Automating Formalization by Statistical and Semantic Parsing of Mathematics

Cezary Kaliszyk[1]*, Josef Urban[2]**, and Jiří Vyskočil[2]**

[1] University of Innsbruck, Austria
[2] Czech Technical University

Abstract. We discuss the progress in our project which aims to automate formalization by combining natural language processing with deep semantic understanding of mathematical expressions. We introduce the overall motivation and ideas behind this project, and then propose a context-based parsing approach that combines efficient statistical learning of deep parse trees with their semantic pruning by type checking and large-theory automated theorem proving. We show that our learning method allows efficient use of large amount of contextual information, which in turn significantly boosts the precision of the statistical parsing and also makes it more efficient. This leads to a large improvement of our first results in parsing theorems from the Flyspeck corpus.

1 Introduction: Learning Formal Understanding

Computer-understandable (formal) mathematics [17] is still far from taking over the mathematical mainstream. Despite recent impressive formalizations such as the Formal Proof of the Kepler conjecture (Flyspeck) [15], Feit-Thompson [9], seL4 [23], CompCert [26], and CCL [1], formalizing proofs is still largely unappealing to mathematicians. While research on AI and strong automation over large theories has taken off in the last decade [2], so far there has been little progress in automating the understanding of informal LaTeX-written and ambiguous mathematical writings.

Automatic parsing of informal mathematical texts into formal ones has been for long time considered a hard or impossible task. Among the state-of-the-art Interactive Theorem Proving (ITP) systems such as HOL (Light) [16], Isabelle [31], Mizar [11] and Coq [4], none includes automated parsing, instead relying on sophisticated formal languages and mechanisms [7, 10, 13, 28]. The past work in this direction – most notably by Zinn [33] – has often been cited as discouraging from such efforts.

We have recently initiated [22, 21] a *project to automatically learn formal understanding* of mathematics and exact sciences using a large corpus of alignments [8] between informal and formal statements. Such learning can additionally integrate strong semantic filtering methods such as typechecking combined

*Supported by the ERC Starting grant no. 714034 *SMART*.
**Supported by the ERC Consolidator grant no. 649043 *AI4REASON*.

with large-theory Automated Theorem Proving (ATP). In more detail, we believe that the current state of human-based formalization can be significantly helped by automatically learning how to formalize ("semanticize") informal texts, based on the knowledge available in existing large formal corpora. There are several justifications for this belief:

1. Statistical machine learning (data-driven algorithm design) has been responsible for a number of recent AI breakthroughs, such as web search, query answering (IBM Watson), machine translation (Google Translate), image recognition, autonomous car driving, etc. Given enough data to train on, data-driven algorithms can automatically learn complicated sets of rules that would be often hard to program and maintain manually.
2. The recent progress of formalization, provides reasonably large corpora such as the Flyspeck project [15]. These, together with additional annotation [14], can be used for experiments with machine learning of formalization. The growth of such corpora is only a matter of time, and automated formalization might gradually "bootstrap" this process, making it faster and faster.
3. Statistical machine learning methods have already turned out to be very useful in proof assistant automation in large theories [2], showing that data-driven techniques do apply also to mathematics.
4. Analogously, strong semantic *automated reasoning in large theories* [30] (ARLT) methods are likely to be useful in the formalization field also for complementing the statistical methods that learn formalization. This could lead to hybrid understanding/thinking AI methods that self-improve on large annotated corpora by cycling between (i) statistical prediction of the text disambiguation based on learning from existing annotations and knowledge, and (ii) improving such knowledge by confirming or rejecting the predictions by the semantic ARLT methods.

The last point (4) is quite unique to the domain of (informal/formal) mathematics, and a good independent reason to work on this AI research. There is hardly any other domain where natural language processing (NLP) could be related to such a firm and expressive semantics as mathematics has, which is additionally to a reasonable degree already checkable with existing ITP and ARLT systems. Gradually improving the computer understanding of how mathematicians (ab)use the normal imprecise vocabulary to convey ideas in the semantically well-grounded mathematical world, may even improve the semantic treatment of arbitrary natural language texts.

1.1 Contributions

This paper extends our previous short papers [22, 21] on the informal-to-formal translation. We first introduce the informal-to-formal setting (Sec. 2), summarize our initial probabilistic context-free grammar (PCFG) approach of [21] (Sec. 3), and extend this approach by fast context-aware parsing mechanisms that very significantly improve the performance.

- **Limits of the context-free approach.** We demonstrate on a minimal example, that the context-free setting is not strong enough to eventually learn correct parsing (Sec. 4) of relatively simple informal mathematical formulas.
- **Efficient context inclusion via discrimination trees.** We propose and efficiently implement modifications of the CYK algorithm that take into account larger parsing subtrees (context) and their probabilities (Sec. 5). This modification is motivated by an analogy with large-theory reasoning systems and its efficient implementation is based on a novel use of fast theorem-proving data structures that extend the probabilistic parser.
- **Significant improvement of the informal-to-formal translation performance**. The methods are evaluated, both by standard (non-semantic) machine-learning cross-validation, and by strong semantic methods available in formal mathematics such as typechecking combined with large-theory automated reasoning (Sec. 6).

2 Informalized Flyspeck and PCFG

The ultimate goal of the informal-to-formal traslation is to automatically learn parsing on informal LaTeX formulas that have been aligned with their formal counterparts, as for example done by Hales for his informal and formal Flyspeck texts [14, 29]. Instead of starting with LaTeX, where only hundreds of aligned examples are so far available for Flyspeck, we reuse the first large informal/formal corpus introduced previously in [21], based on *informalized* (or *ambiguated*) formal statements created from the HOL Light theorems in Flyspeck. This provides about 22000 informal/formal pairs of Flyspeck theorems.

2.1 Informalized Flyspeck

We apply the following ambiguating transformations [21] to the HOL parse trees to obtain the aligned corpus:

- Merge the 72 overloaded instances defined in HOL Light/Flyspeck, such as `("+", "vector_add")`. The constant `vector_add` is replaced by `+` in the resulting sentence.
- Use the HOL Light infix operators to print them as infix in the informalized sentences. Since `+` is declared as infix, `vector_add u v`, would thus result in `u + v`.
- Obtain the "prefixed" symbols from the list of 1000 most frequent symbols by searching for: `real_, int_, vector_, nadd_, treal_, hreal_, matrix_`, `complex_` and make them ambiguous by forgetting the prefix.
- Overload various other symbols used to disambiguate expressions, for example the "c"-versions of functions such as `ccos cexp clog csin`, similarly for `vsum, rpow, nsum, list_sum`, etc.
- Remove parentheses, type annotations, and the 10 most frequent casting functors such as `Cx` and `real_of_num`.

2.2 The Informal-To-Formal Translation Task

The *informal-to-formal translation task* is to construct an AI system that will automatically produce the most probable formal (in this case HOL) parse trees for previously unseen informal sentences. For example, the informalized statement of the HOL theorem `REAL_NEGNEG`:

```
! A0 -- -- A0 = A0
```

has the formal HOL Light representation shown as a tree in Fig. 1.

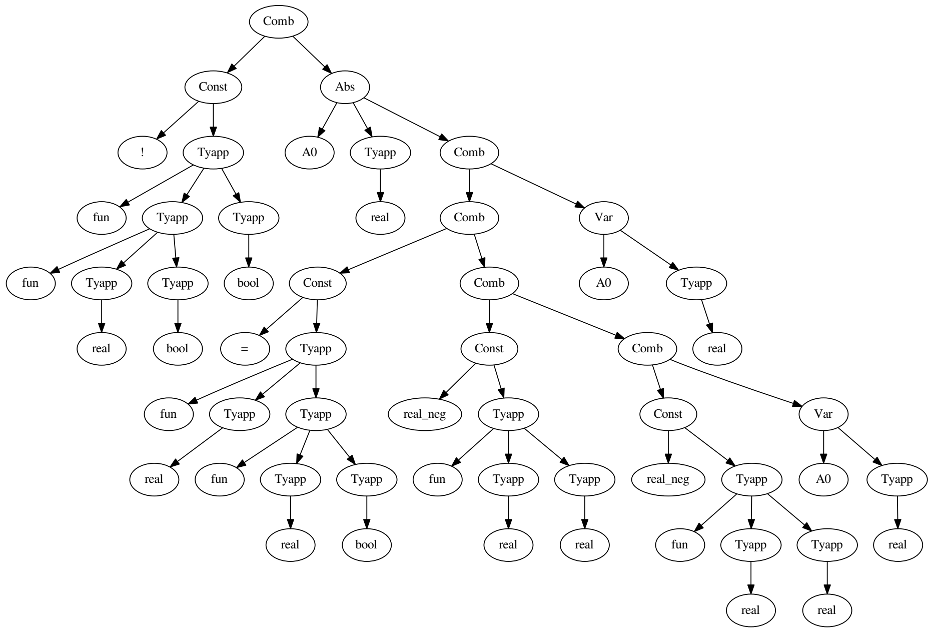

Fig. 1: The HOL Light parse tree of `REAL_NEGNEG`

Note that all overloaded symbols are disambiguated there, they are applied with the correct arity, and all terms are decorated with their result types. To solve the task, we allow (and assume) training on a sufficiently large corpus of such informal/formal pairs.

2.3 Probabilistic Context Free Grammars

Given a large corpus of corresponding informal/formal formulas, how can we train an AI system for parsing the next informal formula into a formal one? The informal-to-formal domain differs from natural-language domains, where millions of examples of paired (e.g., English/German) sentences are available for training machine translation. The natural languages also have many more words (concepts) than in mathematics, and the sentences to a large extent also lack the recursive structure that is frequently encountered in mathematics. Given that there are currently only thousands of informal/formal examples, purely statistical alignment methods based on n-grams seem inadequate. Instead, the methods have to learn how to compose larger parse trees from smaller ones based on those encountered in the limited number of examples.

A well-known approach ensuring such compositionality is the use of CFG (Context Free Grammar) parsers. This approach has been widely used, e.g., in word-sense disambiguation. A frequently used CFG algorithm is the CYK (Cocke–Younger–Kasami) chart-parser [32], based on bottom-up parsing. By default CYK requires the CFG to be in the Chomsky Normal Form (CNF). The transformation to CNF can cause an exponential blow-up of the grammar, however, an improved version of CYK gets around this issue [25].

In linguistic applications the input grammar for the CFG-based parsers is typically extracted from the *grammar trees* which correspond to the correct parses of natural-language sentences. Large annonated *treebanks* of such correct parses exist for natural languages. The grammar rules extracted from the treebanks are typically ambiguous: there are multiple possible parse trees for a particular sentence. This is why CFG is extended by adding a probability to each grammar rule, resulting in Probabilistic CFG (PCFG).

3 PCFG for the Informal-To-Formal Task

The most straightforward PCFG-based approach would be to directly use the native HOL Light parse trees (Fig. 1) for extracting the PCFG. However, terms and types are there annotated with only a few nonterminals such as: `Comb` (application), `Abs` (abstraction), `Const` (higher-order constant), `Var` (variable), `Tyapp` (type application), and `Tyvar` (type variable). This would lead to many possible parses in the context-free setting, because the learned rules are very universal, e.g:

```
    Comb -> Const Var.   Comb -> Const Const.   Comb -> Comb Comb.
```

The type information does not help to constrain the applications, and the last rule allows a series of several constants to be given arbitrary application order, leading to uncontrolled explosion.

3.1 HOL Types as Nonterminals

The approach taken in [21] is to first re-order and simplify the HOL Light parse trees to propagate the type information at appropriate places. This gives the context-free rules a chance of providing meaningful pruning information. For example, consider again the raw HOL Light parse tree for `REAL_NEGNEG` (Fig. 1).

Instead of directly extracting very general rules such as `Comb -> Const Abs`, each type is first compressed into an opaque nonterminal. This turns the parse tree of `REAL_NEGNEG` into (see also Fig. 2):

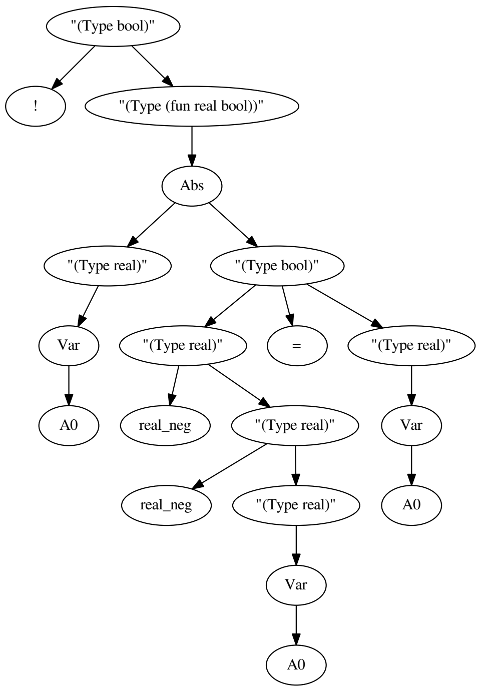

Fig. 2: Transformed tree of `REAL_NEGNEG`

```
("(Type bool)" ! ("(Type (fun real bool))" (Abs ("(Type real)" (Var A0)) ("(Type bool)"
("(Type real)" real_neg ("(Type real)" real_neg ("(Type real)" (Var A0)))) = ("(Type real)"
(Var A0))))))
```

The CFG rules extracted from this transformed tree thus become more targeted. For example, the two rules:

```
"(Type bool)" -> "(Type real)" = "(Type real)".
"(Type real)" -> real_neg "(Type real)".
```

say that equality of two reals has type `bool`, and negation applied to reals yields reals. Such learned *probabilistic typing rules* restrict the number of possible parses much more than the general "application" rules extracted from the original HOL Light tree. The rules still have a non-trivial generalization (learning) effect that is needed for the compositional behavior of the information extracted from the trees. For example, once we learn from the training data that the variable `‘‘u’’` is mostly parsed as a real number, i.e.:

```
"(Type real)" -> Var u.
```

we will be able to apply `real_neg` to `u` even if the subterm `real_neg u` has never yet been seen in the training examples, and the probability of this parse will be relatively high.

In other words, having the HOL types as *semantic categories* (corresponding e.g. to word senses when using PCFG for word-sense disambiguation) is a reasonable choice for the first experiments. It is however likely that even better semantic categories can be developed, based on more involved statistical and semantic analysis of the data such as *latent semantics* [5].

3.2 Semantic Concepts as Nonterminals

The last part of the original setting wraps ambiguous symbols, such as `--`, in their disambiguated *semantic/formal concept* nonterminals. In this case `$#real_neg` would be wrapped around `--` in the training tree when `--` is used as negation on reals. While the type annotation is often sufficient for disambiguation, such explicit disambiguation nonterminal is more precise and allows easier extraction of the HOL semantics from the constructed parse trees. The actual tree of `REAL_-NEGNEG` used for training the grammar is thus as follows (see also Fig. 3):

```
("(Type bool)" ! ("(Type (fun real bool))" (Abs ("(Type real)" (Var A0)) ("(Type bool)"
("(Type real)" ($#real_neg --) ("(Type real)" ($#real_neg --) ("(Type real)" (Var A0))))
($#= =) ("(Type real)" (Var A0))))))
```

3.3 Modified CYK Parsing and Its Initial Performance

Once the PCFG is learned from such data, the CYK algorithm augmented with fast internal semantic checks is used to parse the informal sentences. The semantic checks are performed to require compatibility of the types of free variables in parsed subtrees. The most probable parse trees are then typechecked by HOL Light. This is followed by proof and disproof attempts by the HOL(y)Hammer system [18], using all the semantic knowledge available in the Flyspeck library (about 22000 theorems). The first large-scale disambiguation experiment conducted over "ambiguated" Flyspeck in [21] showed that about 40% of the ambiguous sentences have their correct parses among the best 20 parse trees produced by the trained parser. This is encouraging, but certainly invites further research in improving the statistical/semantic parsing methods.

4 Limits of the Context-Free Grammars

A major limiting issue when using PCFG-based parsing algorithms is the context-freeness of the grammar. This is most obvious when using just the low-level term constructors as nonterminals, however it shows often also in the more advanced setting described above. In some cases, no matter how good are the training data, there is no way how to set up the probabilities of the parsing rules so that the required parse tree will have the highest probability. We show this on the following simple example.

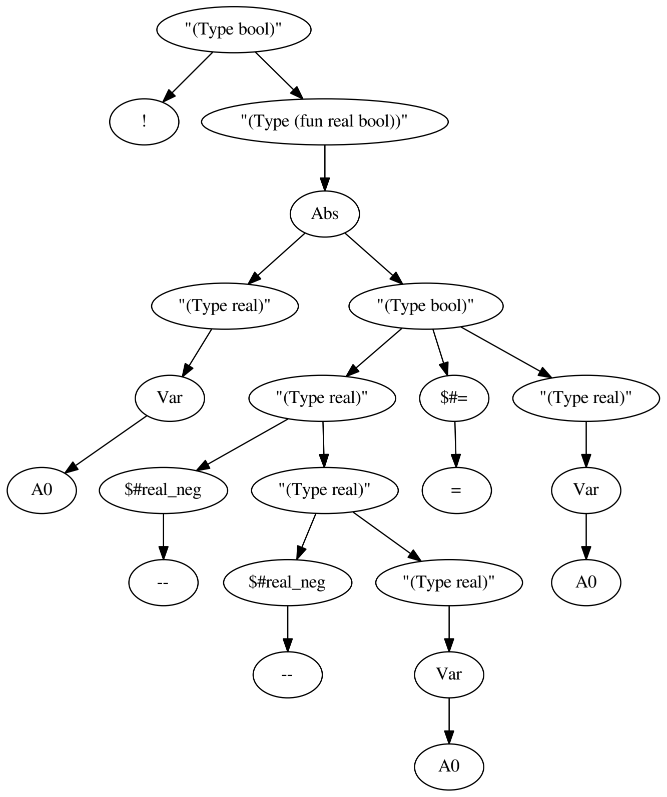

Fig. 3: The parse tree of `REAL_NEGNEG` used for the actual grammar training

Example: Consider the following term t:

```
1 * x + 2 * x.
```

with the following simplified parse tree $T_0(t)$ (see also Fig. 4).

```
(S (Num (Num (Num 1) * (Num x)) + (Num (Num 2) * (Num x))) .)
```

When used as the training data (treebank), the grammar tree $T_0(t)$ results in the following set of CFG rules $G(T_0(t))$:

```
S -> Num .                          Num -> 1
Num ->  Num + Num                   Num -> 2
Num -> Num * Num                    Num -> x
```

This grammar allows exactly the following five parse trees $T_4(t), ..., T_0(t)$ when used on the original (non-bracketed) term t:

```
(S (Num (Num 1) * (Num (Num (Num x) + (Num 2)) * (Num x))) .)
(S (Num (Num 1) * (Num (Num x) + (Num (Num 2) * (Num x)))) .)
(S (Num (Num (Num 1) * (Num (Num x) + (Num 2))) * (Num x)) .)
```

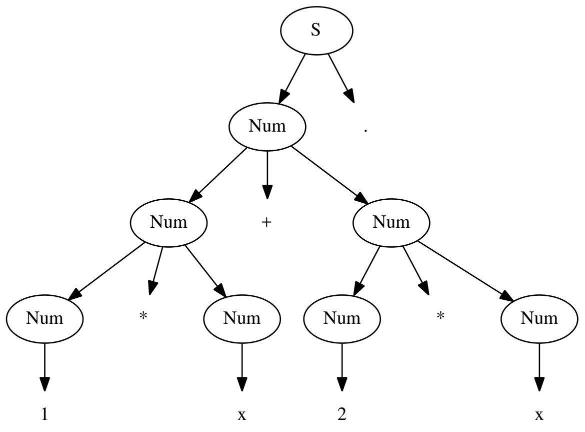

Fig. 4: The grammar tree $T_0(t)$.

```
(S (Num (Num (Num (Num 1) * (Num x)) + (Num 2)) * (Num x)) .)
(S (Num (Num (Num 1) * (Num x)) + (Num (Num 2) * (Num x))) .)
```

Here only the last tree corresponds to the original training tree $T_0(t)$. No matter what probabilities $p(Rule_i)$ are assigned to the grammar rules $G(T_0(t))$, it is not possible to make the priority of `+` smaller than the priority of `*`. A context-free grammar forgets the context and cannot remember and apply complex mechanisms such as priorities. The probability of all parse trees is thus in this case always the same, and equal to:

$$\begin{aligned} p(T_4(t)) = ... = p(T_0(t)) = p(\texttt{S -> Num .}) \times p(\texttt{Num -> Num + Num}) \\ \times p(\texttt{Num -> Num * Num}) \times p(\texttt{Num -> Num * Num}) \\ \times p(\texttt{Num -> 1}) \times p(\texttt{Num -> 2}) \times p(\texttt{Num -> x}) \times p(\texttt{Num -> x}) \end{aligned}$$

While the example's correct parse does not strictly imply the priorities of `+` and `*` as we know them, it is clear that we would like the grammar to prefer parse trees that are in some sense *more similar* to the training data. One method that is frequently used for dealing with similar problems in the NLP domain is *grammar lexicalization* [3]. There an additional terminal can be appended to nonterminals and propagated from the subtrees, thus creating many more possible (more precise) nonterminals. This approach however does not solve the particular problem with operator priorities. We also believe that considering probabilities of larger subtrees in the data as we propose below is conceptually cleaner than lexicalization.

5 Using Probabilities of Deeper Subtrees

Our solution is motivated by an analogy with the n-gram statistical machine-translation models, and also with the large-theory premise selection systems.

In such systems, characterizing formulas by all deeper subterms and subformulas is feasible and typically considerably improves the performance of the algorithms [20]. Considering subtrees of greater depth for updating the parsing probabilities may initially seem computationally involved. Below we however show that by using efficient ATP-style indexing datastructures such as discrimination trees, this approach becomes feasible, solving in a reasonably clean way some of the inherent problems of the context-free grammars mentioned above.

In more detail, our approach is as follows. We extract not just subtrees of depth 2 from the treebank (as is done by the standard PCFG), but all subtrees up to a certain depth. Other approaches – such as frequency-based rather than depth-based – are possible. During the (modified) CYK chart parsing, the probabilities of the parsed subtrees are adjusted by taking into account the statistics of such deeper subtrees extracted from the treebank. The extracted subtrees are technically treated as new "grammar rules" of the form:

root of the subtree `->` *list of the children of the subtree*

Formally, for a treebank (set of trees) $\mathbb{T}$, we thus define $G^n(\mathbb{T})$ to be the grammar rules of depth n extracted from $\mathbb{T}$. The standard context-free grammar $G(\mathbb{T})$ then becomes $G^2(\mathbb{T})$, and we denote by $G^{n,m}(\mathbb{T})$ where $n \leq m$ the union[1] $G^n(\mathbb{T}) \cup ... \cup G^m(\mathbb{T})$. The probabilities of these deeper grammar rules are again learned from the treebank. Our current solution treats the nonterminals on the left-hand sides as disjoint from the old (standard CFG) nonterminals when counting the probabilities (this can be made more complicated in the future). The right-hand sides of such new grammar rules thus contain larger subtrees, allowing to compute the parsing probabilities using more context/structural information than in the standard context-free case.

For the example term t from Section 4 this works as follows. After the extraction of all subtrees of depth 2 and 3 and the appropriate adjustment of their probabilities, we get a new extended set of probabilistic grammar rules $G^{2,3}(T_0(t)) \supset G(T_0(t))$. This grammar could again parse all the five different parse trees $T_4(t), ..., T_0(t)$ as in Section 4, but now the probabilities $p(T_4(t)), ..., p(T_0(t))$ would in general differ, and an implementation would be able to choose the training tree $T_0(t)$ as the most probable one. In the particular implementation that we use (see Section 5.1) its probability is:

$$\begin{aligned} p(T_0(t)) = p(\texttt{Num -> (Num 1)}) \times p(\texttt{Num -> (Num x)}) \times \\ p(\texttt{Num -> (Num 2)}) \times p(\texttt{Num -> (Num x)}) \times \\ p(\texttt{Num -> (Num Num * Num) + (Num Num * Num)}) \times \\ p(\texttt{S -> Num .}) \end{aligned}$$

Here the second line from the bottom stands for the probability of a subtree of depth 3. For the case of the one-element treebank $T_0(t)$, $p(T_0(t))$ would indeed be the highest probability. On the other hand, the probability of some of the

[1] In general, a grammar could pick only some subtree depths instead of their contiguous intervals, but we do not use such grammars now.

other parses (e.g., $T_4(t)$ and $T_3(t)$ above) would remain unmodified, because in such parses there are no subtrees of depth 3 from the training tree $T_0(t)$.

5.1 Efficient Implementation of Deeper Subtrees

Discrimination trees [27], as first implemented by Greenbaum [12], index terms in a trie, which keeps single path-strings at each of the indexed terms. A discrimination tree can be constructed efficiently, by inserting terms in the traversal preorder. Since discrimination trees are based on path indexing, retrieval of matching subtrees during the parsing is straightforward.

We use a discrimination tree D to store all the subtrees $G^{n,m}(\mathbb{T})$ from the treebank $\mathbb{T}$ and to efficiently retrieve them together with their probabilities during the chart parsing. The efficiency of the implementation is important, as we need to index about half a million subtrees in D for the experiments over Flyspeck. On the other hand, such numbers have become quite common in large-theory reasoning recently and do not pose a significant problem. For memory efficiency we use OCaml maps (implemented as AVL trees) in the internal nodes of D. The lookup time thus grows logarithmically with the number of trees in D, which is the main reason why we so far only consider trees of depth 3.

When a particular cell in the CYK parsing chart is finished (i.e., all its possible parses are known), the subtree-based probability update is initiated. The algorithm thus consists of two phases: (i) the standard collecting of all possible parses of a particular cell, using the context-free rules $G^2(\mathbb{T})$ only, and (ii) the computation of probabilities, which involves also the deeper (contextual) subtrees $G^{3,m}(\mathbb{T})$.

In the second phase, every parse P of the particular cell is inspected, trying to find its top-level subtrees of depths $3, ..., m$ in the discrimination tree D. If a matching tree T is found in D, the probability of P is recomputed, using the probability of T. There are various ways how to combine the old context-free and the new contextual probabilities. The current method we use is to take the maximum of the probabilities, keeping them as competing methods. As mentioned above, the nonterminals in the new subtree-based rules are kept disjoint from the old context-free rules when computing the grammar rule probabilities. The usual effect is that a frequent deeper subtree that matches the parse P gives it more probability, because such a “deeper context parse” replaces the corresponding two shallow (old context-free) rules, whose probabilities would have to be multiplied.

Our speed measurement with depth 3 has shown that the new implementation is (surprisingly) faster. In particular, when training on all 21695 Flypeck trees and testing on 11911 of them with the limit of 10 best parses, the new version is 23% faster than the old one (10342.75 s vs. 13406.97 s total time). In this measurement the new version also failed to produce at least a single parse less often than the old version (631 vs 818). This likely means that the deeper subtrees help to promote the correct parse, which in the context-free version is considered at some point too improbable to make it into the top 10 parses and consequently discarded.

depth	correct parse found (%)	avg. rank of correct parse
2	8998 (41.5)	3.81
3	11003 (50.7)	2.66
4	13875 (64.0)	2.50
5	14614 (67.4)	2.34
6	14745 (68.0)	2.13
7	14379 (66.2)	2.17

Table 1: Numbers of correctly parsed Flyspeck theorems within first 20 parses and their average ranks for subtree depths 2 to 7 of the parsing algorithm (100-fold cross-validation).

6 Experimental Evaluation

6.1 Machine Learning Evaluation

The main evaluation is done in the same cross-validation scenario as in [21]. We create the ambiguous sentences (Sec. 2) and the disambiguated grammar trees from all 21695 Flyspeck theorems,[2] permute them randomly and split into 100 equally sized chunks of about 217 trees and their corresponding sentences. The grammar trees serve for training and the ambiguous sentences for evaluation. For each testing chunk C_i ($i \in 1..100$) of 217 sentences we train the probabilistic grammar P_i on the union of the remaining 99 chunks of grammar trees (altogether about 21478 trees). Then we try to get the best 20 parse trees for all the 217 sentences in C_i using the grammar P_i. This is done for the simple context-free version (depth 2) of the algorithm (Section 3), as well as for the versions using deeper subtrees (Section 5). The numbers of correctly parsed formulas and their average ranks across the several 100-fold cross-validations are shown in Table 1.

It is clear that the introduction of deeper subtrees into the CYK algorithm has produced a significant improvement of the parsing precision. The number of correctly parsed formulas appearing among the top 20 parses has increased by 22% between the context-free (depth 2) version and the subtree-based version when using subtrees of depth 3, and it grows by 64% when using subtrees of depth 6.

The comparison of the average ranks is in general only a heuristic indicator, because the number of correct parses found differ so significantly between the methods.[3] However, since the number of parses is higher in the better-ranking methods, this improvement is also relevant. The average rank of the best subtree-based method (depth 6) is only about 56% of the context-free method. The

[2] About 1% of the longest Flyspeck formulas were removed from the evaluation to keep the parsing times manageable.

[3] If the context-free version parsed only a few terms, but with the best rank, its average rank would be 1, but the method would still be much worse in terms of the overall number of correctly parsed terms.

	subtree-2 (%)	subtree-6 (%)
at least one parse (limit 20)	14101 (82.9)	16049 (94.3)
at least one correct parse	5744 (33.8)	10735 (63.1)
at least one OLT parse	808 (4.7)	1584 (9.3)
at least one parse proved	5682 (33.3)	7538 (44.3)
correct parse proved	1762 (10.4)	2616 (15.4)
at least one OLT parse proved	525 (3.1)	814 (4.8)
the first parse proved is correct	1168 (6.7)	2064 (12.1)
the first parse proved is OLT	332 (2.0)	713 (4.2)

Table 2: Statistics of the ATP evaluation for subtree-2 and subtree-6. The total number of theorems tried is 17018 and we require 20 best parses. OLT stands for other library theorem.

results of the best method say that for 68% of the theorems the correct parse of an ambiguous statement is among the best 20 parses, and its average rank among them is 2.13.

6.2 ATP Evaluation

In the ATP evaluation we measure how many of the correctly parsed formulas the HOL(y)Hammer system can prove, and thus help to confirm their validity. While the machine-learning evaluation is for simplicity done by randomization, regardless of the chronological order of the Flyspeck theorems, in the ATP evaluation we only allow facts that were already proved in Flyspeck before the currently parsed formula. Otherwise the theorem-proving task becomes too easy, because the premise-selection algorithm will likely select the theorem itself as the most relevant premise. Since this involves large amount of computation, we only compare the best new subtree-based method (depth 6) from Table 1 (*subtree-6*) with the old context-free method (*subtree-2*).

In the ATP evaluation, the number of the Flyspeck theorems is reduced from 21695 to 17018. This is due to omitting definitions and duplicities during the chronological processing and ATP problem generation. For actual theorem proving, we only use a single (strongest) HOL(y)Hammer method: the distance-weighted k-nearest neighbor (k-NN) [6] using the strongest combination of features [20] with 128 premises and running Vampire 4.0 [24]. Running the full portfolio of 14 AI/ATP HOL(y)Hammer strategies for hundreds of thousands problems would be too computationally expensive.

Table 2 shows the results. In this evaluation we also detect situations when an ambiguated Flyspeck theorem T_1 is parsed as a different known Flyspeck theorem T_2. We call the latter situation *other library theorem (OLT)*. The removal of definitions and duplicitites made the difference in the top-20 correctly parsed sentences even higher, going from 33.8% for subtree-2 to 63.1% in subtree-6. This is an improvement of 86.9%. A correspondingly high increase between subtree-2 and subtree-6 is also in the number of situations when the first parse is correct (or OLT) and HOL(y)Hammer can prove it using previous Flyspeck facts. The much greater easiness of proving existing library theorems than proving new theorems explains the high number of provable OLTs when compared to their total number of occurences. Such OLT proofs are however easy to filter out when using HOL(y)Hammer as a semantic filter for the informal-to-formal translation.

7 Conclusion and Future Work

In this paper, we have introduced our project aiming at automated learning of formal understanding of mathematics. In comparison to our first results [21], we have introduced efficient context-based learning and parsing, which significantly increases the success rate of the informal-to-formal translation task on the Flyspeck corpus. The overall improvement in the number of correct parses among the top 20 is 64%, and even higher (86.9%) when omitting duplicities and definitions. The average rank of the correct parse has decreased to about 56% of the previous approach. We believe that the contextual approach to enhancing CYK we took is rather natural (in particular more natural than lexicalization), the discrimination tree indexing scales to this task, and the performance increase is very impressive.

Future work includes adding further semantic checks and better probabilistic ranking subroutines directly into the parsing process. The chart-parsing algorithm is easy to extend with such checks and subroutines, and already the current semantic pruning of parse trees that have incompatible variable types is extremely important. While some semantic relations might eventually be learnable by less efficient learning methods such as recurrent neural networks (RNNs), we believe that the current approach allows more flexible experimenting and non-trivial integration and feedback loops between advanced deductive and learning components. A possible use of RNNs in such a setup is for better ranking of subtrees and for global focusing of the parsing process.

An example of a more sophisticated deductive algorithm that should be easy to integrate is congruence closure over provably equal (or equivalent) parsing subtrees. For example, `''a * b * c''` can be understood with different bracketing, different types of the variables and different interpretations of `*`. However, `*` is almost always associative across all types and interpretations. Human readers know this, and rather than considering differently bracketed parses, they focus on the real problem, i.e., which types to assign to the variables and how to interpret the operator in the current context. To be able to emulate this ability, we would cache directly in the chart parsing algorithm the results of large-theory ATP runs on many previously encountered equalities, and use them for fast congruence closure over the subtrees.

Similar ATP/logic-based components also seem necessary for dealing with more involved type systems and human-like parsing layers, such as the one used by the Mizar system. Our first experiments in combining the contextual parsing with ATPs to deal with phenomena like hidden variables and intersection types are described in [19].

References

1. G. Bancerek and P. Rudnicki. A Compendium of Continuous Lattices in MIZAR. *J. Autom. Reasoning*, 29(3-4):189–224, 2002.
2. J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.
3. M. Collins. Three generative, lexicalised models for statistical parsing. In P. R. Cohen and W. Wahlster, editors, *Proc. 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23. Morgan Kaufmann Publishers / ACL, 1997.
4. The Coq Proof Assistant. `http://coq.inria.fr`.
5. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by Latent Semantic Analysis. *JASIS*, 41(6):391–407, 1990.
6. S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(4):325–327, 1976.
7. F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. Packaging mathematical structures. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*, volume 5674 of *LNCS*, pages 327–342. Springer, 2009.
8. T. Gauthier and C. Kaliszyk. Matching concepts across HOL libraries. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of *LNCS*, pages 267–281. Springer Verlag, 2014.
9. G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the Odd Order Theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013.
10. G. Gonthier and E. Tassi. A language of patterns for subterm selection. In L. Beringer and A. P. Felty, editors, *Interactive Theorem Proving - Third International Conference, ITP 2012*, volume 7406 of *LNCS*, pages 361–376. Springer, 2012.
11. A. Grabowski, A. Korniłowicz, and A. Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.
12. S. Greenbaum. *Input transformations and resolution implementation techniques for theorem-proving in first-order logic.* PhD thesis, University of Illinois at Urbana-Champaign, 1986.
13. F. Haftmann and M. Wenzel. Constructive type classes in isabelle. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006*, volume 4502 of *LNCS*, pages 160–174. Springer, 2006.
14. T. Hales. *Dense Sphere Packings: A Blueprint for Formal Proofs*, volume 400 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 2012.
15. T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, T. Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. H. T. Ta, T. N. Tran, D. T. Trieu, J. Urban, K. K. Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015.

16. J. Harrison. HOL Light: A tutorial introduction. In M. K. Srivas and A. J. Camilleri, editors, *FMCAD*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.
17. J. Harrison, J. Urban, and F. Wiedijk. History of interactive theorem proving. In J. H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 135–214. Elsevier, 2014.
18. C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014.
19. C. Kaliszyk, J. Urban, and J. Vyskocil. System description: Statistical parsing of informalized Mizar formulas. Submitted, available at `http://grid01.ciirc.cvut.cz/~mptp/synasc17sd.pdf`.
20. C. Kaliszyk, J. Urban, and J. Vyskocil. Efficient semantic features for automated reasoning over large theories. In Q. Yang and M. Wooldridge, editors, *IJCAI'15*, pages 3084–3090. AAAI Press, 2015.
21. C. Kaliszyk, J. Urban, and J. Vyskocil. Learning to parse on aligned corpora (rough diamond). In C. Urban and X. Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015*, volume 9236 of *LNCS*, pages 227–233. Springer, 2015.
22. C. Kaliszyk, J. Urban, J. Vyskocil, and H. Geuvers. Developing corpus-based translation methods between informal and formal mathematics: Project description. In S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics - International Conference, CICM 2014*, volume 8543 of *LNCS*, pages 435–439. Springer, 2014.
23. G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an operating-system kernel. *Com. ACM*, 53(6):107–115, 2010.
24. L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
25. M. Lange and H. Leiß. To CNF or not to CNF? an efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2009.
26. X. Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.
27. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
28. P. Rudnicki, C. Schwarzweller, and A. Trybulec. Commutative algebra in the Mizar system. *J. Symb. Comput.*, 32(1/2):143–169, 2001.
29. C. Tankink, C. Kaliszyk, J. Urban, and H. Geuvers. Formal mathematics on display: A wiki for Flyspeck. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *MKM/Calculemus/DML*, volume 7961 of *LNCS*, pages 152–167. Springer, 2013.
30. J. Urban and J. Vyskočil. Theorem proving in large formal mathematics as an emerging AI field. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William McCune*, volume 7788 of *LNAI*, pages 240–257. Springer, 2013.
31. M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In O. A. Mohamed, C. A. Muñoz, and S. Tahar, editors, *TPHOLs*, volume 5170 of *LNCS*, pages 33–38. Springer, 2008.
32. D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.
33. C. Zinn. *Understanding informal mathematical discourse.* PhD thesis, University of Erlangen-Nuremberg, 2004.