Improving Statistical Linguistic Algorithms for Parsing Mathematics

Cezary Kaliszyk [*]
University of Innsbruck
cezary.kaliszyk@uibk.ac.at

Josef Urban [†]
Czech Technical University in Prague
josef.urban@gmail.com

Jiří Vyskočil [†]
Czech Technical University in Prague
jiri.vyskocil@gmail.com

Abstract

In this paper we describe our combined statistical/semantic parsing method based on the CYK chart-parsing algorithm augmented with limited internal typechecking and external ATP filtering. This method was previously evaluated on parsing ambiguous mathematical expressions over the informalized Flyspeck corpus of 20000 theorems. We first discuss the motivation and drawbacks of the first version of the CYK-based component of the algorithm, and then we propose and implement a more sophisticated approach based on better statistical model of mathematical data structures.

1 Introduction

Computer-understandable (formal) mathematics is today still quite far from taking over the mathematical mainstream. Despite the impressive formalizations such as Flyspeck [5], Feit-Thompson [4], seL4 [11], CompCert [13], and CCL [2], and the progress in general automation over such large formal corpora, formalizing proofs is still largely unappealing to mathematicians. While the research on AI and strong automation over large theories has taken off in the last decade and automation improvements are today coming from several directions, there has been so far very little progress in automating the understanding of informal LaTeX-written and ambiguous mathematical writings.

Recently, we have proposed to try to change this state of affairs by learning how to parse informal mathematics from aligned informal/formal corpora [10]. Such learning can be additionally combined with strong semantic filtering methods such as typechecking and large-theory ATP. Suitable aligned corpora are appearing today, the major example being Flyspeck and in particular its alignment (by Hales) with the detailed informal Blueprint for Formal Proofs [5]. Very recently [9] we have implemented the first version of a statistical/semantic parsing toolchain that learns parsing rules from many pairs of ambiguous/nonambiguous Flyspeck formulas, and combines statistical parsing of new ambiguous formulas with internal semantic pruning and external proving/disproving step. The resulting parsing/proving system trained on all of Flyspeck is available online[1] and can be used for experimenting with parsing ambiguous statements.

In this short paper we explain in more detail the particular statistical learning/parsing approach based on the CYK chart parsing algorithm [14] for probabilistic context-free grammars (PCFG) that we have been using (Sec. 2.1), and focus on some drawbacks of the context-free approach that can negatively influence the statistical learning and parsing performance (Sec. 2.2). We demonstrate this on a simple example, where the PCFG setting is not strong

[*]Supported by the Austrian Science Fund (FWF): P26201.

[†]This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 649043 – AI4REASON).

[1]http://colo12-c703.uibk.ac.at/hh/parse.html

enough to eventually learn the correct parsing (Sec. 3.1). Then we propose and implement a modification of CYK that takes into account larger parsing subtrees and their probabilities (Sec. 3.2). This modification is motivated by an analogy with large-theory reasoning systems. There, the precision of the probabilistic selection of the right premises for a new conjecture can typically be significantly improved by considering the large number of term and formula subtrees of the data, rather than just characterizing formulas and their similarity by the bare symbols appearing in them. Finally, we report the first measurements done with the new implementation and discuss future work (Sec. 3.3).

2 Training Statistical Parsing on Aligned Corpora

2.1 PCFG

Given a large corpus of corresponding informal/formal (ambiguous/nonambiguous or LaTeX/HOL) formulas, how do we automatically train an AI system that will correctly parse the next informal formula into a formal one?

Our domain differs from the natural-language domains, where millions of examples of paired (e.g., English/German) sentences are available for training machine translation, the languages 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 we currently have only thousands of the informal/formal examples, we have decided against using purely statistical alignment methods based on n-grams, and rather investigated methods that can learn how to compose larger parse trees from smaller ones based on those encountered in the limited number of examples that we have.

One well-known approach ensuring this kind of compositionality is the use of CFG (Context Free Grammar) parsers. This approach has been widely used, e.g., for word-sense disambiguation in natural languages, which is another linguistic area close to our informal/formal task. An advantage of this approach is the existence of a parsing algorithm that works in polynomial complexity wrt. the size of the parsed sentence and the input grammar. A well-known and frequently used example is the CYK (Cocke–Younger–Kasami) chart-parsing algorithm [14], using bottom-up parsing and dynamic programming. By default CYK requires the CFG to be in the Chomsky Normal Form (CNF), and the transformation to CNF can cause an exponential blow-up of the grammar. However, an improved version of CYK can be used that gets around this issue [12].

A CFG-based parser obviously needs for its work the *input grammar* – the set of all grammar rules that can be used for parsing. In linguistic applications such grammar is typically extracted from *grammar trees* which correspond to the correct parses of natural-language sentences. Great efforts have been made in the linguistic community to create large *treebanks* of such correct parses – typically taking years of manual annotation work. 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 PCFG (Probabilistic CFG). During the PCFG parsing of an ambiguous sentence, each resulting parse tree is assigned its probability, which allows to focus on the few parses that are most probable wrt. the treebank of training examples. The grammar rule probabilities can be trained e.g. by the inside-outside algorithm [1].

2.2 Using PCFG for learning informal/formal alignment

We have so far experimented with several ways how to set up the parsing grammar and its learning. The most low-level approach consists of using the simplest HOL Light lambda calculus internal term structure [6], where terms and types are 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 has led to many possible parses in the context-free setting, because the top-level learned rules become very universal, e.g:

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

So far, 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.

That is why we have first re-ordered and simplified the HOL Light parse trees to propagate the type information at appropriate places where the context-free rules have a chance of providing meaningful pruning information. For example, the raw HOL Light parse tree for theorem

```
REAL_NEGNEG: !x.  --(--x) = x
```

is as follows (see also the tree in Fig. 1):

```
(Comb (Const "!" (Tyapp "fun" (Tyapp "fun" (Tyapp "real") (Tyapp "bool")) (Tyapp "bool")))
(Abs "A0" (Tyapp "real") (Comb (Comb (Const "=" (Tyapp "fun" (Tyapp "real") (Tyapp "fun"
(Tyapp "real") (Tyapp "bool")))) (Comb (Const "real_neg" (Tyapp "fun" (Tyapp "real") (Tyapp
"real"))) (Comb (Const "real_neg" (Tyapp "fun" (Tyapp "real") (Tyapp "real"))) (Var "A0"
(Tyapp "real"))))) (Var "A0" (Tyapp "real")))))
```

Note that the CFG rules for this tree are often very general: e.g., the top-level node produces the rule `Comb -> Const Abs`, etc. After our re-ordering and simplification the parse tree used for grammar generation becomes (see also Fig. 2):

```
("(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 become quite a bit more meaningful, e.g., 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 "typing" rules restrict the number of possible parses much more than the general "application" rules extracted from the original HOL Light tree, while still having 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 that the variable "u" is mostly parsed as a real number, we will be able to apply `real_neg` to "u" even if the particular subterm `''-- 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 the word

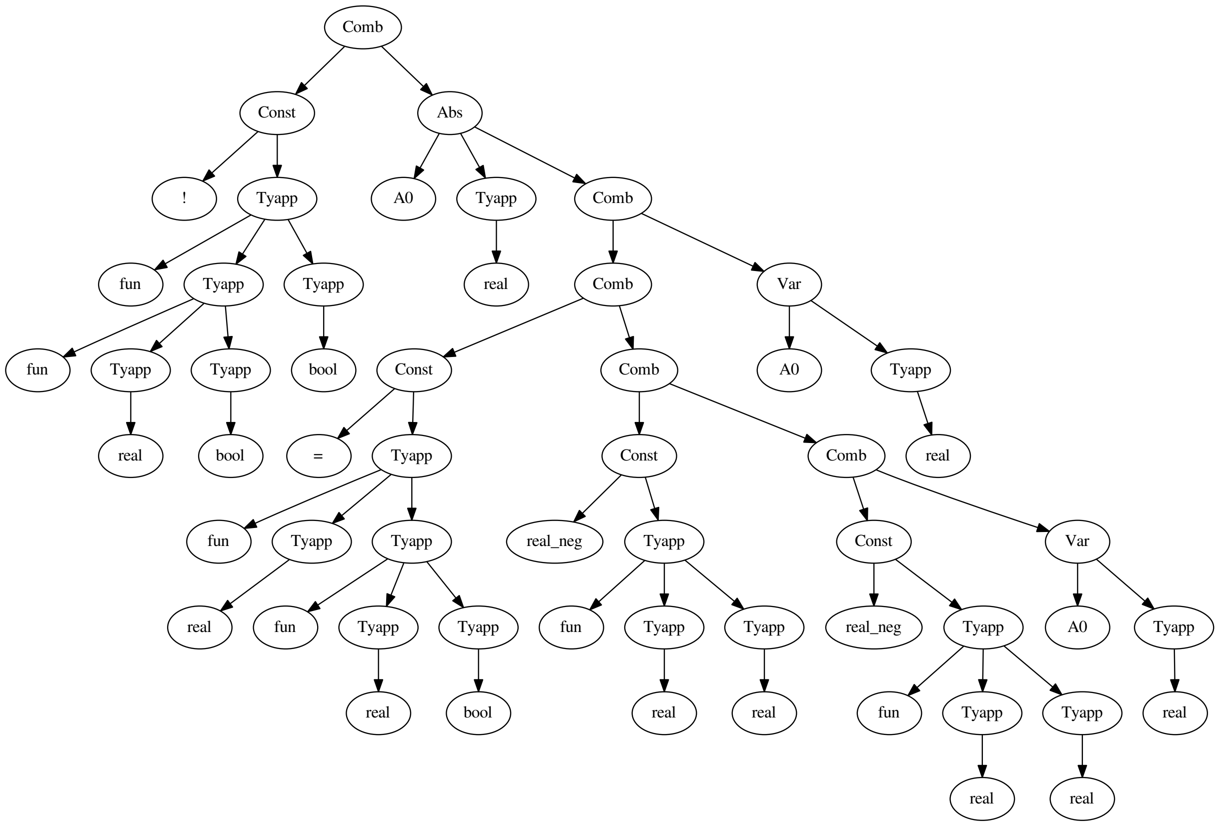

Figure 1: HOL Light parse tree

senses when using PCFG for word-sense disambiguation) seems to be quite a reasonable first choice for the experiments, even though one could probably come up with even more appealing semantic categories based on more involved statistical and semantic analysis of the data.

We should also note that ambiguous notation, such as `‘‘--’’`, is wrapped in the training trees in its disambiguated “semantic” nonterminal – in this case `$#real_neg`. While the type annotation might often be sufficient for disambiguation, such explicit disambiguation nonterminal is both more precise and allows easier extraction of the HOL semantics from the constructed parse trees. The actual tree 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))))))
```

Once the PCFG is learned from such data, we use the CYK algorithm with additional internal lightweight semantic checks to parse ambiguous formulas. These semantic checks are performed to require compatibility of the types of free variables in parsed subtrees. The most probable parse trees are then given to HOL Light and typechecked there, which is followed by proof and disproof attempts by the HOL(y)Hammer system [7], using all the semantic knowledge available in the Flyspeck library (about 22k theorems). See Fig. 4 for the overall structure of the system. The first large-scale disambiguation experiment conducted over “ambiguated” Flyspeck in [9] 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.

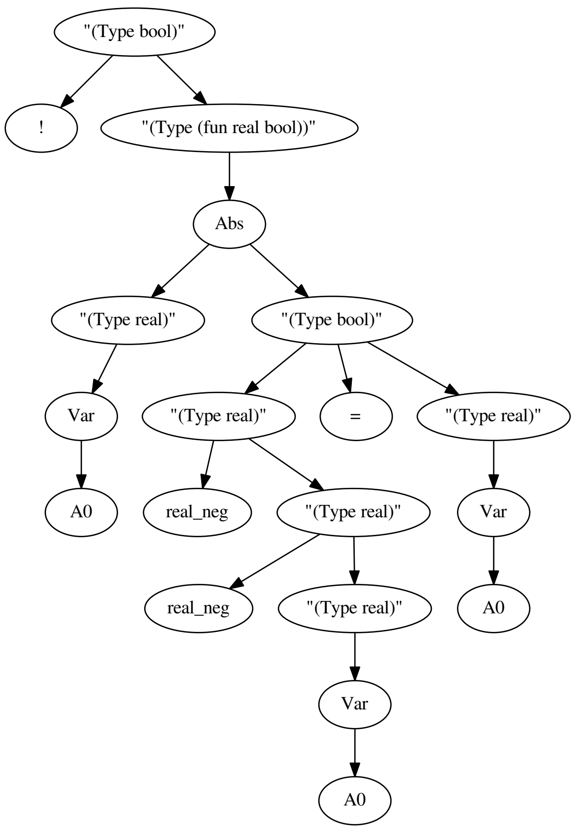

Figure 2: Transformed tree of `REAL_NEGNEG`

3 Adding Context

A major limiting issue when using PCFG-based parsing algorithms is the context-freeness of the grammar. In some cases, no matter how good are the training data, there is no way how to set up the parsing rules probabilities so that the required parse will have the largest probability.

3.1 Example

Consider the following term:

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

with the following simplified grammar tree (Fig. 5) as our training data (treebank):

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

From the grammar tree we extract the following CFG:

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

If we use this grammar for parsing the original (non-bracketed) sentence, we obtain the following five possible parse trees:

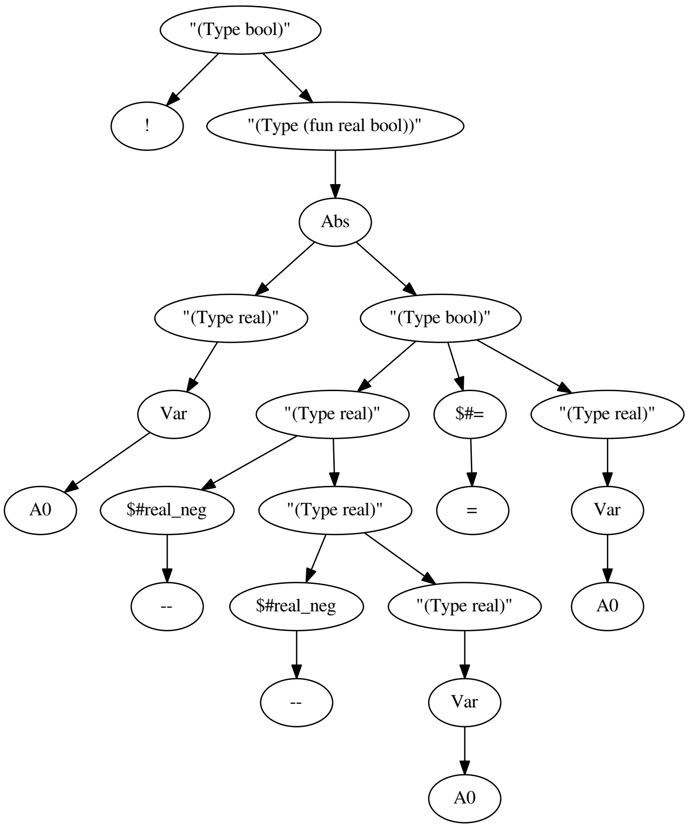

Figure 3: The tree of `REAL_NEGNEG` used for actual grammar training

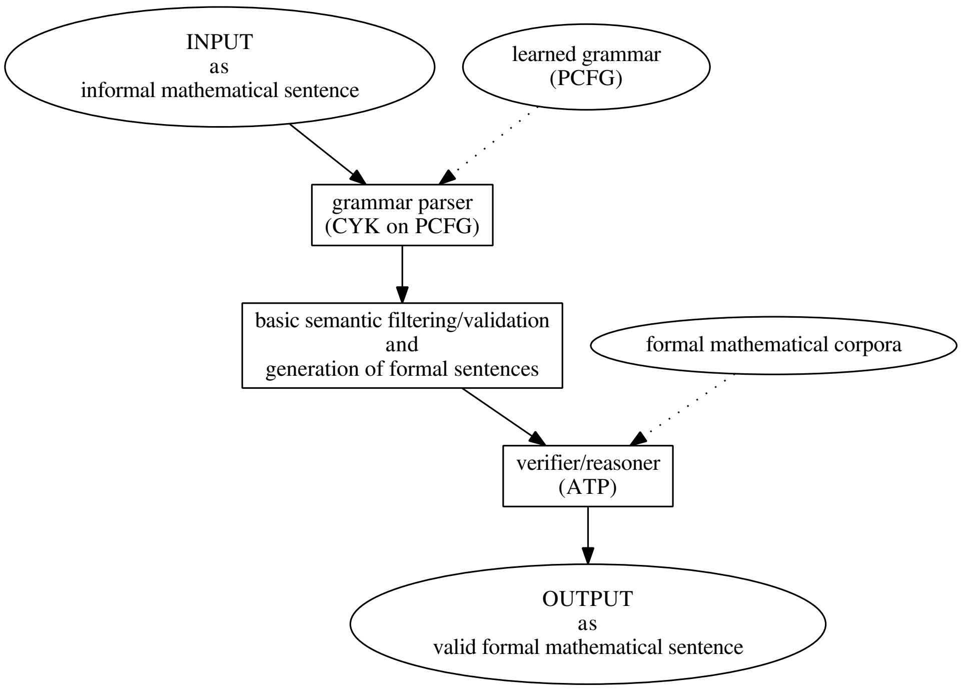
INPUT as informal mathematical sentence
learned grammar (PCFG)
grammar parser (CYK on PCFG)
basic semantic filtering/validation and generation of formal sentences
formal mathematical corpora
verifier/reasoner (ATP)
OUTPUT as valid formal mathematical sentence

Figure 4: The statistical/semantic parsing toolchain.

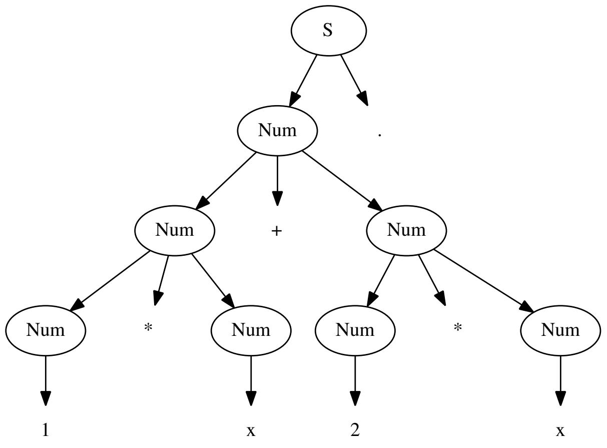

Figure 5: Example grammar tree

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

Only the last tree however corresponds to the training tree. The problem is that no matter what probabilities we add to the grammar rules, we cannot 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 parsed trees is in this case always the same:

$$p(\texttt{S -> Num .}) \times p(\texttt{Num -> Num + Num}) \times p(\texttt{Num -> Num * Num}) \times p(\texttt{Num -> Num * Num}) \times \\ p(\texttt{Num -> 1}) \times p(\texttt{Num -> 2}) \times p(\texttt{Num -> x}) \times p(\texttt{Num -> x})$$

While the example does not strictly imply the priorities 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] where 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 proposed below is conceptually cleaner.

3.2 Considering Subtrees

The underlying idea is a simple analogy with the n-gram statistical machine-translation models, or with the large-theory premise selection systems where characterizing formulas by all subterms and subformulas typically considerably improves the performance of the algorithms [8]. While considering subtrees may initially seem computationally involved, we believe that by using good indexing datastructures it becomes feasible, solving some the PCFG problems mentioned above in a reasonably clean way.

In more detail, the idea is as follows. We will extract not just subtrees of depth 2 from the treebank (as is done by PCFG), but all subtrees of certain depth. So far we work with depth 3, but other depths and approaches (e.g., frequency-based rather than depth-based) are possible. During the CYK parsing we will adjust the probabilities of the parsed subtrees also according to the subtree statistics extracted from the treebank. The extracted subtrees will be technically treated as new "grammar rules" of the form:

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

We will learn the probabilities of these new grammar rules, formally treating the nonterminals on the left-hand side as different from the old nonterminals when counting the probabilities (this is the current technical solution, which can be modified in the future). Since the right-hand side of the new grammar rules contains whole subtrees, we will be able to compute the parsing probabilities using more context/structural information than in PCFG.

In our example, after the extraction of all subtrees of depth 3 followed by a suitable adjustment of their probabilities, we would get a new "extended PCFG". This grammar could again parse all the five different parse trees as above, but they would have different probabilities, and the training tree would obviously be the most probable one. For example the probability of the original treebank parse would be:

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

On the other hand, the probability of some of the parses (e.g., the first two when using the original algorithm) would remain unmodified, because in these parses there are no subtrees of depth 3 from the training tree.

3.3 Technical Implementation

We use a discrimination tree D to store the subtrees from the treebank and to quickly look them up during the chart parsing. When a particular cell in the chart is finished (we know all its parses), we go through all its parses and try to look up their subtree of depth 3 in the discrimination tree D. If we succeed, we recompute the probability according to the new "subtree grammar rule", compare the resulting probability with the old one, and keep the better one.

The subtree lookup is logarithmic, however the number of subtrees we may need to look up can grow quite a lot in the worst case. This is why we have kept the depth at 3 so far. We have not done an extensive evaluation yet, however preliminary experiments with depth 3 and limiting CYK to the best 10 parses show that the new implementation is actually a bit faster than the old one. In particular, when training on all 21873 Flypeck trees and testing on 11911 of them, the new version is about 23% faster than the old one (10342.75s vs 13406.97s total time). The new version also fails to produce at least a single parse less often than the old version (631 vs 818).

This likely means that the subtrees help to promote the correct parse, which in the old version is considered at some point too improbable to make it into the top 10 parses and consequently thrown away by the greedy optimization. The correct (training) parse appears among the best 10 parses in 39% of the 11911 examples for the old algorithm (their average rank there being

2.68), and in 58% cases of the 11911 examples for the new algorithm (their average rank there being 1.97). While this is just a preliminary evaluation where we use the training data also for testing and do not run external typechecking and ATPs, this improvemnt in the parsing precision is very promising. Thorough experimental evaluation and further optimization of the set of subtrees used by the algorithm is future work.

References

[1] James K. Baker. Trainable grammars for speech recognition. In *Speech communication papers presented at the 97th Meeting of the Acoustical Society*, pages 547–550, 1979.

[2] Grzegorz Bancerek and Piotr Rudnicki. A Compendium of Continuous Lattices in MIZAR. *J. Autom. Reasoning*, 29(3-4):189–224, 2002.

[3] Michael Collins. Three generative, lexicalised models for statistical parsing. In Philip R. Cohen and Wolfgang Wahlster, editors, *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, 7-12 July 1997, Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain.*, pages 16–23. Morgan Kaufmann Publishers / ACL, 1997.

[4] 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 Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013.

[5] Thomas Hales. *Dense Sphere Packings: A Blueprint for Formal Proofs*, volume 400 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 2012.

[6] John Harrison. HOL Light: A tutorial introduction. In Mandayam K. Srivas and Albert John Camilleri, editors, *FMCAD*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.

[7] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014.

[8] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *IJCAI'15*, pages 3084–3090. AAAI Press, 2015.

[9] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning to parse on aligned corpora (rough diamond). In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 227–233. Springer, 2015.

[10] Cezary Kaliszyk, Josef Urban, Jiří Vyskočil, and Herman Geuvers. Developing corpus-based translation methods between informal and formal mathematics: Project description. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *LNCS*, pages 435–439. Springer, 2014.

[11] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: formal verification of an operating-system kernel. *Commun. ACM*, 53(6):107–115, 2010.

[12] Martin Lange and Hans Leiß. To CNF or not to CNF? an efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2009.

[13] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

[14] Daniel H. Younger. Recognition and parsing of context-free languages in time nˆ3. *Information and Control*, 10(2):189–208, 1967.