A Promising Path Towards Autoformalization and General Artificial Intelligence

Christian Szegedy

July 2019

Abstract

The goal of autoformalization is to create a system that automatically learns to read natural language content and turns it into abstract, machine verifiable formalization. This is a very difficult task in general, one that would require strong automated reasoning and natural language processing capabilities by the AI system. Here, it is argued that autoformalization might be still the most promising path for systems to learn sophisticated, general purpose reasoning in all domains of mathematics and computer science. The successful creation of such a system could have far reaching implications not just for mathematical research, but also for software synthesis. The system could play the role of a strong, general purpose reasoning component in other AI systems. This paper provides an outline for a realistic path to reach the formalization capabilities of versed human mathematicians in the not too far future.

Contents

1 Introduction

Today, AI systems are able to learn solving tasks that used to be thought of taking uniquely human capabilities until recently: computer vision [77], generating artistic images [22] and music [30], mastering the game of go [70] and discovering novel drugs [24], to name just a few. These and many other domains seemed to require uniquely human intuition and insights, but were transformed by deep learning in the past few years. While progress has been impressive in those areas, each particular solution addresses a relatively narrow use case. On the other hand, general reasoning still seems a uniquely human feat and many [29] would argue that creating AI agents with general reasoning capabilities rivaling those of humans would take decades, maybe centuries, if possible at all.

This paper argues that in the next decade, we will see machines to surpass humans in general reasoning. Our best chance to achieve this is by creating a super-human mathematician first via autoformalization.

This document gives an overview of the hurdles involved, a realistic path ahead and indications on why that path is feasible.

This section starts with a basic introduction on why mathematics matters, what formalization means in general, how does it relate to programming and whether mathematical reasoning can be considered "general purpose".

Mathematics is the discipline of pure reasoning. In mathematics, we typically derive complex statements and create constructs based on minimal number of assumptions. It is like a small world, where everything is built from extremely simple and perfectly defined building blocks. Amazingly, even for very simple systems or processes, one can come up with statements that are substantial or look surprising. Moreover, simple looking statements can be extremely hard to verify or argue about.

For example the Collatz conjecture [53] states that if we start with any natural number n and iterate the following procedure:

- If n is even, divide it by 2,
- If $n = 1$, then stop,
- If $n > 1$ is odd, replace it by $3n + 1$,

then we always arrive at 1 in finitely many steps, regardless of the number we started at.

This is a very simple observation about a well defined process that can be easily formalized or executed in a computer. In fact, for any starting value n,

the correctness of the above statement can be tested by running a few lines of computer code in any sufficiently high level programming language. However, proving that this process terminates for all integer numbers escapes the abilities of the most ingenuous human mathematicians today.

Mathematics is the study of all those statements we can make about some system described in a non-ambiguous manner. One can think of "non-ambiguous" as something that can be given by a computer program. Essentially, every mathematical statement can be translated into the question whether some computer program ever terminates given unlimited computing resources. This is the so called "halting problem" [79]. The halting problem is not solvable in general. There is no program that can decide whether given a computer programs will it ever halt or just run forever.

Still, everything that can be specified precisely can be considered as mathematics and everything we can talk about precisely is what we can talk about it by mathematical arguments. Therefore, mathematics is the language of all things specified formally. That is why it underpins all practical scientific disciplines: computer science, physics, chemistry and, increasingly, biology as well. The more we understand a discipline, the more formally we can describe its objects and phenomena. The more mathematical a discipline becomes, the deeper the truths are that we can derive about it and the higher our confidence can become about their correctness.

Mathematical reasoning is not about mathematics per se, it is about reasoning in general. Whether we want to verify the correctness or resource use of a computer program or derive the consequences of a physical model, it is all mathematical reasoning, as long as it is based on fully formalized premises and the general rules of logic. Some mathematical tasks require very simple reasoning, others, like the verification of computer chips, may require extremely complex, but still relatively straightforward reasoning. These tasks might require such a large number of logical steps that humans find it impossible to check manually, but often they are easily solved by SAT-solvers [7] – programs whose sole goal is to decide if a Boolean expression can ever evaluate to true.

For certain classes of expressions, like those that occur frequently in chip design, SAT solvers work remarkably well[18]. An extreme demonstration of their power is their use in the computer generated proof of a previously unsolved famous conjecture in mathematics [35] – the Boolean Pythagorean triples problem. The final proof was 200 terabyte long.

However, SAT solvers do not always scale for all kinds of reasoning tasks efficiently. For example the verification whether the floating point unit of a chip works according to the specification might require higher level of reasoning in order to be performed efficiently [33].

Also, SAT-solvers cannot verify statements about infinitely many cases. For example, they can't even verify that the addition of integer numbers is commutative, as it is a statement about infinitely many possible pairs of numbers. There are automated methods (so called ATPs [20]) for finding moderately difficult proofs in first order logic and can handle such cases. Additionally, there is existing proof automation (via so called "hammers" [42, 8]) for higher order logic as well. However, most existing proof automation is based on hand engineered heuristics, not on machine learning and is not capable of open-ended improvements by itself.

This paper proposes a completely different approach to tackle the problem of automated reasoning. The proposed approach relies heavily on recent advances in deep and reinforcement learning. We aim for creating a system that is capable of open-ended self-improvement and tries to avoid the use of hand-crafted features or algorithms for special purposes. The more generic is the design of the system, the more freedom it has to improve itself indefinitely.

Mathematical reasoning is just reasoning about anything formal. Reasoning about anything formal is a powerful general tool. If we want to create an artificially intelligent system and demonstrate its general intelligence, it should be able to reason about any area of mathematics or at least it should be able to *learn to do so* when given enough time. If that is the case in practice, then we can be convinced that it is likely that it will be able to learn to cope with any scientific disciplines as well, once they are formalized precisely.

Of course, not all areas of mathematics are equal. Human mathematics consists of a large variety of loosely connected domains, each of them having its own flavor of proofs, arguments and intuition. Human mathematicians spend years studying just to become experts in a few of those domains. An artificial system engineered to produce strong results in a particular area is not a "general purpose" reasoning engine. However, if a system demonstrates that it can learn to reason in virtually any area that it exposed to, then that would be a convincing demonstration of artificial general intelligence.

Therefore it is natural to ask: Will we ever arrive at the point that an AI agent can learn to do reasoning as well as the best humans in world in most established domains of mathematics.

2 What is (Auto-)formalization?

The task of formalization is to turn informal descriptions into some formally correct and automatically checkable format. Examples of human mathematical formalization include the formal proofs of the Kepler conjecture [31], the Four-Color theorem [25] and the Feit-Thompson theorem [26]. These formalization

works required a lot of effort. For example the formalization of the Kepler conjecture took over 20 man-years of work.

More generally, "formalization" can refer to any process that takes an informal description for input and produces machine executable code. By this definition, formalization covers both programming and mathematical formalization. This generalized notion is justified, since computer verifiable proofs are in fact programs to feed some verification engine. The HOL Light [32] and Coq [13] proof assistants (which were used in the above large scale formalization works) are full blown programming languages. The proofs are given by "tactics scripts" that create long sequences true statements connected by elementary inference steps, where the steps are checked by a small trusted kernel. Once the theorem proved is produced by the kernel, it is "proven". The fact that the kernel is small and unchanged allows for extending the system by arbitrarily complex subroutines without the danger of compromising the correctness of proofs.

Another, albeit much less practical, correspondence between proofs and programs is the Curry-Howard isomorphism [37] which serves as theoretical foundation for some of the proof assistants, like Coq, but resembles much less of real life programming than the above mentioned "tactics scripts" that produce the sequence of elementary inference steps.

Just like programming, mathematical formalization is a cumbersome, task. Formalization requires deep programming skills that most mathematicians are less inclined to focus on. Complex mathematics is especially time consuming to formalize even for those reasonably skilled with proof assistants. Therefore, it is highly unlikely that a significant portion of mathematics will be formalized without significant automation in the coming decades.

That raises the question whether formalization could be automated. The ideal solution would be a system that could formalize directly from natural language text of some mathematical topic fully automatically, without minimal intervention from the user. This would allow for formalizing most of known mathematics without the need for intervention from mathematicians.

We call an automated system that is capable of automatically formalizing significant portions of mathematics from a natural language input and verifying it automatically an **autoformalization system.**

3 Why is Autoformalization Essential?

Is targeting autoformalization really a prerequisite for training – and evaluating – AI systems for general purpose reasoning?

As was argued in the introduction, all formalizable reasoning can be viewed as mathematical by nature. Conversely, a general purpose reasoning system should be able to (learning to) reason about **any** domain of mathematics and should be able to discover new mathematical domains when needed or useful for another task.

Avoiding autoformalization (communication in natural language) would certainly simplify a system the main objective of which is formal reasoning. Formal reasoning is a hard task in itself and autoformalization just adds a seemingly unrelated extra complication which seems even harder to engineer. Obviously, we would like to get away without if it is possible. However, if we want to tackle mathematical reasoning and evaluate such a system, we are faced with three related very hard problems:

1. Evaluating a purely formal system would require a wide range of formalized statements which we can set as target task in our prover to train and evaluate it.
2. Any interaction with our system would be by completely formalized means. If the system develops its own web of definitions (about which it does not need to communicate in natural language), it will resemble alien mathematics that is very hard to decipher.
3. Every time the system needs to be applied to a new application domain, it would require full-blown manual formalization of that domain. This would limit the usefulness of the system significantly.

Training a mathematical reasoning system without autoformalization might be still possible if one can develop a concise, well defined notion of "interestingness" of mathematical statements that is used as the objective for open-ended exploration. However it would be very hard to communicate the system as it is not rooted in human mathematics.

However, "interestingness" is not an easy to define notion. It is hard to tell whether some mathematical area will ever have interesting applications or would provide insights for other domains down the line. Whatever was considered interesting 100 years ago might look irrelevant or trivial today. Interestingness is in the eye of the beholder and is highly contextual. There is no known way to guide a search process automatically towards interesting theorems and notions.

Therefore, the safest option is to use the entirety of human mathematics as a basis for training and benchmark for testing. Since only a miniscule portion of mathematics is formalized, the only way for utilizing a significant fraction of accumulated human mathematical knowledge is by processing it from natural language.

What is argued here is that it is easier to engineer and train an AI agent that can reason and formalize at the same time than one designed for just reasoning or formalization. One can expect positive feedback loop between reasoning abilities and formalization capabilities. Improving one aspect of the system should help collecting new training data for the other:

- Better reasoning allows for filling in larger holes in informal arguments.
- Better translation to formal statements expends the amount of data for guiding mathematical exploration.

4 The Implications of Successful Autoformalization

Autoformalization is not just a challenge: successful autoformalization would be a huge breakthrough for general AI with significant implications.

Autoformalization would demonstrate that deep and sophisticated natural language understanding and exchange is feasible and that machines can communicate in natural language over ambiguous content and internal experiences. Essentialy, it would be the first demonstration that natural language is a feasible communication medium for computers as well.

By nature, autoformalization would have huge immediate practical implications for mathematics. Initially, a (close to) human level autoformalization system could be used for verifying existing and new mathematical papers and would lead to strong semantic search engines for mathematical content. However very soon after a successful implementation of such a system, we can expect human or super-human level problem solving skills to emerge and the role of human mathematicians would shift from puzzle solving towards posing interesting questions and giving high level directions and guidance for the system.

In the more general sense of formalization, a solution to autoformalization gives rise to programming agents that can turn natural language descriptions into programs. Since programming languages can be formalized completely, reasoning system trained on mathematical formalization could be fine-tuned for the task of creating algorithms in that particular programming language. By formalizing domain level knowledge, the system could be made produce code from natural language input. It would the formal specification of the task, the executable code and correctness proof of the newly designed algorithm, all at the same time.

These capabilities could revolutionize programming and change it fundamentally. Humans could communicate with machines in more concise natural lan-

guage. This would boost productivity while democratizing the interaction with computers: everybody with good ideas would be able to create complex and safe systems quickly, without the help of expensive IT specialists. It would lower the barrier of entry for new businesses and could usher a new area of creativity where imagination becomes the deciding factor for success.

Most importantly, this solution would give rise to extremely strong general purpose reasoning engines that could be integrated into AI applications that combine reasoning with perception. One could infuse strong reasoning capabilities into other systems and serve as a basis for a wide range of applications.

5 The Hurdles of Autoformalization

While autoformalization would have far reaching implications to research and productivity, it is not easy to design and implement. This is subject of current research efforts. However, given its potential breadth of applications, it is worth investing a significant effort into.

In order to study the difficulties of designing a general purpose autoformalization solution, let us start with a naïve attempt at its construction, based on the following two components:

1. A reasoning engine (theorem prover),
2. and a translation module that attempts to translate informal (natural language) statements into formal statements.

We assume that the translation module can generate multiple formal candidate statements (maybe taking the history of past unsuccessful formalization attempts into account). The system is successful if it can formalize a large fraction of the informal statements after a reasonable number of attempts.

The most direct attempt at the translation system would be a solution that that learns programming from natural language directly by creating programs based on their natural language description. However any obvious attempt at this would require a lot of curated data, since it is hard to create a system that improves this model in a mostly unsupervised manner. Either the generated program does what it was specified to do, in which case the translation is already working, or it generates the wrong code, but there is no clear learning signal if the code does not work according to the specification. If provided only by natural language input, the system cannot distinguish between these two cases and therefore it cannot be learned or bootstrapped without a being given a lot of ground-truth programs

Given the seeming impossibility of bootstrapping programming from natural language alone, we will explore the option on bootstrapping mathematical reasoning from natural language.

Can one bootstrap mathematical autoformalization with limited engineering resources in a semi-automatic manner? The first problem is that it seems to require at least initial core formalization data-sets with a significant amount of parallel corpus in formalized and informal mathematical content. One limiting factor is the cost and effort of generating this seed corpus of translations. However in Section 7.5, we will discuss ways to circumvent this issue.

Let us assume that we have a somewhat working "seed" translation model. Then one can try bootstrapping and training the system by generating several candidate translations and trying to prove/refute each of them until we find a formalization that is correct, but not trivial. Still this system needs a working translation model that has a significant chance of producing the correct formalization, also it requires specifying the notion of "not trivially correct" which is a bit fuzzy.

So there are at least four major obvious potential failure modes of autoformalization:

1. The seed formalization system is too weak to initiate a feedback loop that can open-endedly improve itself.
2. Given too weak a criterion for filtering the resulting formal statements, the system might start to generate mistranslations for further training of the translation model, ending up with a feedback loop perpetuating increasingly worse translations.
3. Translation gets stuck: it would generate a lot of incorrect statements that can never be verified and eventually the system will slow down and stop to improve.
4. The translation never crosses domain boundaries: it is possible that the system manages to formalize significant parts of a certain domain, but the difference between this domain and another new domain is so big, that the translation never succeeds in generalizing to the new domain. So the training of the system gets stuck after formalizing limited parts of the corpus.

The main weak point of our naïve system design is that it creates formal statements directly from informal descriptions. Natural language is very context dependent: mathematical text might contain hidden assumptions in far away parts of the text which are impossible to find without a thorough search. For example, papers tend to refer to text books for "basic terminology". This means

that the formalization system would need to look up the textbook and mine it for all the subtleties of its definitions and verify that those definitions are consistent with those in the repository of the formal system. If not, then the system would need to create new formal definitions that match those in the paper. However this might be impossible, since the paper itself might use inconsistent notations – "abuses of language" – meaning that the same notion might have subtly different meanings in various parts of the text. In addition to that it might just have plain errors that are obvious to the human reader. This means that a simple translation is not a robust solution and is unlikely to work in practice.

6 A Proposed Path to Autoformalization

Although it is hard to avoid or rule out all the failure modes from the previous section, one can still design a system that is more robust to the various error modes and it is more plausible that this robust system can be bootstrapped as opposed to the solution that relies on direct translation attempts into formal statements directly.

Here it is proposed that instead of a direct translation, one should rely on a combination of exploration and approximate translation. By approximate translation, here we mean a translation model that does not try to produce a concrete formal transcription as a statement, but instead approximates some embedding vector thereof.

One engineering question is how to represent informal mathematical content, that is mathematical papers and text books we want to formalize automatically. The straightforward looking path would be to represent them textually, for example by a sequence of unicode characters. This might work well for use cases that do not require the understanding of formulas, diagrams and graphs. However mathematical content is heavily based on the use of formulas and also diagrams and geometric illustrations might play an important role in informing the reader. Therefore the best path seems to rely on images instead of textual representation. Given that humans work from the same representation, this can preempt a lot of unexpected failure modes and engineering that would go into reconciling various types of textual representations of the same input.

In order for a precise description of the proposed system, we introduce the following notions: S will denote the set of syntactically correct formal mathematical statements in some formalization environment (e.g. HOL Light). By S', we denote those statements with a formal proof. C is the set of possible forward reasoning rules called "conversions". C consists of transformations $c : S' \times S'^* \longrightarrow S'$ of statements that given a true starting statement $s \in S'$ and a list of other true statements $p \in S'^*$ (list of conversion parameters), it

generates a new true statement $c(s, p)$. We assume that the informal description of each statement is given by a picture $\mathbb{R}^{k\times l}$. One can think of that as the raw image of a page in some mathematical paper. In addition, we will use a fixed size embedding space for both the formal and informal statements. Our system will rely on several deep learning models:

- A formal embedding model $e_\theta : S \longrightarrow \mathbb{R}^n$ that embeds formal mathematical statements into an embedding space of dimension n.
- An approximate translation model $a_\xi : \mathbb{R}^{k\times l} \longrightarrow \mathbb{R}^n$, that outputs an approximate embedding of the formal translation of the informal input statement given as a picture.
- An exploration guidance model $g_\eta : S \times \mathbb{R}^n \times \mathbb{R}^n \longrightarrow [0, 1]^C \times [0, 1]$. This model acts as a premise selection model, combined with a conversion type prediction, assuming a finite number of possible conversions. $f_\eta(s, t, p)$ which takes a formal statement s, a target embedding t and a conversion parameter embedding p and tries to predict the probabilities of next best conversion steps $[0, 1]^C$ and score a potential conversion parameter p at the same time. The exact working of such models is described in [6].

For technical simplicity, in this section, we made the simplifying assumption that statements and input images are represented by fixed dimensional spaces. This assumption is made in order to promote formal clarity of the approach. However, a practical working system will be most likely some expansion of the outline presented in this section and would use variable size input and variable dimensional embedding spaces. This is irrelevant for the core idea. Later sections will give more refined considerations to various aspects and variations for various components of the system.

The parameters of the deep learning models e_θ, a_ξ and g_η that are trained in lock step as described below.

Our system will learn to explore true statements guided by target embeddings and attempts generating automated formalization of informal statements – in form of the approximated formal embedding vectors of their transcription – at the same time. Training and inference is not decoupled in our setup. During this process we will maintain and update the following data sets:

- A fixed set of target statements in informal format $T \subseteq \mathbb{R}^{k\times l}$ in raw image format. These are the pages the contain the statements given informally that we are aiming to formalize.
- The image of T'_ξ of T under the approximate translation model: $\{a_\xi(t)|t \in T\} \subseteq \mathbb{R}^n$, the predicted embeddings of the informal statements on the formal side.

- A set of already explored mathematical statements $D \subseteq S'$ of true and proven statements,
- The embeddings of the explored mathematical statements: $D'_\theta = \{e_\theta(s) | s \in D\} \subseteq \mathbb{R}^n$.

Our goal is to find a subset of $D' \subseteq D$ whose image under e_θ aligns well with T'. That is the goal is to hit the target embedding vectors of T' with (non-trivially) true formal approximations. The intuition is that if our translation model is reasonably good, then true – but non-trivially true – translations are likely to correspond to the informal description (where the target embedding came from). During our process we aim to train all our models while updating the datasets T', D and D'.

On the formal side, we have an embedding model e_θ that embeds mathematical statements in a way that encourages embedding vectors of semantically similar statements to be close in the embedding space. Formally, if S denotes the set of formal mathematical statements we encounter, then $e_\theta : S \longrightarrow \mathbb{R}^n$ should be a machine learning based model (probably some kind of deep neural network) parameterized by θ. This model is trained for one or multiple particular semantic purposes, for example e_θ could be trained as the premise tower of a network trained for premise selection: that is it embeds statements for the decision whether the statement is immediately useful for proving another specified statement. See [2] and [6] for examples of such models. Training those networks is possible with a combination of imitation and reinforcement learning by training on successful proofs. Section 7.3 proposes a large selection of potential self-supervised training tasks that could also be used for training e_θ.

On the informal side, we have a computer vision model a_ξ that takes the informal statement represented as a picture $p \in \mathbb{R}^{k \times l}$ (or a sequence of characters) and tries to predict $a_\xi(p) = e_\theta(t(p))$, where $t(p)$ is a hypothesized correct formalization of P. One nice side-effect of this approach is that since e_θ is assumed to be an embedding model that reflects semantic similarity, t does not need to be a proper function: typically, there are multiple correct formalizations of the same informal statement, the embeddings of which should cluster in $\mathbb{R}^n$ and therefore their approximate embeddings should be near to each other and predictable.

Our goal is to create a feedback loop between training θ and ξ, the parameters of the models that control the formal statement embedding e_θ and the approximate translation model a_ξ (while also training η, the parameters of the guidance, but that is not a strict necessity or could be done independently).

To this end, we would maintain a set of proved theorems, a large set of informal statements P and translations $T_\xi = \{a_\xi(p) | p \in P\}$ of approximate translations of informal statements. In order to generate training data for training our translation and embedding models, we run guided exploration by sampling forward

reasoning steps using another deep neural network g_η from our already proved theorems with the goal of getting close to as many of the approximate translate embeddings T_ξ as possible. For this purpose, η is trained via reinforcement learning by using a reward function based on how close do we get to some of the target embeddings. g_η is based on model that can sample both conversions and conversion parameters ("premises" used for the conversion).

Note that g_η should be relatively easy to train using random exploration as well, as whenever we create a set of conversions with the model, we can pretend that the embedding of the final statement was our initial goal to start with. This idea is known as hindsight experience replay and was introduced in [3]. This way, one can generate a lot of data with high reward without having to cope with sparse reward issue of theorem proving for unsuccessful proof attempts.

Once our guided search finds enough statements that match some of the prescribed embeddings in T_ξ, we would check that they are non-trivially true and use those as new supervised data for retraining a_ξ. As we go along, we can incrementally train e_θ and g_η as well. For example e_θ could be trained by analyzing the dependency structure of the explored statements (the tactic parameters that led to the new statement), while g_η is trained using some reinforcement learning mechanism using the rewards collected during exploration.

This is the rough outline of the overall idea just in order to highlight the most crucial part of the system. In general, one needs much more implementation details for a working system, but this setup already has a lot of visible advantages compared with the naïve autoformalization variant that would try to generate formalization candidates directly.

One advantage is that this system is much more robust to errors: if exploration is powerful enough, then formalization might work even if we fail to translate some of the statements properly. Also, if the formalization gets stuck, the system might just relax the distance with which accepts autoformalization attempts, however this might become risky in the long run.

Still, we would accept that this system should be able to generalize to completely new domains, as exploration is probably more efficient in the early stages, that is when a completely new domain is explored, since the basic facts are easier to find. This early exploration can bootstrap the easy parts of the system and can give enough information to the translation model so that it can continue bootstrapping successfully.

In the Section 7, we present a set of possible refinements that are not obviously critical for the functioning of the system, but can have a high potential for improving the quality of the system and increasing the chances of success. It is subject to future research which of those ideas are helpful or necessary for the system to bootstrap itself.

7 Further Ideas and Considerations

The last section has given a very rough outline of a system that has the potential of bootstrapping itself for mathematical reasoning via autoformalization. However there are a lot of additional possible details that might or might not be essential for such system to work well. These details are more arguable than the general outline from Section 6 and are to be likely revised in light of insights and later experiments. Still, here is a selection of the most interesting considerations that could help engineering a system described in Section 6.

7.1 The Choice of Foundation and Framework

Traditionally, many people would argue that the choice of the right framework and foundation is essential for the success of a formalization project.

There is some truth in that for human users of interactive proof assistants the right framework can affect the productivity of formalization, but generally these effects are hard to quantify and there had been various types of logics and foundations that have been successfully applied in large scale formalization efforts:

- The Mizar system[60] is the oldest system in this list and has been used successfully for formalizing significant areas of mathematics. Although the most impressive formalization efforts has been performed using HOL Light and Coq, Mizar has the largest and most diverse library of formalized statements. Mizar is based on a first order foundations using the axioms of Tarski-Groethdieck set theory [75]. A unique feature is the use of syntactically weak types that allows for extra flexibility. Axiom schemes are also required for axiomatizing set theory, in lack of higher order logic. The Mizar system and library has been in development since 1973 and gave rise to the Mizar mathematical library that contains close to 60000 formalized theorems.
- HOL Light [32] uses a higher order logic based on simply typed lambda calculus with generic types. Using very similar foundations as those of the HOL4 [72] and Isabelle [88]. HOL Light has extensive mathematical libraries for arithmetic, linear algebra and multivariate complex analysis. HOL Light was successfully used for formalizing the proof of the Kepler conjecture [31]. Overall, the formal proof of the Kepler conjecture and the foundational libraries contain about 30000 theorems. These theorems and their proofs have been made easily accessible for AI researchers in the HOList environment [6].
- Coq [13] is a higher order logic theorem prover that is based on calculus of

inductive constructions [14] which is a dependently typed lambda calculus. In its basic form, this results in a constructive (non-classical logic) making the formalization of certain non-constructive concepts harder. Coq was successfully used for formalizing the proofs of the four color theorem and the Feit-Thompson theorem. The Coq based theorem proving environment CoqGym [92] comes with over 70000 theorems.

- Isabelle [88] uses classical higher order logic. It has a theorem library consisting of tens of thousands of formalized and formally proved theorems.
- HOL4 [72] has a logical foundations similar to HOL Light and Isabelle and has a theorem library containing about ten thousand theorems.
- Metamath [59] is based on a very simple formula rewrite system. Its theorem library contains over 23000 theorems. It is not extended with tactics or automation, but the relative simplicity of the system is alluring. Metamath was used in the first (albeit simplistic) deep reinforcement learning based automated prover [89].

We have only listed proof assistants that have demonstrated a significant amount of successful formalization efforts: tens of thousands of theorems, some of them of great complexity.

There are other logical frameworks that has been been proposed for the formalization of mathematics. For example univalent foundations based on homotopy type theory is an interesting direction [63], but did not garner significant traction in practice. Currently the Lean theorem prover [16] is gaining popularity. Lean has very similar foundations to that of Coq, while not having a significant library of formalized mathematics yet.

By and large, it seems that all foundations: slightly extended first order logic with set theory, as well as classical and non-classical higher order logic foundations are suitable for formalizing extremely complex mathematical content. Most researchers involved in large scale formalization efforts agree that the most important factor for successful formalization is the availability of existing theorem libraries. Even for human formalization efforts, it is not likely that the change of theorem foundation has a significant effect on performance or formalization speed as long as

- the system is expressive enough,
- can be easily extended with automation,
- and has a large enough library formalized theorems to be used as premises.

Still a few considerations apply when it comes to automatically formalizing from natural language. Theorem libraries based on purely constructive non-classical

logic might have a significant mismatch with most mainstream mathematical text for non-constructive mathematical objects. For example the real numbers in Coq are introduced quite differently than in most text books. This might be of concern when trying to match natural language informal mathematics from text books with their formalized counterparts, especially in the initial stages of the bootstrapping process. Similar mismatch might affect formalization attempts based on univalent foundations. It might require a significant extra translation efforts to account for those differences in logic. As the autoformalization system becomes sophisticated enough, the effect of such mismatches should be increasingly mitigated by the higher level reasoning capabilities of the system.

Another issue whether the proof assistant can be extended with new algorithmic methods (new "tactics", for example). Also the engineering efforts required to interface with external libraries, especially machine learning systems is a point of consideration. This is a big handicap for Mizar, as its unclear and restrictive licensing causes concerns. Also, it is written in Pascal that has outdated support for interfacing with modern systems and it seems harder to extend by new methods.

One last concern is the expressiveness of the logic. Although first order logic is generally capable expressing the Zermelo-Fraenkel axiom system, upon which most of mathematics is based, it can only do this with the help of axiom schemes, which is adds a lot of extra unnatural book-keeping. Also some informal mathematical content is expressed most easily by quantifying over propositions and functions. First order logic does not allow for that directly. This is a slight handicap in certain situations. However, a strong enough AI system can be able to learn to cope with those deficiencies. If the system is designed for open-ended improvement, then only the initial phases of the bootstrap process are effected by such restrictions of the foundations.

Based on these considerations, HOL based proof assistants (HOL Light, HOL4 and Isabelle) seem to be the most proven and best suited as the basis for the formalization environment of an autoformalization system due to

1. the significant size of their theorem libraries,
2. the classical nature of their logic,
3. the extensibility of their system,
4. the higher order of their foundations.

7.2 Unsupervised Pretraining for Informal Input

It was argued in Section 6 that it is best to formalize directly from images rather than textual representations. On the other hand self-supervised pre-training on

vast natural language text corpora [64, 17, 93] has been key to significant improvements on a large number of natural language processing tasks [85]: machine translation, question answering [66], sentiment analysis [73], natural language reasoning [98], just to name a few. It is reasonable to assume that all those task bear relevance to autoformalization as strong autoformalization should require strong natural language understanding capabilities.

One could be also tempted to think of autoformalization without aligned corpora as an unsupervised machine translation problem that could be tackled by similar methods to unsupervised translation in natural languages [4], however this would ignore the fact that formal representations are much more different to any natural language mathematical text than other natural language corpora. Also, small errors in translation would render the translated text to be useless.

In general, we would like to reuse the representation capability of language models trained using BERT [17] style self-supervision, but still retain the possibility from training for raw images. One idea for accomplishing this is to OCR textual parts of the image with informal mathematical content, embed the resulting pieces of text and train computer vision models that learn to reproduce those embeddings (e.g. via minimum square error regression). This seems like a strong pretraining methodology for the text part of the image model for formalization, but would ignore the formulas and graphs.

In order to embed formulas more succinctly, we could reuse some of the ideas for text embedding and further pretrain our formalization image model by some new tasks based on images:

1. Skip-token for formulas: cover some part of the formula with solid rectangles and predict the covered part.
2. Context prediction for covered context: OCR the context (some surrounding sentence) of formulas and try to predict NLP-model embeddings of the covered sentences.
3. Continuation prediction: Cut an image in half with a horizontal line (at some empty space) and predict which one is the correct continuation out of a few possibilities.

This is just a small selection of possible tasks to a give a taste of how one could devise self-supervised pretraining for raw image representation of mathematical content that relies on OCR and pretrained language models.

Based on the success of pretraining NLP models on vast amount of text, it is natural to assume that extensive pretraining of formalization models would be instrumental for capturing inherent statistics and would result i more semantic embedding of methematical content in a way that promotes stronger formalization performance, especially in the initial stages of a bootstrap process.

7.3 Unsupervised Formal Pretraining Tasks

Here we address self-supervised pretraining of models on the formal corpus of mathematical statements. At first sight the seeming lack of formal training data might make this idea to be less promising. However, this would ignore several possibilities to acquire a lot of data cheaply:

1. In HOL Light and other higher order tactic based theorem provers, the number of elementary proof steps can be magnitudes greater than that of the high level proof steps. [40] Especially long human proofs that can't be easily reproduced by automated means can result in a lot of formulas. This gives us a huge number of true and relevant formulas to train on by self-supervised methods.
2. It is possible to generate a lot of new true statements by augmenting existing statements by trivial transformations like variable changes or changing the order of variable bindings in lambda expressions, etc.
3. Forward conversions (rewriting statements by equalities that match parts of existing formulas) can generate a large number of true new statements.

Although the diversity of formulas can be negatively effected by these approaches, they would still allow self-supervised models to be trained on a huge number of formulas.

One distinct advantage of formal representations that there are much more natural self-supervised task for them than for the relatively unstructured natural language text. Here, there is a wealth of highly semantic tasks that has the potential for creating strong semantic embedding vectors of formulas. Here is a small taste of tasks could giving rise to interesting and non-trivial neural representation of mathematical formulas.

1. Skip-tree model: A higher order logic formula can be naturally represented as a syntax tree. We can remove some random subtree and either predict some embedding of the missing tree or train a model in a contrastive manner by trying to select the correct subtree from a set of candidates (sampled from the same or other formulas in the corpus). Hard negative mining should come handy for contrastive training.
2. Type inference model: Learn to do (partial) type inference of formulas. There are several variations that range from predicting the types even if type information is fully erased from the syntax tree to predicting the type of individual nodes, when the task can be solved by standard type-inference methods.

3. Predicting the embedding of useful lemmas: those that could be used for simplifying or proving the statement. Or just predict if some identity could be applied in a rewrite rule to change the statement. This could be trained as a discriminative model or as a generative model for predicting the embedding of useful identities, even if they do not occur in the corpus.

4. Predicting the result of rewrites: Given an identity and rewrite rule, try to predict the embedding of the result of the rewrite. Alternatively pick the correct result from a set of candidate results.

5. Context prediction: Given a subtree, try to predict the embedding of the surrounding tree, or pick the containing tree from a list of candidates.

6. Rewrite sequence prediction: Rewrite a formula with a sequence of rewrites, try to predict the sequence of rewrites that has lead to the result.

7. Generative models: try to generate true formulas from scratch or terms that can constitute useful substitutions for simplifying theorems.

Many other tasks are conceivable and it is likely that most of those tasks would be helpful to create stronger semantic embeddings for formal statements. Embedding models trained on semantics task could be useful in proof search, guided mathematical exploration and in formalization.

7.4 Synthesizing Mathematical Formulas

As mentioned above, generating new mathematical formulas from scratch could be considered as an unsupervised task for better representations. For example one could train generative models (recurrent [74], convolutional [15] or transformer [84] based neural networks) that predict new theorems token by token as a sequence the same way it is done in language modeling.

A somewhat less standard approach would be to combine the above idea with using variational autoencoders [49]. This can be combined with methods to create syntactically correct sentences if the syntax is known, such as in our application [52]. The advantage of VAE is that it learns both encode and decode to a compact representation, however it is known to be tricky to implement efficiently for discrete outputs, like text or graphs.

More direct (auto-regressive) generation might be of use as well, like in [94].

Besides giving rise to presumably useful representation as pretraining, there are two other important application domains of this approach:

1. conjecturing,

2. generating terms used in proofs.

Conjecturing could be useful as part of theorem proving: coming up with other helpful theorems that are true, could serve as a lemma for the current statement to be proved while being simpler to prove.

Also conjecturing could be useful for guided exploration by sampling statements that are true (at least in a decent fraction of the cases), while getting closer to some target embeddings. Especially variational autoencoders could be suited for that purpose.

As we saw in Section 6, stronger guided exploration is an important goal, in fact: a potential enabling factor for our guided exploration based autoformalization method. Still, it is unclear how well direct formula generation methods would do on the task of generating true formulas. It is far from clear that the success-rate of this approach would be useful for any particular purpose.

Another critical application would be the generation of simple terms that can be useful in proofs. It is important to note that such terms do not need to be "true" in order to be useful. In fact, they do not have to be propositions, just have the right type to be substituted into some mathematical expression. For example if have a theorem like $\forall x, y \in \mathbb{R} : x + y = y + x$, then we can substitute arbitrary terms into x and y as long as their types allow it. For example, we can deduce $\forall a \in \mathbb{R} : a + 1 = 1 + a$, by substituting 1 into x, and a into y. Substitutions with relatively simple formulas – that are generated based on the use case – can be essential for simple specialization or just fitting the correct variable names in proofs. However, sometimes the using the correct term to substitute may be very tricky, for example the simplest proof of the famous Nullstellensatz by Hilbert [97] relies on an inspired substitution.

In short: direct generation of formulas by generative models (autoregressive, VAE, GAN,... etc.) may or may not be useful for generating more complex true statements, however it seems to be highly likely to work for generating simple terms for substitutions in certain context.

For exploration and conjecturing, it looks more realistic to generate approximate embedding of the conjecture by some direct model and then using reinforcement learning to find true formulas that matches that embedding as closely as possible, as was proposed in Section 6.

7.5 Bootstrapping the Translation Model

Section 6 has proposed a master plan for addressing autoformalization. The weakest point of that plan is to create a translation model that manages to map informal descriptions (given as image) into embedding vectors of formal

statements. Although the overall approach was designed to be relative robust with respect to the quality of the output of the translation model, it is still questionable on how one could start bootstrapping such a model. Here we present multiple ideas that might be of use.

In the very early stages of the bootstrap process we could just start with a rough translation based on bag of word models that tries to match sentences on both sides based on a matching between tokens. Here the idea is that our initial formal corpus contains a very limited set of definitions (less than thousand) that could be translated to their informal counterpart. This could be done with not too much effort even if certain notions might map to multiple informal expressions (for example "orthogonal" and "perpendicular'). Once such a mapping is established, we could just use a bag of word model as embeddings on both sides. In this case, the translation is just given by a word-level translation and we are embedding both informal and formal statements given by their bag of word representations.

Also in the initial stages, we could rely on small corpora of known translations: for example the book on the proof of Kepler conjecture contains matching labeled statements for a few hundred of the formalized statements. This is too small of a corpus to train a reasonable translation mechanism, but it could be used as a test set or alternatively, a subset of it could enrich the above very rough alignment between the two corpora with additional training data.

Another way to collect more aligned data would be utilize a small amount of automatically generated "informalization" of formal statements: one could create some somewhat human readable representation of the formal statements programmatically which could serve as extra training data for the backward translation of those statements from their informal images to formal embeddings.

A more sophisticated variant of the last idea is to exploit cycle-consistency [99] that was successfully used for unsupervised translation between natural languages in [54]. The idea is to train two translation models $T : L \longrightarrow L'$ and $T' : L' \longrightarrow L$ between informal natural language L and formal mathematical statements L' or the embeddings spaces thereof, while enforcing $T' \circ T \approx \mathrm{id}_L$ and $T \circ T' \approx \mathrm{id}_{L'}$. with suitable losses. This could be further anchored by enforcing that translated expressions should have a somewhat similar bag-of-word statistics when one uses a rough initial translation between the concept libraries of the formal and informal corpora.

In the second phase one could use more sophisticated embedding methods: those that were trained in an un/self-supervised manner: pretrained models on the informal side could utilize the ideas of Subsection 7.2, while the embedding model mapping formal statements to vectors would be trained by the methods from Subsection 7.3. The initial match between a subset of formal and informal

statements could be used to train an alignment model between the two embedding spaces by fitting a multi-layer perceptron on those statements for which we found or generated an alignment. This alignment stage could be semi-supervised by utilizing limited human supervision via active learning.

As one gains more confidence that the overall alignment can be improved by matching newly found true statements to the closest matching informal statements, one could start fine-tuning the translation model that translates images of informal statements directly to the embedding vectors of formal statements.

At this point, it is highly uncertain what is the critical mass and quality of the initial alignments that is required to bootstrap a self-perpetuating feedback loop to get to high quality autoformalization system in the end.

7.6 Model Architectures

Computer vision models have improved significantly in the past few years[77, 34, 76, 12, 91], also lately due to the increased use of automated neural architecture search [102, 55, 65], however it is not very likely that our systems needs a more sophisticated based network architecture than one that is good enough for practical OCR, which is a more or less solved problem, especially that most of our input would come from computer generated images rather than scans.

On the vision side, some gains might be expected from utilizing attention mechanism in the context of recurrent evaluation: creating the output in an incremental manner by attending to different parts of the input image. However this level of engineering may not be necessary for evaluating the feasibility of an initial prototype.

More gains might be expected on the formalization side by looking up context: having a database of precomputed formal-to-informal alignment and allowing the system to retrieve from this growing knowledge-base. This might be useful for the system to allow for looking up the definitions of ambiguous notions or just bringing in more knowledge and context to distinguish between alternative interpretations. However, engineering such a system would require significant effort.

On the formal side, it appears that both the network architecture and the translation to input has a significant effect on the performance of the reasoning system. Generally, it seems like deep graph embedding networks [86, 62] with lot of node-sharing do best, but this is a relatively recent finding and it can be expected that the incorporation of attention mechanisms and utilizing automated neural architecture search in this particular context could bring significant further improvements.

7.7 Integrating Reverse Proof Search

Currently, tactics based higher order proof search works in reverse order: instead of trying to get to the theorem from the premises, one starts with the theorem to be proved and by applying so called tactics, one produces new goals to be proved. Once one closes all necessary goals, the forward proof can be constructed by tracing this process in the opposite direction. This is the methodology used in almost all of the higher order order theorem provers [6, 38, 92].

The reason for using backward reasoning is that it is typically simpler to work one's way backwards and find applicable premises than combining exisiting knowledge in a way that results in the expected result perfectly.

Although the plan for formalization in Section 6 does not include backwards reasoning is an essential constituent, it would be hard to imagine a practical system that can do efficient reasoning without backward proof search.

One way to incorporate backward reasoning would be for testing for trivial equivalence of statements. If one finds a new statement but it can be very easily proven that it is equivalent to one previously known statement, then the new statement is unlikely to be interesting, even if it was found as the result of a lengthy exploration process. This could be used for filtering translation candidates: statements whose embedding vector is close to an target embedding, but is still trivially shown to be equivalent to some existing statement in the theorem library.

Another use for backwards reasoning is to verify conjectures that were generated by some model directly, or were conjectured based on examples or generalized from special cases by dropping hypotheses or simple induction. In these cases, the statement is not generated by forward reasoning steps that necessarily resulted in true statements, but is just conjectured by the system heuristically and needs to be verified by proper reasoning.

Finally, reverse reasoning could be used for verifying statements that were given in a formal description by the user.

7.8 Filtering Translation Candidates

As described in the last subsection and was emphasized in Section 6, we need to filter out translation candidates that are incorrect, trivial or uninteresting.

The first criterion is clear: we do not expect wrong statements to be correct formalization candidates. Although it might be that our informal corpus contains incorrect statements, they are hopefully in minority. Incorrect statements do not give strong enough signal for autoformalization, so the best is just not

consider them for translation results.

It is harder to discard candidate translations that are trivially true (e.g. due to too general assumptions or other translation errors). In this case, we could be able to detect these candidates by trying to prove them without utilizing the premises that the translations would suggest. If they are trivially provable without relying on those parts of the mathematics that the informal paper points out as helpful, then it is highly likely that the statement is not the correct formalization of the provided text.

Also, if a statement is overly long, or has a lot of redundant subtrees, then it is highly unlikely that it comes from formalizing human contents.

A machine learning based idea would be to also try to learn a model that translates formal statements into natural language ones and verify that the back-translated results are in the proximity of the original embedding vector of natural language input.

These are just a few heuristic indications on deciding whether some statement with matching embedding vectors are likely to match certain parts of the natural language corpus. However filtering the formalization candidates is one of the most critical and weakest point of our proposal. It will need a lot

7.9 Curriculum Learning

Curriculum learning is a promising way of learning how to find longer proofs. A remarkable result demonstrating the power of strong curriculum is [101] in which they can train a reinforcement learning system to find proofs consisting of several thousand proof elementary proof-steps without any search by just by being predicted by the machine learning system.

Creating good curriculums might be crucial for training strong autoformalization systems. Learning to generate a curriculum based on experience or utilizing the value network of a theorem proving system seems like a very promising direction towards that end.

7.10 Synthesizing New Tactics

Tactics in proof assistants are subroutines that can perform complicated algorithms in order to produce long chains of arguments about the correctness of certain formulas. Some of the tactics might be heuristic: they may just fail when timed out or not finding enough evidence, they are also allowed to produce some new goal statement that implies the current statement, but the produced goal

statement does not need to be true at all.

Examples of existing tactics include:

1. Find and rewrite some part of a statement by a given set of identities.
2. Applying a SAT-solver to prove a statement purely by propositional logic reasoning.
3. Applying a first order logic prover to prove an approximate translation of the statement by first order logic reasoning.
4. Use basic arithmetic reasoning to bring some part of a formula to a normal form.
5. Use Groebner bases [10] to reason about polynomial identities.
6. Use mixed integer-linear program solver to argue about a set of (Diophantine) inequalities or to solve combinatorial optimization problems.

This is just a small selection of tactics that are applied in proof assistant, but it should be visible from this list that the complexity and sophistication of tactics might be very diverse, also tactics might utilize highly complex systems under the hood.

This means that the difficulty of creation of new tactics is open-ended as it may range from from the sequencing of a small number of existing tactics to utilizing complex systems with millions of lines of source code.

Albeit, state of the art software synthesis approaches might scale to the generation of tactics by combining a modest number of existing tactics, it is unlikely that one would be able to synthesize a general purpose computer algebra system from scratch as an initial step.

However, the vast majority of sophisticated human mathematics was discovered without the aid of computer programs, so we can hope the a performance approaching or matching that of human mathematicians can be achieved without the synthesis of new complicated tactics and can be based mostly on explicit rewrites and simple propositional logic automation alone.

7.11 Semantic Exploration, Experimentation

However higher order logic theorem provers tend to work on the fully syntactic level and statements are not necessarily connect to objects constructed in the memory.

Especially for refutation and counterexample generation, it might be important to find substitutions into statements that provide a refutation of that statement. For simpler types like truth values, natural numbers, one could try random substitutions or SAT-solving, but machine learning could be helpful for trying to generate terms, the substitution of which yields a refutation of universally quantified statements.

However more complicated tasks, that rely on objects that are harder to create, for example polynomial, rings or topological spaces with more complex properties would profit from more sophisticated databases of mathematical objects. Again these objects could be described constructively in the same language as the statement is given, but it would be beneficial to engineer special purpose example repositories that can be used for refuting or verifying non-trivial statements.

Also there has been work in finding finite models for statements (especially in the context of refutations) via SAT solvers [87, 9] and also using a quickcheck style random counterexample generator [11]. It seems promising that similarly to deep learning based proof search guidance, one could also train machine learning systems for guiding counterexample generators as well.

In general it is a promising research direction to use deep learning based models to embed not just the syntactic form of formulas, but also some experience stream that is associated with experimentation with the statements, for the result of random or guided substitutions and rewrites.

7.12 Multi-Agent Specialization

One difference between theorem proving and learning game playing engines for two-player games like go, is the breadth of mathematics. The reason for this is the fact that in games, each player should be able to answer moves by any other player, which keeps the skills required for strong game-play concentrated in a relatively narrow domain. However, in mathematics, there are a lot of directions to be explored and each of those domains require different styles of reasoning and intuition. Most human expert mathematicians are skilled in only a handful of mathematical areas thoroughly.

For neural network based systems, it would mean that it might require very large neural networks to distill all the skills required to cope with any areas of mathematics. One way to cope with that issue would be utilizing mixture of expert models [96]. However, this requires a fixed gating mechanism and therefore would be relatively hard to extend.

A more flexible multi-agent architecture would be based on artificial market mechanisms that allows arbitrary agents to bet on the status of mathematical

conjectures and would be rewarded for correct predictions, proving theorems formally and for introducing interesting new conjectures.

Although the above direction opens a large box of interesting mechanism design [61] issues, but besides a straightforward way of scaling up the system flexibly, it could present a great way to decouple reward function design from the implementation of agents. Also it would allow for easier benchmarking of competing approaches in a realistic setting – measuring the competitiveness of agents with respect to the whole system, instead of comparing with a small selection of cherry picked agents. An interesting theoretical work on Logical Induction [21] proposes that a betting market based multi-agent system under resource constraints is useful for assigning consistent probability values to mathematical statements. This could give some theoretical backing and inspiration to this solution.

8 Short History of Autoformalization

The idea of autoformalization was first presented in 1961 by John McCarthy [58], suggesting that: "checking mathematical proofs is potentially one of the most interesting and useful applications of automatic computers". An early attempt at autoformalization dates back to the 1990 doctoral thesis of Donald Simons [71] in the proof checking context. A first thorough study on the topic was performed in the doctoral thesis of Claus Zinn [100] in 2004. However none of those works resulted in even partially useful solutions.

Realizing both the importance and challenges of autoformalization, Josef Urban started to work on the topic in the early 2000s. He devised a first large scale benchmark for reasoning in large theories [80], especially motivated by the insight that reasoning in the presence of a large knowledge base of mathematical facts must be a critical component in any autoformalization system.

In 2007, Josef Urban published his pioneering MaLARea [82] system for reasoning in large theories. A crucial aspect of it is a machine learning based solution to the premise selection problem in large knowledge bases. This was made possible by his first order logic translation [81] of the Mizar corpus by enabling the benchmarking of various premise selection methods by leveraging state of the art automated first order theorem provers, especially theorem prover E [68]. Ever since then, he with Cezary Kaliszyk have been spearheading the research of automated reasoning in large theories [36, 51, 1, 41, 44, 46, 43, 44, 8]. Also their research includes most of the pioneering attempts at various aspects of autoformalization as well: [47, 48].

The domain of autoformalization and reasoning in large theories is deeply connected to that of automated theorem proving which is a large domain with lot

of history. However, given the extensive literature on that topic, here I just sketch the high level picture on how it relates to the topic and high level vision of this paper.

Most of traditional proof automation was done for propositional logic (mostly via SAT-solvers) and for first order logic, via automated theorem provers (ATP) and SMT-solvers. However all of that automation is done via combinatorial search methods that work with intermediate formulas that can grow very big and hard to interpret and process for humans and neural networks.

Most human formalization effort was done by interactive proof assistants (ITPs). Although proof automation by hammers for ITPs has gained some traction, most complex formalization work was done in ITPs with very simple automation, using relatively weak first order logic solvers (like the `auto` tactic in Coq or `MESON` in HOL light). However, despite this relatively low level of automation, proof assistants have been used for formalizing the proof of extremely complex theorems, like that of the Feit-Thomposon theorem and the Kepler Conjecture (although the proof the Kepler conjecture made extensive use of linear program solvers as well). This proposal aims at automatizing the human process by augmenting ITPs with deep neural network based reasoners which looks like the most promising path toward open-ended improvements. Although integration of ITPs with ATPs is possible, it is relatively complex and does not seem to be necessary in the presence of really strong AI systems in charge of controlling the ITP system. Therefore, we consider most previous work on first order automated theorem proving relatively irrelevant in the context of this proposal.

9 Indications of Feasibility

Given the great complexity and breadth of the problem, it is justified to ask why is autoformalization even considered as a realistic goal in the short term – that is, within years. This section tries to give a heuristic arguments for the feasibility of this task by methods that are either known or are on a strong improvment trajectory.

The success of autoformalization hinges on solving two difficult looking tasks:

1. General purpose symbolic reasoning
2. Strong natural language understanding

I argue that deep learning will enable the advancement of both of those areas to the extent that is necessary for human level formalization and reasoning performance in the coming years. Although the two domains are getting increasingly connected, let us try to revise their recent progress in separation with the focus

on the question on whether they are on a path that makes strong autoformalization realistic in the short term.

9.1 Search and Reasoning

Recently, it has been demonstrated by AlphaZero [69] that the same relatively simple algorithm based on Monte Carlo tree search (MCTS) [50] and residual convolutional networks [34] could achieve higher than human performance in several two-person games: go, chess and shogi, by self-play alone, utilizing the same algorithm for each of those games, **without** learning on any human expert games at all. Effectively, AlphaZero was able to rediscover all of the important chess, go and shogi knowledge in few days that took human players centuries of work to discover.

Out of those games, go is the most impressive feat as it required skills that were considered to be human specific: strong intuition and very deep reasoning. Often, making the correct move in go requires foresight to dozens of moves ahead, anticipating all possible counter-moves by the opponent. Acquiring professional level skills takes a decade long studying for the world's best players. This requires analyzing thousands of games and learning a lot of special tricks which often require very deep understanding on how to counter hard-to-spot traps that could be set by another skilled opponent.

AlphaZero managed to acquire such skills just by utilizing the pattern matching capabilities of deep convolutional networks and bootstrapping itself using the incremental information propagation of MCTS, gradually deepening its foresight via self-play. One can say that the less than 50 million floating point parameters of the policy network encodes all the accumulated knowledge that it would take exceptionally gifted human players to learn in a life-time and which surpasses the knowledge that took dozens of generations of go players to collect.

This suggests that MCTS could serve as the method of choice for automated reasoning as well, just as was suggested and demonstrated in [19, 45]. While these works are based on relatively simple prover, logic and datasets and do not address premise selection, they still give a strong indication of the potential of MCTS in the more general reasoning context.

However, mathematical reasoning differs from game playing in many respects:

1. The impossibility of self play: for example if open ended exploration is considered as an alternative, it would be hard to decide what to explore. Also self-play provides a nice, well behaved, adaptive curriculum for training. The lack of self play makes automated curriculum learning much harder for theorem proving.

2. Large, indefinitely growing knowledge base, resulting in a virtually infinite action space.

3. Very sparse reward: in two player games either player wins in most cases, so it is relatively easy to assign reward to each situation to train on. In the case of theorem proving, it is very hard to assign reward to failed proof attempts.

4. The diversity of mathematical knowledge: by nature, two-player games are very coherent, since each player has to be able to answer any move by any other player. Basically, each player has to accumulate the same body of knowledge in order to compete with all other possible players. This means that the accumulated knowledge concentrates on a much narrower area, also unsupervised exploration can work due to this high coherence. On the other hand, mathematics consists of wide range of loosely connected disciplines and it takes a lot of human experts to cover each of them. This suggests that both discovering and compressing the reasoning skills for all of interesting mathematics might be much harder, if not impossible.

In 2016, as evidence started to mount for the scalability of deep learning, especially for its potential for open ended improvement when provided with enough data and given enough depth and parameters, DeepMath was the first attempt for its application at premise selection for the Mizar corpus [81] via convolutional networks yielding some initial improvements for this task. Also theorem prover E was improved by integrating neural network guidance [57], but those works were relatively inefficient due to requiring many neural network evaluations even for relatively simple proofs.

In 2017, TacticToe [23], has demonstrated convincingly that tactic based higher order theorem proving via machine learning (even without the use of deep learning) is not just realistic, but can outperform previous ATP based baselines by a large margin.

More recently the DeepHOL system [6] gave further demonstration for the power of deep learning in the more general case: for higher order logic and in the presence of a large knowledge base of premises to be used. In fact, it turns out that a relatively simple Dagger [67] style training-set accumulation in connection with WaveNet [83] convolutional networks and simple depth-first search, can outperfom relatively complex solutions. This was demonstrated on a practical large scale benchmark proposed in the same paper.

However, a sequence of symbols is not the most natural representation for mathematical formulas. Formulas can be best described as graphs, suggesting the use of graph neural networks, which was suggested first in [86] and was then verified to yield significant gains (40% relative increase in success rate) on the HOList benchmark in the end-to-end proving scenario [62] as well.

While all previous machine learning based solutions relied on training human proofs, DeepHOL-Zero [5] has demonstrated that relatively simple exploration heuristics allow to bootstrap systems that can reason as well without training on existing human proof logs. Although this approach still needs a repository of statements, a large fraction of which with short proofs, it is still remarkable, since proof search can pick premises from a large repository at every step.

Still most of the proofs found automatically are relatively short. This gives ground for skepticism: can these methods scale to proofs that are really long, taking 100s or even 1000s steps? In fact there is promising work in that direction as well: [101] demonstrates successfully, that with the right curriculum, at least in a limited setting, it is possible to train models that can create proofs of several thousand steps without error. However, it requires a very careful, synthetic training schedule with a lot of automatically generated tasks of increasing complexity.

One can argue by the analogy of AlphaZero and based on the fact that network architecture changes have a huge effect on prover performance that open-ended improvement of proof-search should be possible by using the a decent network architecture. Graph neural networks and transformer networks are the most promising candidates today, but given the pace of progress in neural network architecture design, this might change quickly. Leveraging automated network architecture search is another promising option to leverage.

Still this leaves open the question of obtaining training data and devising the right curriculum. As Section 6 suggested, training data could be collected by natural language guided exploration, while Section 7.9 suggests some initial ideas for bootstrapping a good curriculum.

9.2 Natural Language Processing and Understanding

Since 2017, natural language processing went through a revolution similar to that of computer vision, due to new neural model architectures, especially transformer networks [84] and large scale self-supervised training on vast corpora [64, 17, 93]. This has spurred fast advances in machine translation and language understanding. On some of the benchmark, this has resulted in human or close to human performance, for example on SQuAD 1.0 [95]. However this has lead to development of improved benchmarks to target the common weak points of those algorithms. Still, progress is still strong in this domain and improved model architectures and training on better tasks and larger corpora have yielded significant gains at a steady pace. Based on the analogy with computer vision, one can also foresee that natural architecture search could give rise to further advances in this field as well.

An autoformalization system can leverage those advances for stronger transla-

tion models from natural language to the embedding space of formal statements.

9.3 Generative Modelling

Another important task in mathematical reasoning is to come up with new terms to be substituted. This can be used for creating counterexamples and also in creative proofs, for example for coming up with useful induction hypothesis. Finally, conjecturing might require the generation of new statements based on analogies and previous experiences.

However, neural networks based generative models have demonstrated increasingly higher quality performance on a wide range of generative tasks: image generation, music generation, text generation and the generation of chemical compounds. This models have demonstrated impressive improvements in the quality of generated artifacts. For example computer generated faces start to become increasingly indistinguishable from human ones and neural networks could generate novel compounds for drug discovery as well.

The fast pace of improvements suggests that generating true or useful mathematical formulas should be feasible and efficient as well.

9.4 Overview

Here is a short overview of the factors that support the potential success of autoformalization in the coming years:

1. The success of deep learning infused search in two person games, especially AlphaZero [69] style Monte Carlo tree search [50].
2. The demonstrations of the usefulness of deep learning in automated reasoning: premise selection [2] and proof guidance [57, 6, 62]
3. The demonstration that automated proof search can be learned without imitation [5].
4. The recent success of graph neural networks in various domains [90], including theorem proving [86, 62] and the fast progress in this domain.
5. The success of imposing cyclic translation consistency [99] in image generation and unsupervised translation [54] give strong indications that autoformalization could be bootstrapped using very limited set of labeled pairs of formalized theorems.
6. The success of hindsight experience replay [3] to address the sparse reward problem for robotics applications.

7. The quick pace of progress in natural language processing via large, deep network models, and large scale self-supervised pretraining. Impressive results in several translation and natural language understanding benchmarks [56].

8. Generative neural models improve at a fast pace and yield impressive result in a wide range of domains from image generation to drug discovery.

9. Multi-agent system with agents specialized in different domains of mathematics with a common, growing repository of statements could provide means both for scaling up the performance without individual neural networks growing too big, also they would help assessing the plausibility of mathematical statements [21] and could give a rise to open-ended self-improvement.

10. Automated optimization of neural architectures via neural architecture search [102, 78] and other automated methods [28] help scaling up the performance of deep networks.

11. Although the steady pace of miniaturization of integrated chips is approaching a certain end [39], the computation resources available for deep learning purposes are still expanding quickly and are getting cheaper. For example, as of July 2019, Google' TPUv3 based pods can deliver over 100 petaFLOPS performance for deep learning purposes [27]. However hardware is just one side of the story: software advances in deep learning framework have resulted in huge improvements for machine learning workloads on the same hardware.

10 General Summary and Conclusions

We have argued in this paper, that:

1. Autoformalization enables the development of a human level mathematical reasoning engine in the next decade. This could lead to the development of a human or super-human artificial mathematicians and programmers.

2. The implementation of autoformalization presents significant technical and engineering challenges.

3. Successful implementation of mathematical reasoning (theorem proving) and autoformalization has many implications that go far beyond just transforming mathematics itself: It could usher a new era of computer human interaction and result in the creation of a general purpose reasoning module to be used in other AI systems.

4. A reasoning system based purely on self-driven exploration for reasoning without informal communication capabilities would be hard to evaluate and use.

5. It is easier to engineer and bootstrap a system that learns to perform both formalization and reasoning than either task in separation.

6. It is easier to create a formalization system from image than text data.

7. A naïve, direct translation approach for autoformalization would be brittle, hard to engineer and unlikely to work.

8. Combining approximate formalization (predicting embedding vectors instead of formulas) and guided exploration is a relatively robust and promising direction to autoformalization than direct translation.

9. As long as the foundational system is strong enough for formalizing the set theory and demonstrated its feasibility on some large scale human formalization effort, its choice should have negligible effect. However classical higher order logic might be the closest match for informally given mathematics and the easiest to bootstrap, initially.

10. Deep Learning should be crucial for open ended improvement and reaching human level reasoning and formalization performance.

11. Recent progress in neural architectures, language modelling, self- and semi-supervised training, reinforcement learning, automated neural architecture search, and AI driven theorem proving paves the way for strong automated reasoning and formalization systems.

11 Acknowledgements

My warmest thanks go to my close collaborators and colleagues Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, Francois Chollet, Alex Alemi, Stewart Wilcox, Niklas Een, Geoffrey Irving, Victor Toman and Aditya Paliwal for their contributions towards the goals sketched here. I am also indebted to Josef Urban and Cezary Kaliszyk for their pioneering work and selflessly sharing their vision and expertise and also for their collaboration on this area. I am also thankful to Ilya Sutskever, Henryk Michalewski, Daniel Huang, Quoc Le, Dániel Varga, Zsolt Zombori, Adrián Csiszárik for their feedback and valuable discussions on this topic. I would like to thank to Jay Yagnik, Rahul Sukthankar, Ashok Popat, Rif Saurous, Jeff Dean and Geoffrey Hinton for their support of deep learning based reasoning work at Google.

References

[1] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning*, 52(2):191–213, 2014.

[2] Alexander A Alemi, François Chollet, Geoffrey Irving, Niklas Eén, Christian Szegedy, and Josef Urban. Deepmath-deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, pages 2235–2243, 2016.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[4] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. *arXiv preprint arXiv:1710.11041*, 2017.

[5] Kshitij Bansal, Sarah M Loos, Markus N Rabe, and Christian Szegedy. Learning to reason in large theories without imitation. *arXiv preprint arXiv:1905.10501*, 2019.

[6] Kshitij Bansal, Sarah M Loos, Markus N Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher-order theorem proving. *ICML 2019. International Conference on Machine Learning*, 2019.

[7] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer, 1999.

[8] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C Paulson, and Josef Urban. Hammering towards qed. *Journal of Formalized Reasoning*, 9(1):101–148, 2016.

[9] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *International conference on interactive theorem proving*, pages 131–146. Springer, 2010.

[10] Bruno Buchberger. Introduction to groebner bases. In *Logic of Computation*, pages 35–66. Springer, 1997.

[11] Lukas Bulwahn. The new quickcheck for isabelle. In *International Conference on Certified Programs and Proofs*, pages 92–108. Springer, 2012.

[12] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[13] The Coq Proof Assistant. `http://coq.inria.fr`.

[14] Thierry Coquand and Gérard Huet. *The calculus of constructions*. PhD thesis, INRIA, 1986.

[15] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 933–941. JMLR. org, 2017.

[16] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[19] Michael Färber, Cezary Kaliszyk, and Josef Urban. Monte carlo tableau proof search. In *International Conference on Automated Deduction*, pages 563–579. Springer, 2017.

[20] Melvin Fitting. *First-order logic and automated theorem proving*. Springer Science & Business Media, 2012.

[21] Scott Garrabrant, Tsvi Benson-Tilsen, Andrew Critch, Nate Soares, and Jessica Taylor. Logical induction. *arXiv preprint arXiv:1609.03543*, 2016.

[22] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[23] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with hol4 tactics. In *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46, pages 125–143, 2017.

[24] Erik Gawehn, Jan A Hiss, and Gisbert Schneider. Deep learning in drug discovery. *Molecular informatics*, 35(1):3–14, 2016.

[25] Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[26] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the Odd Order Theorem. In *ITP*, pages 163–179, 2013.

[27] Google's scalable supercomputers for machine learning, cloud tpu pods, are now publicly available in beta. `https://bit.ly/2YkZh3i`.

[28] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

[29] Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. When will ai exceed human performance? evidence from ai experts. *Journal of Artificial Intelligence Research*, 62:729–754, 2018.

[30] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1362–1371. JMLR. org, 2017.

[31] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the kepler conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

[32] John Harrison. HOL Light: A tutorial introduction. In *FMCAD*, pages 265–269, 1996.

[33] John Harrison. Floating-point verification using theorem proving. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 211–242. Springer, 2006.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[35] Marijn JH Heule, Oliver Kullmann, and Victor W Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 228–245. Springer, 2016.

[36] Kryštof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In *International Conference on Automated Deduction*, pages 299–314. Springer, 2011.

[37] William A Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.

[38] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *ICLR 2018. International Conference on Learning Representations*, 2018.

[39] Hiroshi Iwai. End of the scaling theory and moore's law. In *2016 16th International Workshop on Junction Technology (IWJT)*, pages 1–4. IEEE, 2016.

[40] Cezary Kaliszyk, François Chollet, and Christian Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. *arXiv preprint arXiv:1703.00426*, 2017.

[41] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with flyspeck. *Journal of Automated Reasoning*, 53(2):173–213, 2014.

[42] Cezary Kaliszyk and Josef Urban. Hol (y) hammer: Online atp service for hol light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

[43] Cezary Kaliszyk and Josef Urban. Learning-assisted theorem proving with millions of lemmas. *Journal of symbolic computation*, 69:109–128, 2015.

[44] Cezary Kaliszyk and Josef Urban. Mizar 40 for mizar 40. *Journal of Automated Reasoning*, 55(3):245–256, 2015.

[45] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement learning of theorem proving. *arXiv preprint arXiv:1805.07563*, 2018.

[46] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Efficient semantic features for automated reasoning over large theories. In *IJCAI*, 2015.

[47] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning to parse on aligned corpora (rough diamond). In *International Conference on Interactive Theorem Proving*, pages 227–233. Springer, 2015.

[48] Cezary Kaliszyk, Josef Urban, and Jiri Vyskocil. System description: statistical parsing of informalized mizar formulas. In *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 169–172. IEEE, 2017.

[49] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[50] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[51] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: machine learning for sledgehammer. In *International Conference on Interactive Theorem Proving*, pages 35–50. Springer, 2013.

[52] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1945–1954. JMLR. org, 2017.

[53] Jeffrey C Lagarias. *The ultimate challenge: The 3x+ 1 problem*. American Mathematical Soc., 2010.

[54] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017.

[55] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[56] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[57] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, 2017.

[58] John McCarthy. Computer programs for checking mathematical proofs, a paper presented at the symposium on recursive function theory. *New York, April*, 1961.

[59] Norman Megill. Metamath. In *The Seventeen Provers of the World*, pages 88–95. Springer, 2006.

[60] The Mizar Mathematical Library. `http://mizar.org`.

[61] Noam Nisan et al. Introduction to mechanism design (for computer scientists). *Algorithmic game theory*, 9:209–242, 2007.

[62] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *arXiv preprint arXiv:1905.10006*, 2019.

[63] Álvaro Pelayo and Michael Warren. Homotopy type theory and voevodsky's univalent foundations. *Bulletin of the American Mathematical Society*, 51(4):597–648, 2014.

[64] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[65] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[66] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[67] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[68] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.

[69] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[70] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[71] Donald Lee Simon. Checking number theory proofs in natural language. *PhD thesis*, 1990.

[72] Konrad Slind and Michael Norrish. A brief overview of hol4. In *International Conference on Theorem Proving in Higher Order Logics*, pages 28–32. Springer, 2008.

[73] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[74] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[75] Patrick Suppes. *Axiomatic set theory*. Courier Corporation, 1972.

[76] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[77] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[78] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[79] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[80] Josef Urban. Translating mizar for first order theorem provers. In *International Conference on Mathematical Knowledge Management*, pages 203–215. Springer, 2003.

[81] Josef Urban. Mptp 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37(1-2):21–43, 2006.

[82] Josef Urban. Malarea: a metasystem for automated reasoning in large theories. *ESARLT*, 257, 2007.

[83] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[85] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint 1804.07461, 2018.

[86] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Advances in Neural Information Processing Systems*, pages 2786–2796, 2017.

[87] Tjark Weber. *Sat-based finite model generation for higher-order logic*. PhD thesis, Technische Universität München, 2008.

[88] Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The isabelle framework. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2008.

[89] Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic. *arXiv preprint arXiv:1608.02644*, 2016.

[90] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[91] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[92] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. *arXiv preprint arXiv:1905.09381*, 2019.

[93] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[94] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[95] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[96] Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193, 2012.

[97] Oscar Zariski. A new proof of hilbert's nullstellensatz. *Bulletin of the American Mathematical Society*, 53(4):362–368, 1947.

[98] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.

[99] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[100] Claus Zinn. Understanding informal mathematical discourse. *PhD thesis, Institut fur Informatik, Universitat Erlangen-Nurnberg*, 2004.

[101] Zsolt Zombori, Adrián Csiszárik, Henryk Michalewski, Cezary Kaliszyk, and Josef Urban. Towards finding longer proofs. *arXiv preprint arXiv:1905.13100*, 2019.

[102] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.