terminated when there are no more new facts to add to the database, or in other words, where the database didn't change in two consecutive stages.

In Datalog we distinguish between extensional and intensional relations. A relation symbol appearing at least once in the head of a rule is called intensional, otherwise it is extensional. So on our example, T is intensional and E is extensional. Datalog allows negation over intentional symbols only. P-DATALOG is an extension of Datalog where negation may appear everywhere, including in the head or body of a rule. There are many ways to give semantics to Datalog programs with negation, most notably the Well-Founded Semantics (WFS) and stratified Datalog. Those two are PTIME-Complete, while P-DATALOG is PSPACE-Complete.

In P-DATALOG, negation in heads means *deletion*. Just like a positive head means to add records to the database, a negative head means the deletion of records. Negation in bodies naturally refers to all substitutions of variables that do not satisfy the relation. However for both negations in heads and in bodies, the order of execution of the program

[78]There are more ways for evaluating a Datalog program but we give the case which we can use in order to demonstrate P-DATALOG.

becomes significant. Negation in bodies is interpreted *per each stage*. So it might be the case where a negated term in a body is true at one stage, but not in a later stage. Similarly, deletion, by having negated heads, is done relative to the current stage only.

For example, on

```
E(?x ?y) :- E(?x ?z), E(?z ?y), ~E(?x ?x).
```

the variable ?x will bind to all values such that E(?x ?x) does not appear in the current-step database. Similarly

```
~E(?x ?x) :- E(?x ?x).
```

Will make the next-step database to not include all terms of the form E(?x ?x) included in the current step database.

Unlike in Datalog, a fixed point doesn't necessarily exist. The termination condition for P-DATALOG is therefore as follows. If the same term is inserted and deleted at the same stage, the program halts and evaluates to "false", or a contradiction. Otherwise it continues to the next stage again performing a single evaluation of every rule. This process must eventually halt in either of the following forms:

1. The database obtained from the current step is equal to the database resulted from the previous step. This is a fixed point. When this happens, the resulted database is considered as the final result.

2. Or, the database obtained in one step equals to the database state in some previous, not immediate predecessor, state. In this case the program will loop forever unless we detect it and halt the program, and is therefore evaluated to "undefined", as no fixed point exists.

Note that only one of the two options can happen because the arity and the universe size are fixed. Ultimately, for universe size n and maximum arity k, they will occur in no more than 2^{n^k} steps.

It should be remarked that the PFP halting condition and negation in bodies pose a serious performance difficulty, even reaching reasonable performance. We have mostly solved this problem by incorporating Binary Decision Diagrams, but the details are out of the scope of this paper. We shall only mention that Binary Decision Diagrams also give a very significant compression of data that typically arises during logical reasoning,

and by that may perform reasoning tasks that are far from being feasible without such a compression.

It should also be noted that the body may be an arbitrary first or second order logic formula. It is then still equivalent to P-DATALOG and is still PSPACE-Complete [1].

E Proof of Theorem 2

We assume that Q translates a formula into a relational structure representing a string which in turn represents the formula. Since the set of all formulas in FO[PFP] with or without Q+E may be given in a context-free grammar, recognizing whether a structure indeed represents a formula is expressible in FO[PFP] (can be seen easily by using e.g. "definite clause grammars", or by complexity-theoretic considerations). Given a formula in FO[PFP]+EVAL we replace all occurrences of E with a code that detects whether the structure is indeed a well formed formula, and if not, it returns 0. Otherwise, we convert that occurrence of E to pure FO[PFP] formulae. We sketch the next steps by example over the formula

$$F := R(x,z) \wedge R(z,y) \rightarrow R(x,y)$$

first:

$$Q(F) = R(0,x,z), R(0,z,y), R(1,x,y)$$

where the additional 0 and 1 tells us which is term is a body term and which is a head term, and additional such constants should be added in order to specify the "rule number" as in FO[PFP] taking the form of P-DATALOG (see Appendix D) but we omitted this and other small details in order to demonstrate the main idea, and the rest is left as an easy exercise to the reader. We also implicitly assume an additional relation V which tells us which symbols are variables and which are constants (this V should be added explicitly to the translated formula, but again, we only sketch the high level details here). Now the evaluation function

$$E(Q(F)) = E(R(0,x,z), R(0,z,y), R(1,x,y))$$

may be replaced by a formula G after applying the PFP operator to it:

$$G := \forall a,b,c,d,e,f,g,h,i,j.$$

$$R(a,b) \leftarrow R(1,a,b) \wedge R(0,c,d) \wedge R(0,e,f) \wedge R(g,h) \wedge R(i,j) \wedge T(a,b,c,d,e,f,g,h,i,j)$$

where T is defined by

$$T(a,b,c,d,e,f,g,h,i,j) \leftarrow (a=c) \wedge (b=d) \wedge$$

$$(c=d) \rightarrow (g=h) \wedge$$

$$(c=e) \rightarrow (g=i) \wedge$$

$$(c=f) \rightarrow (g=j) \wedge$$

$$(d=e) \rightarrow (h=i) \wedge$$

$$(d=f) \rightarrow (h=j) \wedge$$

$$(e=f) \rightarrow (i=j)$$

note that the resulted translation holds not only for F but to more formulas because T works for any choice of variables in the bodies. We can now replace each occurrence of $E(Q(F),R)$ with PFP[G,R]. We then convert the resulting FO[PFP] formula to P-DATALOG as demonstrated in [1].