Lemmas: Generation, Selection, Application[⋆]

Michael Rawson[1], Christoph Wernhard[2], Zsolt Zombori[3], and Wolfgang Bibel[4]

[1] TU Wien, Austria michael@rawsons.uk
[2] University of Potsdam, Germany info@christophwernhard.com
[3] Alfréd Rényi Institute of Mathematics, Hungary zombori@renyi.hu
[4] Technical University Darmstadt, Germany bibel@gmx.net

Abstract. Noting that lemmas are a key feature of mathematics, we engage in an investigation of the role of lemmas in automated theorem proving. The paper describes experiments with a combined system involving learning technology that generates useful lemmas for automated theorem provers, demonstrating improvement for several representative systems and solving a hard problem not solved by any system for twenty years. By focusing on condensed detachment problems we simplify the setting considerably, allowing us to get at the essence of lemmas and their role in proof search.

1 Introduction

Mathematics is built in a carefully structured way, with many disciplines and subdisciplines. These are characterized by concepts, definitions, axioms, theorems, lemmas, and so forth. There is no doubt that this inherent structure of mathematics is part of the discipline's long-lasting success.

Research into Automated Theorem Proving (ATP) to date has taken little notice of the information provided by this structure. Even state-of-the-art ATP systems ingest a conjecture together with pertinent definitions and axioms in a way completely agnostic to their place in the mathematical structure. A comparatively small but nevertheless important part of the structure of mathematics is the identification and application of *lemmas*. It is this aspect which is the focus of the work presented here.

The purpose of lemmas in mathematics is at least threefold. First, and perhaps most importantly, lemmas support the search for proofs of assertions. If some lemma applies to a given problem, a proof may be found more easily. Second, it is often the case that a lemma may be applied more than once. If this happens, its use will shorten the length of the overall proof since the proof of the lemma need only be carried out once, not repeatedly for every application.

[⋆] Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 457292495, by the North-German Supercomputing Alliance (HLRN), by the ERC grant CoG ARTIST 101002685, by the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008 and the Hungarian Artificial Intelligence National Laboratory Program (RRF-2.3.1-21-2022-00004).

Third, the structuring effect of proofs by the use of lemmas is an important feature for human comprehension of proofs. In our work we are motivated primarily by the first two of these three aspects.

These considerations give rise to the crucial question: how can we find useful lemmas for proving a given problem? Here we mean useful in the sense of the two aforementioned aspects: lemmas should be applicable to the problem at hand, preferably many times. In full generality this is a difficult question indeed, which will require much further research. In this first step we restrict the question to a narrow range of problems, known in literature as *condensed detachment* (CD) problems [41]. Proofs of CD problems can be represented in a simple and accessible form as *proof structure terms*, enabling structure enumeration to enhance proof search and lemma maintenance, as well as feature extraction for learning. Our investigation thus focuses on the question of how ATP performance may be improved for CD problems by the generation and selection of useful lemmas before search begins.

CD problems are of the form "axiom(s) and *Det* imply a goal" where *Det* represents the well-known modus ponens rule, or *condensed detachment.* They have a single unary predicate. A typical application is the investigation of an axiomatization of some propositional logic, whose connectives are then represented by function symbols. In order to support this study experimentally, we have built a combined system for dealing with these problems. It features SGCD [75] as prover and lemma generator along with a learning module based on either an easily-interpreted linear model over hand-engineered features, or a graph neural network supporting end-to-end learning directly from lemmas.

Our work results in a number of inter-related particular contributions:

1. Incorporation of proof structure terms into ATP with Machine Learning (ML). Consideration of features of the proof structure terms, explicitly in linear-model ML or implicitly in a neural ML model. A novel ATP/ML dataflow that is centered around proof structure terms.
2. Experimentally validated general insights into the use of learned lemmas for provers of different paradigms, with different ways to incorporate lemmas, and based on two alternate ML models. At the same time pushing forward the state of the art on proving CD problems. Insights include: SGCD is competitive with leading first-order provers; Learned lemmas significantly extend the set of problems provable by the leading first-order prover Vampire; Provers without internal lemma maintenance, such as Connection Method (CM) [6,7,8] systems, are drastically improved; Vampire and SGCD are able to handle a few hundreds of supplied lemmas; Learning based on manual features and on automatic feature extraction perform similarly.
3. An automatic proof of the Meredith single axiom theorem `LCL073-1`, which has persisted in the TPTP rated 1.00 since 1997. The first and only system to succeed was OTTER [39], after intensive massaging by Wos [85]. It was proven by SGCD in a novel systematic way.
4. An implemented framework with the new techniques for generation, selection and application of lemmas.

Structure of the Paper. Section 2 presents condensed detachment and its embedding into the CM by way of so-called *D-terms*, as well as background material on lemmas and machine learning in ATP. Section 3 introduces a method for generating and selecting useful lemmas and presents experimental results with it, leading up to the proof of `LCL073-1` in Sect. 4. We conclude with a summary and outlook for further work in this area in Sect. 5. Supplementary material is provided in the appendix of the submission and will be made publicly available in a preprint version. All experiments are fully reproducible and the artefacts are available at https://github.com/zsoltzombori/lemma, commit `df2faaa`. We use CD Tools [75] and PIE [72,73], implemented in SWI-Prolog [78], for reasoning tasks and PyTorch [48] for learning.

2 Background and Related Work

In a very general sense, lemmas in ATP factorize duplication. This may be between different proofs that make use of the same lemma, or within a single proof where a lemma is used multiple times. It may not even be a particular formula that is shared, but a *pattern*, such as a *resonator* [82]. In the presence of machine learning, we may think of even more abstract entities that are factorized: the *principles* by which proofs are written, repeated in different proofs or contexts.

Depending on the proving method, lemmas in ATP play different roles. Provers based on *saturation*, typically resolution/superposition (RS) systems [3], inherently operate by generating lemmas: a resolvent is itself a lemma derived from its parents. Nevertheless, one may ask for more meaningful lemmas than the clauses of the proof. This is addressed with *cut introduction* [79,20,13], which studies methods to obtain complex lemmas from resolution proofs. Such lemmas provide insight about the high-level structure of proofs, extract interesting concepts and support research into the correspondence between natural mathematical notions and possible proof compressions. Other approaches to interesting theorems or lemmas are described for example in [65,53].

Another question concerning lemmas and ATP systems is whether performance can be improved by supplementing the input with lemmas. This is particularly applicable if lemmas are obtained with methods that are *different* from those of the prover. Otherwise, it may have obtained these by itself.[5] As we will see, leading ATP systems such as Vampire and E [59] can indeed be improved in this way. Different *methods* does not necessarily mean different *systems*: it is possible to use different configurations of the same system for lemma generation and proving, as well as for intermediate operations. This was the workflow used by Larry Wos to prove the challenge problem `LCL073-1` with OTTER [85]. Our SGCD system also supports this, which played a major role in its ability to prove the aforementioned challenge problem.

Lemmas play a quite different role for a family of provers which we call *CM-CT* for *Connection Method/Clausal Tableaux*, exemplified by PTTP [61], SETHEO [33], and leanCoP [47,46]. Underlying conceptual models are model elimination [35], clausal tableaux [31] and the CM. They enumerate proof struc-

[5] We note here that in some cases systems *cannot* generate certain lemmas because of e.g. ordering restrictions.

tures while propagating variable bindings initialized by the goal through unification, and hence proceed in an inherently goal-driven way. While they are good at problems that benefit from goal direction, in general they are much weaker than RS provers and have not been among the top provers at *CASC* for about two decades. This is attributed to the fact that they do not re-use the proof of one subgoal as the solution of another: they do not use lemmas *internally*.

The lack of lemmas was identified early as a weakness of CM-CT [14], so there have been various proposed remedies [14,2,62,60,32,16,46,19]. Despite some insight and success, this did not yet elevate CM-CT to the level of the best RS systems. Nevertheless, the expectation remains that CM-CT provers would benefit from supplying lemmas as additional input. Hence, we included two CM-CT systems in our experiments, leanCoP and CMProver [11,72,73] and show that the expectation is greatly confirmed. Two other systems considered here, SGCD and CCS [74], can be viewed as CM-CT systems extended to support specific forms of lemma generation and application.

Lemmas can be maintained within the prover as an inherent part of the method, as in saturation. They may also be created and applied by different systems, or different instances of the same system [12,55]. Larry Wos calls this *lemma adjunction* [84]. Lemmas created by one system are passed to a second system in two principal ways. First, they can be passed as *additional axioms*, in the hope that the second system finds a shorter proof in the wider but shallower search space. Second, external lemmas can be used to *replace search*. The second system then starts with the given lemmas as if they were the cached result of its previous computation. Moreover, the provided lemmas can be restricted in advance by heuristic methods, such as by a machine-learned model. SGCD supports this *replacing* lemma incorporation. The basic distinction between augmenting and replacing search with lemmas was already observed by Owen L. Astrachan and Mark E. Stickel [2] in the context of improving CM-CT provers.

2.1 Machine Learning for ATP

The past decade has seen numerous attempts to leverage machine learning in the automated theorem proving effort. Early systems mostly focused on premise selection, e.g. [68,1,71], aiming to reduce the number of axioms supplied as input to the prover. Other works provide internal guidance directly at the level of inferences during search, e.g. [34,25,17,27,86,54]. The emergence of generative language models has also led to some initial attempts at directly generating next proof steps, e.g. [49,67,50], moving the emphasis away from search.

In contrast to these lines of work, our focus is on learning the utility of lemmas. Close to our aims is [26,28], trying to identify globally useful lemmas in a collection of millions of proofs in HOL Light. Besides differences in the formal system, what distinguishes our work is that we learn a much more focused model: we put great emphasis on evaluating lemmas in the context of a particular goal and axiom set; in fact, our entire system was designed around the question whether a given lemma is moving the goal closer to the axioms. We argue that the D-term representation of all involved components (goal, lemma, axioms, proof) makes our framework particularly suitable for the lemma selection task.

We employ an iterative improvement approach first used in MaLARea [68]: in each iteration, we run proof search guided by a learned model, extract training data from proving attempts, and fit a new model to the new data. These steps can be repeated profitably until performance saturates.

2.2 Condensed Detachment: Proofs as Terms

Condensed detachment (CD) was developed in the mid-1950s by Carew A. Meredith as an evolution of *substitution and detachment* [51,30,52,43]. Reasoning steps are by *detachment*, or modus ponens, under implicit substitution by most general unifiers. Its primary application is the investigation of axiomatizations of propositional logics at a first-order meta-level. CD also provides a technical approach to the Curry-Howard correspondence, "formulas as types" [22,21] and is considered in witness theory [57]. Many early successes in ATP were on CD problems [40,66], but success was also found in the reverse direction. Refinements of the OTTER prover in the 1990s, some of which have found their ways into modern RS provers, were originally conceived and explored in the setting of CD [80,81,40,82,83,69,15,85].

From a first-order ATP perspective, a CD problem consists of *axioms*, i.e. positive unit clauses; a *goal theorem*, i.e. a single negative ground unit clause representing a universally-quantified atomic goal theorem after Skolemization; and the following ternary Horn clause that models detachment.

$$\mathit{Det} \overset{\text{def}}{=} \mathsf{P}(\mathsf{i}(x, y)) \land \mathsf{P}(x) \rightarrow \mathsf{P}(y).$$

The premises of *Det* are called the *major* and *minor* premise, respectively. All atoms in the problem have the same predicate P, which is unary and stands for something like *provable*. The formulas of the investigated propositional logic are expressed as terms, where the binary function symbol i stands for *implies*.

CD may be seen as an *inference rule*. From an ATP perspective, a *CD inference step* can be described as a hyperresolution from *Det* and two positive unit clauses to a third positive unit clause. A *CD proof* is a proof of a CD problem constructed with the CD inference rule. CD proofs can be contrasted with other types of proof, such as a proof with binary resolution steps yielding non-unit clauses. Prover9 [38] chooses positive hyperresolution by default as its only inference rule for CD problems and thus produces CD proofs for these.

It is, however, another aspect of CD that makes it of particular interest for developing new ATP methods, which only recently came to our attention in the ATP context [76]: the structure of CD proofs can be represented in a very simple and convenient way as full binary trees, or as terms. In ATP we find this aspect in the CM, where the proof structure as a whole is in focus, in contrast to extending a set of formulas by deduction [9]. This view of CD is made precise and elaborated upon in [77], on which the subsequent informal presentation is based. We call the structure representations of CD proofs *D-terms.* A D-term is a term recursively built from numeral constants and the binary function symbol D whose arguments are D-terms. In other words, it is a full binary tree where the leaf nodes are labeled with constants. Four examples of D-terms are

$$1, \quad 2, \quad \mathsf{D}(1, 1), \quad \mathsf{D}(\mathsf{D}(2, 1), \mathsf{D}(1, \mathsf{D}(2, 1))).$$

A D-term represents the structure of a proof. A proof in full is represented by a D-term together with a mapping of constant D-terms to axioms. Conversion between CD proofs and D-terms is straightforward: the use of an axiom corresponds to a constant D-term, while an inference step corresponds to a D-term $\mathsf{D}(d_1, d_2)$ where d_1 is the D-term that proves the major premise and d_2 the minor.

Through first-order unification, constrained by axioms for the leaf nodes and the requirements of *Det* for inner nodes, it is possible to obtain a most general formula proven by a D-term [77]. We call it the *most general theorem* (MGT) of the D-term with respect to the axioms, unique up to renaming of variables. For a given axiom map, not all D-terms necessarily have an MGT: if unification fails, we say the D-term has no MGT. It is also possible that different D-terms have the same MGT, or that the MGT of one is subsumed by the MGT of another. A D-term is a proof of the problem if its MGT subsumes the goal theorem.

As an example, let the constant D-term 1 be mapped to $\mathsf{P}(\mathsf{i}(x, \mathsf{i}(x, x)))$, known as *Mingle* [66]. Then, the MGT of the D-term 1 is just this axiom. The MGT of the D-term $\mathsf{D}(1, 1)$ is $\mathsf{P}(\mathsf{i}(x, \mathsf{i}(x, x)), \mathsf{i}(x, \mathsf{i}(x, x)))$, that is, after renaming of variables, $\mathsf{P}(y)\sigma$ where σ is the most general unifier of the set of pairs $\{\{\mathsf{P}(\mathsf{i}(x, y)),\ \mathsf{P}(\mathsf{i}(x', \mathsf{i}(x', x')))\},\ \{\mathsf{P}(x),\ \mathsf{P}(\mathsf{i}(x'', \mathsf{i}(x'', x'')))\}\}$.

D-terms, as full binary trees, facilitate characterizing and investigating structural properties of proofs. While, for a variety of reasons, it is far from obvious how to measure the size of proofs obtained from ATP systems in general, for D-terms there are at least three straightforward size measures:

– The *tree size* of a D-term is the number of its inner nodes.
– The *height* of a D-term is the length of the longest root-leaf path.
– The *compacted size* of a D-term is the number of distinct compound subterms, or, in other words, the number of inner nodes of its minimal DAG.

Alternative names in the literature are *length* for compacted size, *level* for height and *CDcount* [69] for tree size. The D-term $\mathsf{D}(\mathsf{D}(1, \mathsf{D}(1, 1)), \mathsf{D}(\mathsf{D}(1, 1), 1))$, for example, has tree size 5, compacted size 4 and height 3. *Factor equations* provide a compact way of writing D-terms: distinct subproofs with multiple incoming edges in the DAG receive numeric labels, by which they are referenced. The D-term $\mathsf{D}(\mathsf{D}(1, 1), \mathsf{D}(\mathsf{D}(1, \mathsf{D}(1, 1)), \mathsf{D}(1, \mathsf{D}(1, 1))))$, for example, can be written as $2 = \mathsf{D}(1, 1),\ 3 = \mathsf{D}(1, 2),\ 4 = \mathsf{D}(2, \mathsf{D}(3, 3))$.

CD problems have core characteristics of first-order ATP problems: first-order variables, at least one binary function symbol and cyclic predicate dependency. But they are restricted: positive unit clauses, one negative ground clause, and one ternary Horn clause. Equality is not explicitly considered. The generalization of CD to arbitrary Horn problems is, however, not difficult [74].

2.3 Condensed Detachment for ATP and Lemmas

From an ATP point of view, D-terms provide access to proofs as a whole. This exposes properties of proofs that are not merely local to an inference step, but spread across the whole proof. It suggests a shift in the role of the calculus from providing a recipe for building the structure towards an inductive structure *specification.* Moreover, D-terms as objects provide insight into *all* proofs: for example, growth rates of the number of binary trees for tree size, height and compacted size are well-known with entries in *The On-Line Encyclopedia of Integer Sequences* [45] and provide upper bounds for the number of proofs [77]. A practical consequence for ATP is the justification of proof structure enumeration techniques where each structure appears at most once.

CD proofs suggest and allow for a specific form of lemmas, which we call *unit* or *subtree* lemmas, reflecting two views on them. As formulas, they are positive unit clauses, which can be re-used in different CD inference steps. In the structural view, they are subterms, or subtrees, of the overall D-term. If they occur multiply there, they are factored in the minimal DAG of the overall D-term. The views are linked in that the formula of a lemma is the MGT of its D-term. The *compacted size* measure specified above takes into account the compression achievable by unit/subtree lemmas. From the perspective of proof structure compression methods, unit/subtree lemmas have the property that the compression target is unique, because each tree is represented by a unique minimal DAG. CM-CT provers do not support such lemmas, which is the main reason for their notorious weakness on CD problems.

2.4 SGCD — Structure Generating Theorem Proving

SGCD *(Structure Generating Theorem Proving for Condensed Detachment)* [75] is the central system used in our experiments as prover as well as lemma generator. It realizes an approach to first-order theorem proving combining techniques known from the CM and RS that was not fully recognized before. It generalizes (for CD problems) bottom-up preprocessing for and with CM-CT provers [60] and hypertableaux [4]. SGCD works by enumeration of proof structures together with unification of associated formulas, which is also the core method of the CM-CT provers. Structures for which unification fails are excluded. Each structure appears at most once in the enumeration.

Let the proof structures be D-terms. Partition the set of all D-terms according to some *level* such that those in a lower level are strict subterms of those in a higher level. Tree size or height are examples of such a level. Let

`enum_dterm_mgt_pairs(`*+Level*`, `*?DTerm*`, `*?Formula*`)`

be a Prolog[6] predicate enumerating D-terms and corresponding MGTs at a certain level, with respect to some quietly assumed axioms. We say that the predicate generates these pairs in an *axiom-driven* way. If the predicate is invoked with the formula argument instantiated by a ground formula, it enumerates D-terms that prove the formula at the specified level. The predicate is then used *goal-driven*, like a CM-CT prover. Invoking it for increasing level values realizes iterative deepening. There are further instantiation possibilities: if only the D-term is instantiated and the level is that of the D-term, its MGT is computed. If both D-term and formula are instantiated, the predicate acts as verifier.

The implementation includes several *generators*, concrete variants of the `enum_dterm_mgt_pairs` predicate for specific level characterizations. SGCD main-

[6] Prolog serves here as a suitable specification language.

```
$C := \emptyset$;
for $l := 0$ to $maxLevel$ do
    for $m := l$ to $l + preAddMaxLevel$ do
        enum_dterm_mgt_pairs($m, d, g$);
        throw proof_found($d$)
    $N := \{\langle l, d, f\rangle \mid$ enum_dterm_mgt_pairs($l, d, f$)$\}$;
    if $N = \emptyset$ then throw exhausted;
    $C :=$ merge_news_into_cache($N, C$)
```

Fig. 1. The nested loops of the SGCD theorem proving method.

tains a cache of $\langle level, D\text{-}term, formula\rangle$ triples used to obtain solutions for subproblems in levels below the calling level. This cache is highly configurable. In particular, the number of entries can be limited, where only the best triples according to specified criteria are kept. Typical criteria are height or size of the formula, a heuristic shared with RS provers. Subsumed entries can be deleted, another feature in common with RS. Novel criteria are also supported, some of which relate the formula to the goal. Most criteria are based on the formula component of the triples, the MGT. Due to rigid variables [23], MGTs are not usually available in CM-CT provers [77] and cannot be used as a basis for heuristics.

When lemmas are provided to SGCD, they are used to initialize the cache, replacing search at levels lower than the calling level.[7] SGCD further maintains a set of *abandoned* $\langle level, D\text{-}term, formula\rangle$ triples, those that are generated but do not qualify for entering the cache or were removed from the cache. These are kept as a source for heuristic evaluation of other triples and for lemma generation.

For theorem proving, SGCD proceeds as shown in Fig. 1. Input parameter g is the goal formula, while parameters *maxLevel* and *preAddMaxLevel* are configurable. `enum_dterm_mgt_pairs` represents a particular generator that is also configurable. It enumerates argument bindings nondeterministically: if it succeeds in the inner loop, an exception returns the D-term d. C is the cache. The procedure `merge_news_into_cache(`N, C`)` merges newly generated $\langle level, D\text{-}term, formula\rangle$ triples N into the cache C. If *maxLevel* is configured as 0, the method proceeds in purely goal-driven mode with the inner loop performing iterative deepening on the level m. Similarity to CM-CT provers can be shown empirically by comparing the sets of solved TPTP problems [75]. Generally successful configurations of *preAddMaxLevel* typically have values 0–3.

3 Improving a Prover via Learned Lemma Selection

We employ machine learning to identify lemmas that can enhance proof search. Unlike the standard supervised scenario in which we learn from some training problems and evaluate performance on separate test problems, we take a reinforcement learning approach of self-improvement that has already been successfully applied in several theorem proving projects since [68]. In this approach, we perform proof search with a *base prover* on our entire problem set and learn

[7] Replacement can be subject to heuristic restrictions.

from the proof attempts.[8] The learning-assisted prover is evaluated again in the problem set to see if it can find more or different problems. If there is improvement, the process can be repeated until performance saturates. In a bit more detail, our system has the following components.

1. **Base Prover**: Performs proof search and its main role is to provide training data to the utility model.
2. **Utility Model**: The model takes ⟨*conjecture*, *lemma*, *axioms*⟩ triples and outputs an utility score, i.e., some measure of how useful the lemma is for proving the conjecture from the axioms. The utility model is trained from the D-terms emitted by the base prover.
3. **Lemma Generator**: Produces a large set of candidate lemmas for each problem separately. All candidates are derivable from the axioms.
4. **Evaluated Prover**: For each problem, we evaluate the candidate sets with the utility model and select the best ones. These lemmas are provided to the evaluated prover which performs proof search on the problem set. The evaluated prover can be identical to or different from the base prover.

Base Prover. Any prover that emits proofs as D-terms is suitable as a base prover. Given a D-term proof tree P of some formula C from axiom set As, any connected subgraph S of P can be considered as the proof of a lemma L. If S is a full tree, it proves a unit lemma, which is the formula associated with its root. Otherwise, it proves a Horn clause, whose head is the root formula of S and whose body corresponds to the open leaves of S. We currently focus on unit lemmas and leave more general subgraphs for future work. To approximate the utility of lemma L for proving C from As, there are several easy-to-compute logical candidates, such as the reduction in tree size, tree height or compressed size. A more refined measure is obtained if we reprove C with the lemma L added to the axioms As and observe how the number of inference steps changes.[9] This is slower to compute, but takes into account the particularities of the base prover, hence provides more focused guidance. In our experiments, we find that the best performance is obtained by reproving and then computing utility U as the inference step reduction normalised into $[-1, 1]$, where -1 means that the problem could not be solved within the original inference limit and 1 is assigned to the lemma that yields the greatest speedup. We end up with tuples $\langle C, As, L, U\rangle$ to learn from.

Utility Model Training. We experiment with gradient-descent optimisation for two classes of functions: linear models and graph neural networks (GNNs). Our linear model is based on 51 manually-identified features, some of them novel, described in App. A. For each feature f_i there is an associated weight parameter w_i to produce the final predicted utility

$$U(\boldsymbol{f}; \boldsymbol{w}) = \sum_i f_i w_i$$

[8] We currently only learn from successful proof attempts and sketch an extension to learning from failure.

[9] The number of inferences is a measure provided by the Prolog engine and is not identical to the number of steps in the FOL calculus.

The second, more involved model is a GNN. Describing this model is beyond the scope of this paper: see e.g. [58] for a gentle introduction. What is crucial for our purposes is that no manual feature extraction is involved: a specialized neural network processes the D-terms of involved formulas directly and learns to extract useful features during optimisation. As input, the model is given a graph, losslessly encoding D-terms of the lemma to be evaluated, the conjecture and the axioms. The precise network architecture is provided in App. B.

Candidate Lemma Generation. Candidate lemmas are generated separately for each problem via the structure enumeration mechanism of SGCD, as explained in Fig. 1. The goal g is provided and *preAddMaxLevel* is set to 0, making SGCD proceed axiom-driven, generating lemmas level by level. However, it does intersperse the goal-driven inner loop, which is only trying to prove the goal on the level directly above the last cached level. SGCD may terminate with a proof, in which case further lemma generation is pointless. Otherwise it terminates after *maxLevel* is reached, generation of new levels is exhausted, or a time limit is reached. We then use the cache C and the abandoned triples as the generated output lemmas. Furthermore, there are many ways to configure SGCD. We obtained the best results generating by tree size and by PSP-level (explained below), combined with known good heuristic restrictions. In particular we restrict the size of the lemma formulas to the maximum of the size of the axioms and the goal, multiplied by some factor (usually 2–5). We also restrict the number of elements in the cache, typically to 1,000. The lemmas are sorted by formula size measures, smaller preferred, to determine which are retained in the cache.

Proof structure generation by PSP-level is a novel technique introduced in [75,77], based on an observation by Łukasiewicz and Meredith. In a detachment step, often the D-term that proves one premise is a subterm of the D-term that proves the other. We turn this relationship into a proof structure enumeration method: structures in level $n + 1$ are D-terms where one argument D-term is at level n and the other argument is a subterm of that D-term. The method is incomplete, but combines features of DAG enumeration while being compatible with a simple global lemma maintenance as realized with SGCD's cache [77].

Evaluated Prover. For each problem, we evaluate the candidate set with the utility model and select k lemmas with the highest predicted utility, where k is a hyperparameter. The evaluated prover then tries to solve the problems with the help of the selected lemmas. The lemmas can either be treated as additional axioms – applicable to any prover – or have a specialised treatment if the prover provides for it: in particular, SGCD and CCS-Vanilla use the lemmas to replace inner lemma enumeration.[10] The evaluated prover can be any prover, since there is no specialised requirement to handle lemmas as new axioms. If, however, it is the base prover – or any other system that emits proofs as D-terms, then the learning procedure can be iterated as long as there are new problems solved.

[10] Before the obtained input lemmas are passed to a prover we supplement them with the lemmas for all their subproofs, i.e. we close the set of D-terms under the subterm relationship. This proved beneficial in experiments (see, e.g., App. D). An alternative would be to perform this closure on all generated lemmas before selection.

Table 1. Features of the considered provers: whether their proofs are available as D-terms (possibly after some conversion), whether they were used with *replacing* lemma incorporation (Sect. 2), whether they operate goal-driven, and the underlying method.

	SGCD	Prover9	CMProver	leanCoP	CCS-Vanilla	Vampire	E
D-terms	•	•	•	–	•	–	–
Replacing lemmas	•	–	–	–	•	–	–
Goal-driven	•/–	–	•	•	•	–	–
CM-CT	–	–	•	•	–	–	–
RS	–	•	–	–	–	•	•

3.1 Learning-Based Experiments

We experiment with a total of 312 CD problems, including all 196 pure CD problems from TPTP 8.1.2 [64], enriched with single-axiom versions of all the problems to which a technique by Tarski [37], as specified by Rezuş [56], was applicable. We test several representative ATP systems, including state-of-the-art systems for both general first-order reasoning and for CD problems.

Table 1 gives an overview of the considered provers. CCS-Vanilla is CCS [74] in a restricted configuration to find only those CD proofs with minimal compacted size, identifying problems that can clearly be solved with exhaustive search. It operates goal-driven, like the CM-CT provers, but by enumerating DAGs instead of trees through a local lemma maintenance mechanism. Vampire and E represent the state of the art of first-order ATP. Provers that produce D-terms as proofs (SGCD, Prover9, CMProver, CCS) can serve as base provers. We always rely on SGCD for lemma candidate generation. All provers are recent public versions: Vampire 4.5.1, E 2.6, leanCoP 2.1. We provide results in terms of *time* limits, although for the Prolog provers SGCD, CMProver and CCS-Vanilla we used a roughly-equivalent inference limit to avoid fluctuations due to server workload.

Improving the Base Prover. In our first experiment, we evaluate base provers after learning from their own proof attempts. The provers are given $k = 200$ best lemmas according to the linear utility model. Table 2[11] shows problems solved by four base provers without lemmas (*Base* case) and with two iterations of learning. The *Total* row gives the number of theorems proved by any of the three iterations shown. The stronger the base model, the harder it is to improve. CMProver and CCS-Vanilla are purely goal-driven and benefit greatly, reaching over 37% improvement for larger time limits. SGCD and Prover9 improve over 5% for shorter time limits, but this effect gradually vanishes as the time limit is increased.

Table 2. Number of problems solved over 2 iterations of training a linear model.

	SGCD				Prover9				CMProver				CCS-Vanilla			
Time	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m
Base	266	275	285	285	240	252	259	262	82	85	94	103	81	88	99	105
Iter 1	280	282	284	281	250	254	262	257	83	93	105	121	96	101	117	130
Iter 2	281	283	281	283	247	247	267	265	79	98	95	126	96	97	120	128
Total	282	284	286	286	253	258	269	267	91	105	112	141	106	105	133	145

[11] Further visualisations of our experiments are provided in App.C.

An analysis, provided in App. D, reveals that in the proofs not found during lemma generation and found by SGCD after the provision of lemmas, $63 - 96\%$ of the distinct subterms originate from the lemmas, i.e., a substantial portion of the proofs are built up from the provided lemmas.

Learned Lemmas to Enhance other Provers. Next, we fix SGCD as base prover and evaluate other provers, namely Vampire, E, Prover9 and leanCoP. Again, the provers are given $k = 200$ best lemmas according to the linear utility model. Table 3 shows the greatest boost is for the purely goal-driven leanCoP, where there is over 40% improvement for all time limits. Second is Vampire with $8 - 15\%$ improvement, followed by Prover9 and E with around 3% improvement. Interestingly, E does not solve more problems with the lemmas, but it solves different ones, hence the improvement. These results suggest a great deal of transferability of the benefits of the lemma selector.

Table 3. Number of problems solved by Vampire (casc), E (autoschedule), Prover9 and leanCoP without and with additional lemmas using various time limits.

	Vampire				E				Prover9				leanCoP			
Time	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m	50s	100s	500s	30m
Base	221	224	252	263	253	264	275	281	236	244	257	260	70	71	77	77
Lemmas	249	257	274	283	256	266	275	275	246	250	261	269	100	103	111	113
Total	249	257	276	284	269	276	287	286	248	252	264	269	100	103	111	113

Changing the Number of Lemmas Added. Adding lemmas has potential to shorten proofs, but it also widens the search space, so it is not obvious how many lemmas are beneficial. In the next experiment, we again fix SGCD as base prover and evaluate SGCD and Vampire with different number of lemmas selected. Table 4 shows that as little as 25 added lemmas yield substantial improvement, 7% for Vampire and 4% for SGCD, and performance does not drop as we add more lemmas: even at 500 we see no negative effect of the expanded search space.

Table 4. Number of problems solved by Vampire (casc) and SGCD as we alter the number k of supplemented lemmas. We use a time limit of 100s.

	Vampire						SGCD					
Lemma count	10	25	50	100	200	500	10	25	50	100	200	500
Base	227	227	227	227	227	227	275	275	275	275	275	275
Lemmas	226	242	246	258	257	258	278	285	284	281	283	284
Total	231	243	247	258	257	258	282	285	284	283	284	285

Linear vs GNN Model. The preceding experiments suggest that even a simple linear model can provide useful guidance when features are carefully selected. Table 5 shows that the GNN — which processes the formulas directly and has no access to expert designed features — also successfully learns to identify useful lemmas for SGCD and even slightly surpasses the linear model. `LCL125-1` can only be solved by the GNN-assisted prover, even at extremely large time limits.

Table 5. Number of problems solved by SGCD over 2 iterations of training both a linear and a graph neural network model, for time limits 50 s, 100 s, 500 s and 30 min.

	Linear				GNN			
Time	50s	100s	500s	30m	50s	100s	500s	30m
Base	266	275	285	285	266	275	285	285
Iter 1	280	282	284	281	272	282	283	284
Iter 2	281	283	281	283	279	282	282	284
Total	282	284	286	286	279	285	287	287

3.2 Discussion of Learning-Based Experiments

When enhanced by learning-based lemma selection, SGCD solves 287 of the 312 problems. These include 28 problems not solved by the leading first-order prover Vampire [29], which solves 263 problems in its *CASC* [63] portfolio mode. Supplemented with our lemmas, Vampire is boosted to 284 solved problems. In combination, boosted SGCD and Vampire give 293 solved problems. Taking into account the solutions obtained by further provers with our lemmas, we obtain a total of 297. For detailed results see App. E and http://cs.christophwernhard.com/cdtools/exp-lemmas/lemmas.html.

A notable observation is that all systems – with the exception of E – improve when provided with selected lemmas. We argue that our framework addresses fundamental weaknesses of both purely goal-driven systems such as CMProver, leanCoP and CCS-Vanilla, as well as those of saturation style systems such as Vampire and E. For the former, it is their inability to generate lemmas, which results in unduly long proofs. For the latter, it is their unrestricted expansion of the branching of the search space. We find that goal-driven systems demonstrate huge improvement when lemmas are added: usually $20 - 40\%$ depending on the configuration. The improvement is much more modest for saturation style systems, partly because their baselines are already stronger and partly because learned lemma selection still has a large room for improvement. This is the focus of our immediate future work. SGCD already provides a balance between goal-driven search and axiom-driven lemma generation and we only see significnt improvement from lemmas when the time limit on proof search is smaller. Our manual feature-based linear model allows for exploiting expert knowledge. However, we see more potential in automated feature extraction via GNNs. The fact that the two models perform similarly suggests that we are not providing enough training data for the GNN to manifest its full capabilities.

4 Proving `LCL073-1`

`LCL073-1` was proven by Meredith in the early 1950s with substitution and detachment [42] but it remains outstandingly hard for ATP, where it came to attention in 1992 [40]; TPTP reports rating 1.0 and status *Unknown* since 1997. Only Wos proved it in the year 2000 with several invocations of OTTER [85],

$$\begin{aligned}&2 = \mathsf{D}(1, \mathsf{D}(1, \mathsf{D}(1, 1))),\ 3 = \mathsf{D}(2, 2),\ 4 = \mathsf{D}(1, 3),\ 5 = \mathsf{D}(1, 4),\ 6 = \mathsf{D}(5, 1),\ 7 = \mathsf{D}(5, 6),\\ &8 = \mathsf{D}(\mathsf{D}(\mathsf{D}(1, \mathsf{D}(1, 7)), 6), 1),\ 9 = \mathsf{D}(8, 6),\ 10 = \mathsf{D}(8, \mathsf{D}(1, 9)),\ 11 = \mathsf{D}(\mathsf{D}(1, \mathsf{D}(1, \mathsf{D}(4, 10))), 1),\\ &12 = \mathsf{D}(1, \mathsf{D}(6, \mathsf{D}(1, \mathsf{D}(\mathsf{D}(1, \mathsf{D}(9, \mathsf{D}(9, \mathsf{D}(\mathsf{D}(11, 3), 4)))), 1)))),\ 13 = \mathsf{D}(\mathsf{D}(\mathsf{D}(12, \mathsf{D}(5, \mathsf{D}(8, 12))), 1), 7),\\ &14 = \mathsf{D}(1, \mathsf{D}(13, \mathsf{D}(1, \mathsf{D}(13, 5)))),\ 15 = \mathsf{D}(\mathsf{D}(1, \mathsf{D}(13, \mathsf{D}(\mathsf{D}(\mathsf{D}(\mathsf{D}(13, 6), 9), 11), 10))), \mathsf{D}(14, \mathsf{D}(14, 1)))\end{aligned}$$

Fig. 2. The D-term of our proof of `LCL073-1` represented by factor equations.

transferring output and insight between runs. The problem has a single axiom,

$$\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{i}(x, y), \mathsf{i}(\mathsf{n}(z), \mathsf{n}(u))), z), v), \mathsf{i}(\mathsf{i}(v, x), \mathsf{i}(u, x)))),$$

and the goal $\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{a}, \mathsf{b}), \mathsf{i}(\mathsf{i}(\mathsf{b}, \mathsf{c}), \mathsf{i}(\mathsf{a}, \mathsf{c}))))$, known as *Syll* [66]. The wider context is showing that a single axiom entails the elements of a known axiomatization of a propositional logic. Experiments with SGCD in our workflow led to a proof of `LCL073-1` (Fig. 2, also App. F) surprisingly quickly. Its compacted size is 46, between that of Meredith (40, reconstructed with CD in [85]) and that of Wos (74). Our workflow is much simpler than Wos', basically the same as our other experiments but restricted to one phase of lemma generation and incorporation, with only heuristic lemma selection, no learning. Nevertheless, success is fragile with respect to configuration, where reasons for failure or success are not obvious.

Our configuration parameters are not problem specific, although we started out with lemma generation by PSP-level because it led earlier to a short proof of `LCL038-1` [75,77]. We first call SGCD to generate lemmas by PSP-level enumeration, configured with a cache size of 5,000, terminating after 60 s with exhaustion of the search space.[12] Lemma features are computed for the 98,198 generated lemmas and written to disk, taking another 120 s. Lemmas are then ordered lexicographically according to five features relating to sharing of symbols and subterms with the goal, and to formula dimensions, taking a further 70 s. These five features are `lf_h_height`, `lf_h_excluded_goal_subterms`, `lf_h_tsize`, `lf_h_distinct_vars`, `dcterm_hash`, see App. A for their specification. We now call SGCD again, configured such that it performs PSP-level enumeration for axiom-driven phases, interleaved with level enumeration by height for goal-driven phases with 0 as *preAddMaxLevel*. It incorporates the first 2,900 ordered lemmas[13] as input by *replacement* (Sect. 2). The cache size limit is set to 1,500, a value used in other generally successful configurations. Formulas occuring as subformulas of an earlier-proven formula are excluded, a variation of the *organic* property [37,77]. The proof is then found in 20 s, total time elapsed about 270 s.

The D-term dimensions $\langle \mathit{compacted\ size}, \mathit{tree\ size}, \mathit{height} \rangle$ are $\langle 46, 3276, 40 \rangle$, compared to Meredith's $\langle 40, 6172, 30 \rangle$[14] and Wos' $\langle 74, 9207, 48 \rangle$. The maximal size (occurrences of non-constant function symbols) of a lemma formula (MGT of a subproof) in the proof is 19, the maximal height (tree height, disregarding the predicate symbol) 9, and the maximal number of variables 7. Of the 46 lemmas in the proof 12 are present in the 2,900 input lemmas. 35 of the 46 lemma formulas are weakly organic [77]. 4 of the 46 formulas involve double

[12] Notebook hardware, Intel® Core™ i7-1260P processor, 32 GB RAM.

[13] 2,900 is one of the fragile parameters. Depending on features chosen for ordering lemmas, there are ranges around 3,000 where the problem is solved.

[14] The *length* reported in [85] is the compacted size if also the proofs of the two other goals required to prove completeness of the single axiom are considered. The notion of compacted size straightforwardly generalizes from trees to *sets* of trees [77].

negation. N-simplification [77] applies to 65 occurrences but does not effect a size reduction. The proof is S- and C-regular [77]. Certain configurations of SGCD for the proving phase also yield further proofs. In experiments so far, these are enumerated after the presented proof and have larger compacted size.

Proof structure enumeration by PSP-level [77] is the main key to finding our proof of LCL073-1. It is used for lemma generation and for axiom-driven proof search, whereas goal-driven phases use height instead. The structure of the proof reflects this: all steps with the exception of the root can be considered PSP steps, i.e. one premise is a subproof of the other. The particular challenge of the problem lies in the fact that it was solved by a human (Meredith). Unlike in recent ATP successes for Boolos' curious inference [10,5], where the key is two particular second-order lemmas, the key here is a proof-structural *principle* for building-up proofs by lemmas. Intuitively it might express a form of economy, building proofs from proofs at hand, that belonged to Meredith's repertoire.

5 Conclusion

We presented encouraging results about the use of lemmas in proof search. Provers are provided with lemmas generated via structure enumeration, a feature of the CM, and filtered with either learned guidance or manual heuristics. As a first step with this new methodology, we focus on the class of CD problems where we obtained strong results with our own system and substantial improvement of general first-order provers based on different paradigms, including the long-time competition leader Vampire. Moreover, our approach has led to the — in a sense first — automatic proof for the well-known Meredith single axiom problem with TPTP difficulty rating 1.0.

An important and novel aspect in our work was the explicit consideration of proof structures, which for CD have a particularly simple form in D-terms. Proof structures of the CM have a direct correspondence to these [77], such that the CM may guide the way to generalizations for more expressive logics. Another course of generalization is to move from unit lemmas, i.e. sharing of *subtrees* of D-terms, to more powerful lemmas. Preliminary work shows a correspondence between Horn clause lemmas, D-terms with variables, proofs in the connection structure calculus [14], and combinatory compression [74].

The learning-based experiments show little difference in performance between using a simple linear model and a more sophisticated graph neural network. We believe this is due to the small problem corpus, which yields a limited training signal. Hence, we plan to scale the system up to larger problem sets.

Our work also sheds new light on perspectives for the CM. It is well-known that the lack of inherent lemma maintenance is a disadvantage of the CM compared to resolution, which can be overcome with the connection structure calculus [14], a generalization of the CM. Here we see in experiments a drastic improvement of the CM-CT provers by supplementing their input with externally generated lemmas. SGCD, which grew out of the CM-CT approach and integrates repeated lemma generation into the proving process, keeps up with RS provers on CD problems, and can even be applied to improve these by supplying its lemmas as additional input.

References

1. Alemi, A.A., Chollet, F., Een, N., Irving, G., Szegedy, C., Urban, J.: DeepMath — Deep Sequence Models for Premise Selection. In: Lee, D., et al. (eds.) NIPS'16. pp. 2243–2251. Curran Associates Inc., USA (2016), http://dl.acm.org/citation.cfm?id=3157096.3157347
2. Astrachan, O.L., Stickel, M.E.: Caching and lemmaizing in model elimination theorem provers. In: Kapur, D. (ed.) CADE-11. pp. 224–238. Springer, Berlin (1992). https://doi.org/10.1007/3-540-55602-8_168
3. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handb. of Autom. Reasoning, vol. 1, chap. 2, pp. 19–99. Elsevier (2001). https://doi.org/10.1016/B978-044450813-3/50004-7
4. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: Alferes, J.J., Pereira, L.M., Orlowska, E. (eds.) JELIA'96. LNCS (LNAI), vol. 1126, pp. 1–17. Springer (1996). https://doi.org/10.1007/3-540-61630-6_1
5. Benzmüller, C., Fuenmayor, D., Steen, A., Sutcliffe, G.: Who finds the short proof? Logic Journal of the IGPL (2023). https://doi.org/10.1093/jigpal/jzac082
6. Bibel, W.: Automated Theorem Proving. Vieweg, Braunschweig (1982). https://doi.org/10.1007/978-3-322-90102-6, second edition 1987
7. Bibel, W.: Deduction: Automated Logic. Academic Press, London (1993)
8. Bibel, W., Otten, J.: From Schütte's formal systems to modern automated deduction. In: Kahle, R., Rathjen, M. (eds.) The Legacy of Kurt Schütte, chap. 13, pp. 215–249. Springer (2020). https://doi.org/10.1007/978-3-030-49424-7_13
9. Bonacina, M.P.: A taxonomy of theorem-proving strategies. In: Artificial Intelligence Today: Recent Trends and Developments, pp. 43–84. Springer (2001)
10. Boolos, G.: A curious inference. J. Philos. Logic **16**, 1–12 (1987). https://doi.org/10.1007/BF00250612
11. Dahn, I., Wernhard, C.: First order proof problems extracted from an article in the Mizar mathematical library. In: Bonacina, M.P., Furbach, U. (eds.) FTP'97. pp. 58–62. RISC-Linz Report Series No. 97–50, Joh. Kepler Univ., Linz (1997), https://www.logic.at/ftp97/papers/dahn.pdf
12. Denzinger, J., Kronenburg, M., Schulz, S.: DISCOUNT — a distributed and learning equational prover. J. Autom. Reasoning **18**(2), 189 (1997)
13. Ebner, G., Hetzl, S., Leitsch, A., Reis, G., Weller, D.: On the generation of quantified lemmas. J. Autom. Reasoning **63**(1), 95–126 (2019). https://doi.org/10.1007/s10817-018-9462-8
14. Eder, E.: A comparison of the resolution calculus and the connection method, and a new calculus generalizing both methods. In: Börger, E., Kleine Büning, H., Richter, M.M. (eds.) CSL '88. LNCS, vol. 385, pp. 80–98. Springer (1989). https://doi.org/10.1007/BFb0026296
15. Fitelson, B., Wos, L.: Missing proofs found. J. Autom. Reasoning **27**(2), 201–225 (2001). https://doi.org/10.1023/A:1010695827789
16. Fuchs, M.: Lemma generation for model elimination by combining top-down and bottom-up inference. In: Dean, T. (ed.) IJCAI 1999. pp. 4–9. Morgan Kaufmann (1999), http://ijcai.org/Proceedings/99-1/Papers/001.pdf
17. Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., Norrish, M.: Learning to prove with tactics. CoRR **abs/1804.00596** (2018), http://arxiv.org/abs/1804.00596
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR 2016. pp. 770–778 (2016)

19. Hester, J.: Novel Methods for First Order Automated Theorem Proving. Ph.D. thesis, University of Florida (2021)
20. Hetzl, S., Leitsch, A., Weller, D.: Towards algorithmic cut-introduction. In: Bjørner, N., Voronkov, A. (eds.) LPAR 2012. LNCS, vol. 7180, pp. 228–242. Springer (2012). https://doi.org/10.1007/978-3-642-28717-6_19
21. Hindley, J.R.: Basic Simple Type Theory. Cambridge University Press (1997). https://doi.org/10.1017/CBO9780511608865
22. Hindley, J.R., Meredith, D.: Principal type-schemes and condensed detachment. Journal of Symbolic Logic **55**(1), 90–105 (1990). https://doi.org/10.2307/2274956
23. Hähnle, R.: Tableaux and related methods. In: Robinson, A., Voronkov, A. (eds.) Handb. of Autom. Reasoning, vol. 1, chap. 3, pp. 101–178. Elsevier (2001). https://doi.org/10.1016/b978-044450813-3/50005-9
24. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML 2015. pp. 448–456. Proceedings of Machine Learning Research (2015)
25. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS, vol. 10383, pp. 292–302. Springer (2017). https://doi.org/10.1007/978-3-319-62075-6_20, https://doi.org/10.1007/978-3-319-62075-6_20
26. Kaliszyk, C., Urban, J.: Learning-assisted theorem proving with millions of lemmas. Journal of Symbolic Computation **69**, 109–128 (2015). https://doi.org/https://doi.org/10.1016/j.jsc.2014.09.032, https://www.sciencedirect.com/science/article/pii/S074771711400100X, symbolic Computation in Software Science
27. Kaliszyk, C., Urban, J., Michalewski, H., Olšák, M.: Reinforcement learning of theorem proving. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) NeurIPS 2018. pp. 8836–8847 (2018), https://papers.nips.cc/paper/2018/file/55acf8539596d25624059980986aaa78-Paper.pdf
28. Kaliszyk, C., Urban, J., Vyskočil, J.: Lemmatization for stronger reasoning in large theories. In: Lutz, C., Ranise, S. (eds.) Frontiers of Combining Systems - 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings. LNCS, vol. 9322, pp. 341–356. Springer (2015). https://doi.org/10.1007/978-3-319-24246-0_21, https://doi.org/10.1007/978-3-319-24246-0_21
29. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25. pp. 1–35. Springer (2013)
30. Lemmon, E.J., Meredith, C.A., Meredith, D., Prior, A.N., Thomas, I.: Calculi of pure strict implication. In: Davis, J.W., Hockney, D.J., Wilson, W.K. (eds.) Philosophical Logic, pp. 215–250. Springer Netherlands, Dordrecht (1969). https://doi.org/10.1007/978-94-010-9614-0_17, reprint of a technical report, Canterbury University College, Christchurch, 1957
31. Letz, R.: Tableau and Connection Calculi. Structure, Complexity, Implementation. Habilitationsschrift, TU München (1999), available from http://www2.tcs.ifi.lmu.de/~letz/habil.ps, accessed Jun 30, 2022
32. Letz, R., Mayr, K., Goller, C.: Controlled integration of the cut rule into connection tableaux calculi. J. Autom. Reasoning **13**(3), 297–337 (1994)
33. Letz, R., Schumann, J., Bayerl, S., Bibel, W.: SETHEO: A high-performance theorem prover. J. Autom. Reasoning **8**(2), 183–212 (1992). https://doi.org/10.1007/BF00244282
34. Loos, S.M., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. In: Eiter, T., Sands, D. (eds.) LPAR-21. EPiC, vol. 56, pp. 85–105 (2017). https://doi.org/10.29007/8mwc

35. Loveland, D.W.: Automated Theorem Proving: A Logical Basis. North-Holland, Amsterdam (1978)
36. Łukasiewicz, J.: Selected Works. North Holland (1970), edited by L. Borkowski
37. Łukasiewicz, J., Tarski, A.: Untersuchungen über den Aussagenkalkül. Comptes rendus des séances de la Soc. d. Sciences et d. Lettres de Varsovie **23** (1930), English translation in [36], p. 131–152
38. McCune, W.: Prover9 and Mace4 (2005–2010), http://www.cs.unm.edu/~mccune/prover9
39. McCune, W.: OTTER 3.3 Reference Manual. Tech. Rep. ANL/MCS-TM-263, Argonne National Laboratory (2003), https://www.cs.unm.edu/~mccune/otter/Otter33.pdf, accessed Jun 30, 2022
40. McCune, W., Wos, L.: Experiments in automated deduction with condensed detachment. In: Kapur, D. (ed.) CADE-11. LNCS (LNAI), vol. 607, pp. 209–223. Springer (1992). https://doi.org/10.1007/3-540-55602-8_167
41. Meredith, C.A., Prior, A.N.: Notes on the axiomatics of the propositional calculus. Notre Dame J. of Formal Logic **4**(3), 171–187 (1963). https://doi.org/10.1305/ndjfl/1093957574
42. Meredith, C.A.: Single axioms for the systems (C, N), (C, O) and (A, N) of the two-valued propositional calculus. J. Computing Systems **1**, 155–164 (1953)
43. Meredith, D.: In memoriam: Carew Arthur Meredith (1904–1976). Notre Dame J. of Formal Logic **18**(4), 513–516 (Oct 1977). https://doi.org/10.1305/ndjfl/1093888116
44. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: ICML 2010. pp. 807–814 (2010)
45. OEIS Foundation Inc.: The On-Line Encyclopedia of Integer Sequences (2021), http://oeis.org
46. Otten, J.: Restricting backtracking in connection calculi. AI Communications **23**(2-3), 159–182 (2010). https://doi.org/10.3233/AIC-2010-0464
47. Otten, J., Bibel, W.: leanCoP: lean connection-based theorem proving. J. Symb. Comput. **36**(1-2), 139–161 (2003). https://doi.org/10.1016/S0747-7171(03)00037-3
48. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
49. Piotrowski, B., Urban, J.: Guiding inferences in connection tableau by recurrent neural networks. In: Benzmüller, C., Miller, B.R. (eds.) CICM. LNCS, vol. 12236, pp. 309–314. Springer (2020). https://doi.org/10.1007/978-3-030-53518-6_23, https://doi.org/10.1007/978-3-030-53518-6_23
50. Polu, S., Sutskever, I.: Generative language modeling for automated theorem proving. CoRR **abs/2009.03393** (2020), https://arxiv.org/abs/2009.03393
51. Prior, A.N.: Logicians at play; or Syll, Simp and Hilbert. Australasian Journal of Philosophy **34**(3), 182–192 (1956). https://doi.org/10.1080/00048405685200181
52. Prior, A.N.: Formal Logic. Clarendon Press, Oxford, 2nd edn. (1962). https://doi.org/10.1093/acprof:oso/9780198241560.001.0001
53. Pudlák, P.: Search for faster and shorter proofs using machine generated lemmas. In: Sutcliffe, G., Schmidt, R., Schulz, S. (eds.) ESCoR 2006. CEUR Workshop

Proc., vol. 192, pp. 34–53. CEUR-WS.org (2006), http://ceur-ws.org/Vol-192/paper03.pdf

54. Rawson, M., Reger, G.: lazyCoP: Lazy paramodulation meets neurally guided search. In: Das, A., Negri, S. (eds.) TABLEAUX 2021. LNCS (LNAI), vol. 12842, pp. 187–199. Springer (2021). https://doi.org/10.1007/978-3-030-86059-2_11
55. Reger, G., Tishkovsky, D., Voronkov, A.: Cooperating proof attempts. In: CADE-25. pp. 339–355. Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_23
56. Rezuş, A.: Tarski's Claim thirty years later (2010). In: Witness Theory – Notes on λ-calculus and Logic [57], pp. 217–225, preprint (2016): http://www.equivalences.org/editions/proof-theory/ar-tc-20160512.pdf
57. Rezuş, A.: Witness Theory – Notes on λ-calculus and Logic, Studies in Logic, vol. 84. College Publications, London (2020)
58. Sanchez-Lengeling, B., Reif, E., Pearce, A., Wiltschko, A.B.: A gentle introduction to graph neural networks. Distill (2021). https://doi.org/10.23915/distill.00033, https://distill.pub/2021/gnn-intro
59. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) CADE 27. pp. 495–507. No. 11716 in LNAI, Springer (2019). https://doi.org/10.1007/978-3-030-29436-6_29
60. Schumann, J.M.P.: DELTA — A bottom-up preprocessor for top-down theorem provers. In: CADE-12. LNCS (LNAI), vol. 814, pp. 774–777. Springer (1994). https://doi.org/10.1007/3-540-58156-1_58
61. Stickel, M.E.: A Prolog technology theorem prover: implementation by an extended Prolog compiler. J. Autom. Reasoning **4**(4), 353–380 (1988). https://doi.org/10.1007/BF00297245
62. Stickel, M.E.: Upside-down meta-interpretation of the model elimination theorem-proving procedure for deduction and abduction. J. Autom. Reasoning **13**(2), 189–210 (1994). https://doi.org/10.1007/BF00881955
63. Sutcliffe, G.: The CADE ATP system competition — CASC. AI Magazine **37**(2), 99–101 (2016)
64. Sutcliffe, G.: The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0. J. Autom. Reasoning **59**(4), 483–502 (2017)
65. Sutcliffe, G., Gao, Y., Colton, S.: A grand challenge of theorem discovery. In: Worksh. Challenges and Novel Applications for Automated Reasoning, 19th IJ-CAR. pp. 1–11 (2003), online: https://www.cs.miami.edu/home/geoff/Papers/Conference/2003_SGC03_CNAAR-1-11.pdf
66. Ulrich, D.: A legacy recalled and a tradition continued. J. Autom. Reasoning **27**(2), 97–122 (2001). https://doi.org/10.1023/A:1010683508225
67. Urban, J., Jakubův, J.: First neural conjecturing datasets and experiments. In: Benzmüller, C., Miller, B.R. (eds.) CICM 2020. LNCS, vol. 12236, pp. 315–323. Springer (2020). https://doi.org/10.1007/978-3-030-53518-6_24, https://doi.org/10.1007/978-3-030-53518-6_24
68. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLARea SG1 – Machine Learner for Automated Reasoning with Semantic Guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 441–456. Springer (2008). https://doi.org/10.1007/978-3-540-71070-7_37
69. Veroff, R.: Finding shortest proofs: An application of linked inference rules. J. Autom. Reasoning **27**(2), 123–139 (2001). https://doi.org/10.1023/A:1010635625063
70. Walsh, M., Fitelson, B.: Answers to some open questions of Ulrich and Meredith (2021), under review, preprint: http://fitelson.org/walsh.pdf, accessed Jun 30, 2022

71. Wang, M., Tang, Y., Wang, J., Deng, J.: Premise selection for theorem proving by deep graph embedding. In: Guyon, I., et al. (eds.) NIPS 2017. pp. 2783–2793 (2017), http://papers.nips.cc/paper/6871-premise-selection-for-theorem-proving-by-deep-graph-embedding
72. Wernhard, C.: The PIE system for proving, interpolating and eliminating. In: Fontaine, P., Schulz, S., Urban, J. (eds.) PAAR 2016. CEUR Workshop Proc., vol. 1635, pp. 125–138. CEUR-WS.org (2016), http://ceur-ws.org/Vol-1635/paper-11.pdf
73. Wernhard, C.: Facets of the PIE environment for proving, interpolating and eliminating on the basis of first-order logic. In: Hofstedt, P., et al. (eds.) DECLARE 2019. LNCS (LNAI), vol. 12057, pp. 160–177 (2020). https://doi.org/10.1007/978-3-030-46714-2_11
74. Wernhard, C.: Generating compressed combinatory proof structures — an approach to automated first-order theorem proving. In: Konev, B., Schon, C., Steen, A. (eds.) PAAR 2022. CEUR Workshop Proc., vol. 3201. CEUR-WS.org (2022), https://arxiv.org/abs/2209.12592
75. Wernhard, C.: CD Tools — Condensed detachment and structure generating theorem proving (system description). CoRR **abs/2207.08453** (2023). https://doi.org/10.48550/ARXIV.2207.08453
76. Wernhard, C., Bibel, W.: Learning from Łukasiewicz and Meredith: Investigations into proof structures. In: Platzer, A., Sutcliffe, G. (eds.) CADE 28. LNCS (LNAI), vol. 12699, pp. 58–75. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_4
77. Wernhard, C., Bibel, W.: Investigations into proof structures. CoRR **abs/2304.12827** (2023). https://doi.org/10.48550/ARXIV.2304.12827, submitted
78. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming **12**(1-2), 67–96 (2012). https://doi.org/10.1017/S1471068411000494
79. Woltzenlogel Paleo, B.: Atomic cut introduction by resolution: Proof structuring and compression. In: Clarke, E.M., Voronkov, A. (eds.) LPAR-16. LNCS, vol. 6355, pp. 463–480. Springer (2010). https://doi.org/10.1007/978-3-642-17511-4_26
80. Wos, L., Winker, S., McCune, W., Overbeek, R., Lusk, E., Stevens, R., Butler, R.: Automated reasoning contributes to mathematics and logic. In: Stickel, M.E. (ed.) CADE-10. pp. 485–499. Springer (1990). https://doi.org/10.1007/3-540-52885-7_109
81. Wos, L.: Automated reasoning and Bledsoe's dream for the field. In: Boyer, R.S. (ed.) Automated Reasoning: Essays in Honor of Woody Bledsoe, pp. 297–345. Automated Reasoning Series, Kluwer Academic Publishers (1991). https://doi.org/10.1007/978-94-011-3488-0_15
82. Wos, L.: The resonance strategy. Computers Math. Applic. **29**(2), 133–178 (1995). https://doi.org/10.1016/0898-1221(94)00220-F
83. Wos, L.: The power of combining resonance with heat. J. Autom. Reasoning **17**(1), 23–81 (1996). https://doi.org/10.1007/BF00247668
84. Wos, L.: Lemma inclusion versus lemma adjunction. Association for Automated Reasoning Newsletter **44** (September 1999), online: https://aarinc.org/Newsletters/044-1999-09.html, accessed 24 Feb 2023
85. Wos, L.: Conquering the Meredith single axiom. J. Autom. Reasoning **27**(2), 175–199 (2001). https://doi.org/10.1023/A:1010691726881

86. Zombori, Z., Urban, J., Brown, C.E.: Prolog technology reinforcement learning prover (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS, vol. 12167, pp. 489–507. Springer (2020). https://doi.org/10.1007/978-3-030-51054-1_33, https://doi.org/10.1007/978-3-030-51054-1_33

A Manually Extracted Lemma Features

Below we specify the features that are manually extracted from lemmas and used our linear model (Sect. 3). We also provide their types as follows:

– *NatNum* A natural number.
– *NormalizedValue* A number between 0 and 1.

A.1 Features of the Lemma's D-term

`lf_d_csize` : *NatNum*
`lf_d_tsize` : *NatNum*
`lf_d_height` : *NatNum*
Compacted size, tree size and height of the lemma's D-term.

`lf_d_grd_csize` : *NatNum*
Compacted size of the lemma's D-term after replacing all variables with 0.

`lf_d_major_minor_relation` : *NatNum*
Describes the structural relationship of the subproofs of the major and minor premise of the lemma. Can have the following values: 0 identical or D-term is atomic; 1 is a strict superterm; 2 is a strict subterm; 3 none of these relationships; 4 for nonground D-terms.

`lf_d_number_of_terminals` : *NatNum*
Number of subterms in the lemma's D-term which are of the form $\mathtt{d}(d_1,d_2)$ where neither of d_1, d_2 is a compound term.

A.2 Context Dependent Features of the Lemma's D-Term

The features depend on the lemma as embedded in a given proof. They are only available for training data extracted from proofs.

`lfp_containing_proof` : `proof`
The proof that contains the lemma. Specified as the atom that identifies the proof. See also `lf_proof`.

`lfp_d_occs` : *NatNum*
If the lemma's D-term is ground, the number of occurrences of the lemma's D-term in the proof's D-term. If the lemma's D-term is non-ground, the number of subterm occurrences of the proof's D-term that are instances of the lemma's D-term (note that these subterm occurrences may overlap).

`lfp_d_incoming` : *NatNum*
The number of incoming edges of the lemma in the minimal DAG representing the proof's D-term. This can not be meaningfully determined for all lemma computation methods that yield s-lemmas.

`lfp_d_occs_innermost_matches` : *NatNum*
If the lemma's D-term is ground, the same as `lf_d_occs`. If the lemma's D-term has variables, the number of subterm occurrences of the proof's D-term that would be rewritten by INNERMOST replacement.

`lfp_d_occs_outermost_matches` : *NatNum*
: If the lemma's D-term is ground, the same as `lf_d_occs`. If the lemma's D-term has variables, the number of subterm occurrences of the proof's D-term that would be rewritten by OUTERMOST replacement.

`lfp_d_min_goal_dist` : *NatNum*
: Number of edges in the proof's D-term of the shortest downward path from the root to a subtree that is an instance of the lemma's D-term.

A.3 Special Features of the Lemma's Formula Components

`lf_b_length` : *NatNum*
: Length of the *Body* component of the lemma's formula.

`lf_hb_distinct_hb_shared_vars` : *NatNum*
: Number of distinct variables that occur in the *Head* as well as in the *Body* component of the lemma's formula.

`lf_hb_distinct_h_only_vars` : *NatNum*
: Number of distinct variables that occur in the *Head* but not the *Body* component of the lemma's formula.

`lf_hb_distinct_b_only_vars` : *NatNum*
: Number of distinct variables that occur in the *Body* but not the *Head* component of the lemma's formula.

`lf_hb_singletons` : *NatNum*
: Number of distinct variables with a single occurrence in *Head* and *Body* taken together.

`lf_hb_double_negation_occs` : *NatNum*
: Number of instances of `n(n(_))` in *Head* and *Body*.

`lf_hb_nongoal_symbol_occs` : *NatNum*
: Number of occurrences of symbols (functions, constants) in *Head* and *Body* that do not appear in the problem's goal. Context-dependent in that it refers to the problem's goal formula.

`lf_h_excluded_goal_subterms` : *NatNum*
: Number of distinct subterms of the goal formula (after replacing constants systematicall by variables) that are not a subterm of *Head* (modulo the variant relationship). Context-dependent in that it refers to the problem's goal formula.

`lf_h_subterms_not_in_goal` : *NatNum*
: Number of distinct subterms of the head that are not a subterm of the goal formula (after replacing constants systematicall by variables, modulo the variant relationship). Context-dependent in that it refers to the problem's goal formula.

`lf_hb_compression_ratio_raw_deflate` : *NormalizedValue*
`lf_hb_compression_ratio_treerepair` : *NormalizedValue*
`lf_hb_compression_ratio_dag` : *NormalizedValue*

Indicates how much the lemma's formula can be compressed. The value is roughly compressed size divided by original tree size. That is, formulas with "much regularity" such that they can be compressed stronger receive smaller values. The different properties realize this in variants for different notions and implementations of compression. The `raw_deflate` version depends on intrinsics of SWI-Prolog's term representation and possibly gives different results for the same formula, depending on how it internally shares subterms.

`lf_hb_organic` : *NatNum*
: Whether the formula is organic. A nonenmpty *Body* is translated to an implication, e.g., if *Body* = [a,b,c], the considered formula is i(a,i(b,i(c,*Head*))). Determined with Minisat. Values: 0: the formula is organic; 1 the formula is not organic but weakly organic; 2 the formula is not weakly organic. See also http://cs.christophwernhard.com/cdtools/downloads/cdtools/pldoc/organic_cd.html.

`lf_hb_name` : *Atom*
: A name of the formula if it is well known under some name. For a formula with nonempty body the translation to implication is considered (as for `lf_hb_organic`) and the name is prefixed with `meta_`. If the formula is not known under some name, the value is `zzz`. See also `named_axiom`/2 in http://cs.christophwernhard.com/cdtools/downloads/cdtools/pldoc/named_axioms_cd.html.

`lf_hb_name_status` : *NatNum*
: Number indicating whether the formula has a name in the sense of `lf_hb_name`: 0 if it has a name and an empty body, 1 if it has a nonempty body and a name (prefixed with `meta_`), 2 otherwise.

A.4 General Features of the Lemma's Formula Components

These features are specified below schematically with *COMP* for `h`, `b`, and `hb`, referring the respective features for the *Head* component, the *Body* component and both of the components joined together. The schema parameter *ITEM* for `var`, `const` and `fun` refers to variables, constants (atomic values in Prolog syntax) and function symbols with arity ≥ 1, respectively.

`lf_`*COMP*`_csize` : *NatNum*
`lf_`*COMP*`_tsize` : *NatNum*
`lf_`*COMP*`_height` : *NatNum*
: Compacted size, tree size and height, respectively. (We use these notions, which are also used for D-terms, here for formula terms.)

`lf_`*COMP*`_distinct_vars` : *NatNum*
: Number of distinct variables.

`lf_`*COMP*`_`*ITEM*`_occs` : *NatNum*
: Number of occurrences of syntactic objects of kind *ITEM*.

`lf_`*COMP*`_occs_of_most_frequent_`*ITEM* : *NatNum*
: Maximum number of occurrences of a syntactic object of kind *ITEM*.

B Implementation Details of the Network Architecture

This appendix supplements Sect. 3 with details of the network architecture and hardware setup.

Graph Neural Network Architecture

We use a graph neural network with 8 convolution layers of 128 channels arranged into 4 residual blocks [18], followed by a hidden dense layer of 1024 neurons and a final dense layer that produces a single utility output. Batch normalization [24] is applied before each convolutional layer, and the non-linearity throughout is a rectified linear unit [44]. The precise configuration was found by manual optimisation with respect to a holdout set.

Computation Used in the Experiments

We used a single NVIDIA A100 GPU and 50 CPU cores (100 hyperthreads). Approximate times for a typical experiment are: lemma extraction 20 min, a single iteration of proof search 5 min, model training 50 min, lemma selection 60 min. So far, we have run 151 experiments.

C Bar Chart Presentations of Selected Results

For convenience of the reader, we provide the following bar chart representations of selected data from Tables 2, 3 and 4 in Sect. 3.

Table 6. Performance of different provers over 2 iterations of training a linear model for 30 m time limit. Data from Table 2.

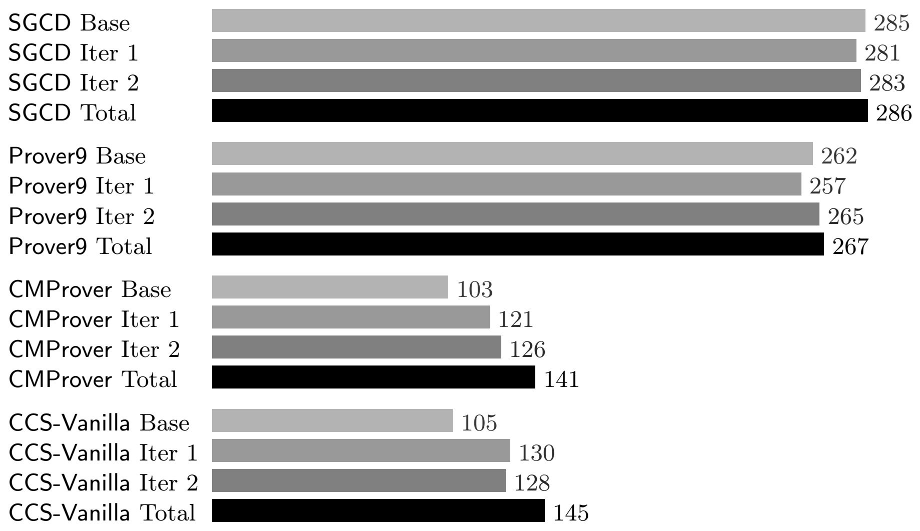

Table 7. Number of problems solved by Vampire (casc), E (autoschedule), Prover9 and leanCoP without and with additional lemmas for 30 m time limit. Data from Table 3.

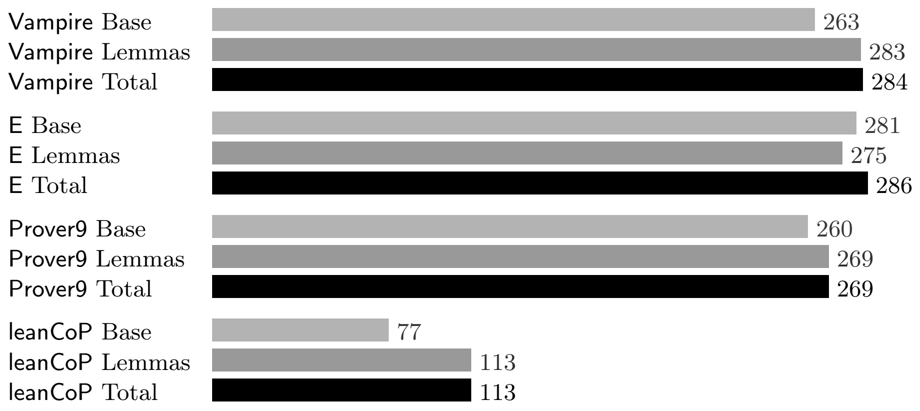

Table 8. Number of problems solved by Vampire (casc) and SGCD as we alter the number of supplemented lemmas between 10 and 500. We use a time limit of 100 s. Data from Table 4.

Prover	#Lemmas	#Solved problems
Vampire	Base	227
Vampire	10	226
Vampire	25	242
Vampire	50	246
Vampire	100	258
Vampire	200	257
Vampire	500	258
SGCD	Base	275
SGCD	10	278
SGCD	25	285
SGCD	50	284
SGCD	100	281
SGCD	200	283
SGCD	500	284

D Lemma Usage

Table 9 shows for the experiments described in Sect. 3 typical data on the usage of supplied input lemmas, indicating how much proofs actually draw from the supplied lemmas. The problems are either from the TPTP or, indicated by the *-tc* postfix in the problem name (suggesting *Tarski's Claim* [56]), single-axiom problems derived from multi-axiom TPTP problems as described in in Sect. 3.1. The data are from the first iteration of an experiment with SGCD, using a GNN without manual features and 30 min time limit. Of the 312 problems, 60 ones where proven with lemmas. The remaining problems were either proven already with lemma generation (224 problems) or not proven (28 problems). The number of selected input lemmas was 200. Closure under the subterm relationship for their D-terms led to 327–601 lemmas, median 418 (column **A**). We refer to this set as the *subproof-closed lemmas*. The compacted size of the 60 proofs was between 12 and 119, median 41.5 (column **B**). The ratio of compacted size, that is, the cardinality of the set of distinct non-atomic subproofs to the cardinality of the intersection of this set with the subproof-closed lemmas (column **C**) was between 0.63 and 0.96, median 0.87 (column **C**). This means that substantial portions of the proofs are actually built-up from the supplied lemmas. If the subproof closure is not considered, figures are quite different. The ratio with respect to the intersection with the original set of 200 lemmas (column **E**) was then only between 0 and 0.48, median 0.14 (column **F**).

Table 9. Usage of Learned Lemmas in Proofs

Problem	A	B	C	D	E	F	Problem	A	B	C	D	E	F
LCL019-1	445	51	45	0.88	3	0.06	LCL028-1-tc	407	35	29	0.83	3	0.09
LCL032-1	436	108	75	0.69	10	0.09	LCL030-1-tc	389	23	20	0.87	1	0.04
LCL037-1	463	119	75	0.63	8	0.07	LCL031-1-tc	388	35	31	0.89	2	0.06
LCL038-1	529	54	51	0.94	5	0.09	LCL054-1-tc	390	46	40	0.87	7	0.15
LCL054-1	504	54	50	0.93	4	0.07	LCL058-1-tc	399	34	31	0.91	5	0.15
LCL058-1	518	31	28	0.90	0	0.00	LCL060-1-tc	389	36	29	0.81	13	0.36
LCL060-1	460	40	36	0.90	8	0.20	LCL061-1-tc	368	37	28	0.76	9	0.24
LCL061-1	498	42	37	0.88	2	0.05	LCL062-1-tc	384	37	28	0.76	10	0.27
LCL062-1	493	45	38	0.84	6	0.13	LCL084-2-tc	331	46	32	0.70	19	0.41
LCL074-1	411	51	42	0.82	1	0.02	LCL084-3-tc	327	59	38	0.64	28	0.47
LCL084-2	520	53	50	0.94	6	0.11	LCL114-1-tc	425	42	39	0.93	2	0.05
LCL084-3	524	57	55	0.96	5	0.09	LCL116-1-tc	429	42	37	0.88	2	0.05
LCL100-1	508	23	18	0.78	0	0.00	LCL373-1-tc	368	47	40	0.85	10	0.21
LCL103-1	499	14	13	0.93	1	0.07	LCL374-1-tc	378	34	29	0.85	5	0.15
LCL122-1	510	35	32	0.91	3	0.09	LCL375-1-tc	384	35	26	0.74	5	0.14
LCL127-1	601	32	29	0.91	6	0.19	LCL376-1-tc	380	39	33	0.85	4	0.10
LCL129-1	491	12	11	0.92	2	0.17	LCL377-1-tc	386	41	34	0.83	6	0.15
LCL167-1	366	48	45	0.94	12	0.25	LCL382-1-tc	375	28	25	0.89	3	0.11
LCL374-1	482	42	38	0.90	7	0.17	LCL383-1-tc	394	32	29	0.91	3	0.09
LCL375-1	491	43	37	0.86	6	0.14	LCL385-1-tc	398	37	31	0.84	5	0.14
LCL376-1	504	30	26	0.87	2	0.07	LCL388-1-tc	402	37	31	0.84	6	0.16
LCL377-1	508	48	42	0.88	5	0.10	LCL389-1-tc	407	43	37	0.86	7	0.16
LCL388-1	500	54	51	0.94	2	0.04	LCL390-1-tc	384	42	35	0.83	9	0.21
LCL389-1	491	45	42	0.93	2	0.04	LCL391-1-tc	378	36	27	0.75	7	0.19
LCL391-1	455	70	62	0.89	18	0.26	LCL392-1-tc	384	25	22	0.88	6	0.24
LCL393-1	480	42	39	0.93	8	0.19	LCL393-1-tc	377	38	30	0.79	10	0.26
LCL394-1	480	44	39	0.89	7	0.16	LCL394-1-tc	384	39	30	0.77	11	0.28
LCL395-1	504	59	50	0.85	9	0.15	LCL395-1-tc	364	43	28	0.65	7	0.16
LCL403-1	483	59	56	0.95	7	0.12	LCL403-1-tc	365	34	28	0.82	10	0.29
LCL404-1	467	39	36	0.92	4	0.10	LCL404-1-tc	369	34	31	0.91	9	0.26

E "Best" Encountered Proofs of Individual TPTP Problems

Tables 10 and 11 below show for each of the 196 pure CD problems in TPTP 8.1.2[15] characteristics of the "best" proof encountered in our experiments. Proofs were there ordered by lexical comparison of the number tuple

$$\langle \mathit{cdproof}, \mathit{csize}, \mathit{tsize}, \mathit{height}, \mathit{nsimp}, \mathit{lemma\text{-}use}, \mathit{time}\rangle.$$

There *cdproof* is 0 or 1 depending on whether the proof is by CD (0) or some other calculus or unreported (1).[16] CD proofs are obtained directly from SGCD and CCS-Vanilla, and via straightforward conversions from Prover9 and CM-Prover. Compacted size, tree size and height of the proof, after n-simplification [76], are considered as *csize*, *tsize* and *height*. If the proof is not a CD proof, these values are not available. If n-simplification had no reducing effect on these size measures, then *nsimp* is 0, otherwise 1. If the proof was obtained without input lemmas, then *lemma-use* is 0, otherwise 1. The time in seconds used for proving (with input lemmas only for the last prover invocation with the lemmas) is *time*.

The tables show for each problem the data of the "best" proof: The problem and its rating in TPTP 8.1.2, the proof dimensions as *csize*/*tsize*/*height*, prefixed with *n* to indicate that n-simplification had reducing effect, *time* and the prover or configuration that produced the proof. An asterisk (*) indicates that the configuration involved lemma application. The suffix indicates the selection method for the lemmas: -LIN* learning with a linear model, -GNN* learning with a GNN model, -LIN-GNN* learning with a combined linear/GNN model, -HEU-X* lemma selection by sorting according to heuristic features. For all shown provers with exception of E and Vampire, the -LIN*, -GNN* and -LIN-GNN* configurations involved *iterative improvement*. Also some results for special configurations with heuristic lemma selection are included, indicated by the following prover names: Prover9-HEU-1* is Prover9 with 1,000 PSP optim input lemmas, Vampire-HEU-2* is Vampire with 1,000 PSP plain input lemmas and SGCD-HEU-3* is the setup described in Sect. 4. The data from the unstarred versions are from base runs of our experiments, before learning. A comprehensive table that extends Tables 10 and 11 is at http://cs.christophwernhard.com/cdtools/exp-lemmas/lemmas.html.

These tables for specific TPTP problems facilitate comparison with other provers and approaches. They include solutions for 189 of the 196 problems, where 6 of the 7 unsolved problems are rated 1.00 and one is rated 0.86. This shows that our discourse indeed takes place at the very edge of the overall state

[15] Those CD problems that remain after excluding from all 206 CD problems in the TPTP those two with status *satisfiable*, those five with a form of detachment that is based on implication represented by disjunction and negation, and those three with a non-atomic goal theorem. In our experiments we also used further problems, 312 in total, derived from these TPTP problems.

[16] In applications with CD, e.g [70], usually CD proofs (see Sect. 2.2) are desired.

of the art of first-order proving, as far as CD problems are concerned. The tables show that the lemma-enhanced configurations become more relevant for problems with increased difficulty rating. However, for four of the most difficult solved problems, lemmas selected with manually crafted heuristic led to success. It remains to cover this also with machine-learned lemma selection, where the identified difficult problems provide test cases. Only four of the proven problems could be not be proven with CD proofs, two of them actually quickly by Vampire and E, which calls for a deeper investigation of their non-CD proofs, their possible proof translation to CD, and the possible role of non-unit lemmas corresponding to binary resolution there.

Table 10. Data of "best" encountered proofs I/II

Problem	Rtg	Dims	Time	Prover
LCL006-1	0.00	5/7/4	0.06	CCS-Vanilla
LCL007-1	0.00	1/1/1	0.01	SGCD
LCL008-1	0.00	5/5/5	0.01	SGCD
LCL009-1	0.00	7/17/6	0.06	CCS-Vanilla
LCL010-1	0.00	5/8/5	0.03	SGCD
LCL011-1	0.00	7/16/7	0.07	CCS-Vanilla
LCL013-1	0.00	2/2/2	0.01	SGCD
LCL015-1	0.00	24/73/19	162.96	SGCD
LCL022-1	0.00	8/33/7	0.46	CCS-Vanilla
LCL023-1	0.00	7/18/7	0.09	CCS-Vanilla
LCL025-1	0.00	6/9/6	0.23	CCS-Vanilla
LCL026-1	0.00	22/29/15	2.76	SGCD
LCL027-1	0.00	3/3/3	0.01	SGCD
LCL029-1	0.00	7/14/6	10.79	CCS-Vanilla
LCL033-1	0.00	6/7/6	0.02	SGCD
LCL034-1	0.00	24/46/19	1.85	SGCD
LCL035-1	0.00	5/6/5	0.02	SGCD
LCL036-1	0.00	n 11/59/11	596.99	CCS-Vanilla
LCL038-1	0.00	52/119/27	86.11	SGCD-LIN-GNN*
LCL041-1	0.00	3/3/2	0.03	SGCD
LCL043-1	0.00	2/2/2	0.01	SGCD
LCL044-1	0.00	3/3/3	0.02	SGCD
LCL045-1	0.00	5/5/4	0.06	CMProver
LCL046-1	0.00	2/2/2	0.01	SGCD
LCL047-1	0.00	16/22/5	0.62	SGCD
LCL048-1	0.00	16/19/13	113.13	SGCD
LCL049-1	0.00	20/24/14	105.82	SGCD
LCL050-1	0.00	22/27/17	112.76	SGCD
LCL051-1	0.00	21/26/10	128.74	SGCD
LCL052-1	0.00	17/26/13	69.77	SGCD
LCL053-1	0.00	18/28/15	81.14	SGCD
LCL055-1	0.00	15/24/11	50.73	SGCD
LCL056-1	0.00	16/25/12	76.68	SGCD
LCL057-1	0.00	20/30/16	241.43	SGCD
LCL058-1	0.00	30/68/17	1.15	SGCD-LIN*
LCL059-1	0.00	15/19/5	854.67	CMProver
LCL064-1	0.00	6/9/6	0.12	CCS-Vanilla
LCL064-2	0.00	6/9/6	0.14	CCS-Vanilla
LCL065-1	0.00	7/9/7	0.04	SGCD
LCL066-1	0.00	7/7/7	0.07	SGCD
LCL067-1	0.00	10/20/6	914.91	CCS-Vanilla
LCL068-1	0.00	15/26/11	4.36	SGCD
LCL069-1	0.00	8/8/5	0.04	SGCD
LCL070-1	0.00	10/18/6	178.51	CCS-Vanilla
LCL071-1	0.00	13/16/6	24.78	SGCD
LCL072-1	0.00	7/8/4	0.02	SGCD
LCL075-1	0.00	8/20/8	0.16	Prover9
LCL076-1	0.00	7/9/7	0.06	SGCD
LCL076-2	0.00	1/1/1	0.01	SGCD
LCL077-1	0.00	6/8/6	0.05	SGCD
LCL079-1	0.00	3/3/3	0.02	SGCD
LCL080-1	0.00	9/10/8	383.43	CCS-Vanilla
LCL080-2	0.00	9/12/6	0.31	SGCD
LCL081-1	0.00	6/10/6	0.06	SGCD
LCL082-1	0.00	6/7/6	0.02	SGCD
LCL083-1	0.00	11/15/11	0.35	SGCD
LCL083-2	0.00	8/9/8	0.06	SGCD
LCL085-1	0.00	23/29/20	13.85	SGCD
LCL086-1	0.00	n 10/22/8	77.91	CCS-Vanilla
LCL087-1	0.00	8/12/8	0.03	SGCD
LCL088-1	0.00	12/18/8	1157.65	CMProver
LCL089-1	0.00	n 10/27/10	254.25	CCS-Vanilla
LCL090-1	0.00	14/20/14	203.68	SGCD
LCL091-1	0.00	11/16/10	35.83	CMProver
LCL092-1	0.00	n 12/34/11	369.00	CCS-Vanilla
LCL093-1	0.00	25/29/16	103.38	SGCD
LCL094-1	0.00	16/26/11	0.56	SGCD
LCL095-1	0.00	17/21/16	143.91	SGCD
LCL096-1	0.00	4/4/3	0.02	SGCD
LCL097-1	0.00	4/6/4	0.03	SGCD
LCL098-1	0.00	4/6/4	0.02	SGCD
LCL101-1	0.00	7/14/6	0.06	CCS-Vanilla
LCL102-1	0.00	7/7/4	0.06	CMProver
LCL103-1	0.00	10/16/9	984.38	CCS-Vanilla
LCL104-1	0.00	6/15/5	0.09	CCS-Vanilla
LCL106-1	0.00	4/4/4	0.01	SGCD
LCL107-1	0.00	5/9/5	0.03	SGCD
LCL108-1	0.00	7/20/6	0.04	Prover9
LCL110-1	0.00	7/9/6	1.01	CCS-Vanilla
LCL111-1	0.00	5/5/3	0.03	SGCD
LCL112-1	0.00	8/10/7	6.77	CCS-Vanilla
LCL113-1	0.00	15/18/5	1286.31	CMProver
LCL114-1	0.00	21/31/8	525.31	SGCD
LCL115-1	0.00	11/16/9	0.24	SGCD
LCL116-1	0.00	24/52/13	10.65	Prover9
LCL117-1	0.00	3/4/3	0.03	SGCD
LCL118-1	0.00	7/8/7	0.02	SGCD
LCL120-1	0.00	6/7/6	0.03	SGCD
LCL121-1	0.00	12/92/12	649.07	CCS-Vanilla
LCL123-1	0.00	10/45/7	0.75	CCS-Vanilla
LCL126-1	0.00	4/4/4	0.01	SGCD
LCL127-1	0.00	22/49/13	116.04	CMProver-LIN*
LCL128-1	0.00	24/352/19	4.05	Prover9
LCL129-1	0.00	11/42/11	336.22	CCS-Vanilla
LCL130-1	0.00	5/8/5	0.02	SGCD
LCL131-1	0.00	11/50/11	125.01	CCS-Vanilla
LCL256-1	0.00	20/31/13	114.94	SGCD
LCL257-1	0.00	7/13/6	0.09	CCS-Vanilla

Table 11. Data of "best" encountered proofs II/II

Problem	Rtg	Dims	Time	Prover
LCL355-1	0.00	1/1/1	0.01	SGCD
LCL356-1	0.00	2/3/2	0.01	SGCD
LCL357-1	0.00	2/2/2	0.01	SGCD
LCL358-1	0.00	4/5/4	0.02	SGCD
LCL359-1	0.00	3/5/3	0.02	SGCD
LCL360-1	0.00	1/1/1	0.01	SGCD
LCL361-1	0.00	4/4/3	0.02	SGCD
LCL362-1	0.00	4/4/4	0.02	SGCD
LCL363-1	0.00	6/6/5	0.03	SGCD
LCL364-1	0.00	9/14/8	45.56	CCS-Vanilla
LCL366-1	0.00	14/16/5	693.32	CMProver
LCL367-1	0.00	15/19/5	1.42	SGCD
LCL378-1	0.00	14/23/10	38.98	SGCD
LCL379-1	0.00	19/28/15	99.51	SGCD
LCL380-1	0.00	17/26/13	225.93	SGCD
LCL381-1	0.00	18/27/14	4.67	SGCD
LCL385-1	0.00	30/44/16	293.39	SGCD
LCL386-1	0.00	27/39/14	234.04	SGCD
LCL387-1	0.00	27/46/14	15.98	SGCD
LCL388-1	0.00	38/109/19	92.99	SGCD
LCL389-1	0.00	37/411/22	4.69	Prover9
LCL396-1	0.00	21/31/17	193.84	SGCD
LCL397-1	0.00	7/8/7	0.17	SGCD
LCL398-1	0.00	3/3/3	0.04	SGCD
LCL399-1	0.00	12/13/11	6.45	SGCD
LCL400-1	0.00	n 31/233/20	0.85	Prover9
LCL401-1	0.00	29/62/15	320.04	SGCD
LCL402-1	0.00	n 30/191/20	0.41	Prover9
LCL404-1	0.00	36/409/22	4.36	Prover9
LCL405-1	0.00	26/34/14	199.42	SGCD
LCL416-1	0.00	10/17/8	14.75	SGCD
LCL012-1	0.14	18/31/16	62.51	SGCD
LCL014-1	0.14	10/15/7	11.58	CCS-Vanilla
LCL016-1	0.14	39/89/14	195.72	SGCD
LCL017-1	0.14	50/129/17	194.28	SGCD
LCL018-1	0.14	22/54/16	147.63	SGCD
LCL019-1	0.14	35/132/18	202.29	SGCD
LCL021-1	0.14	68/225/19	331.46	SGCD
LCL024-1	0.14	10/15/9	79.22	CMProver
LCL030-1	0.14	8/16/6	62.68	CCS-Vanilla
LCL031-1	0.14	23/26/10	209.34	SGCD
LCL040-1	0.14	8/9/5	15.85	CCS-Vanilla
LCL042-1	0.14	8/8/4	499.52	CCS-Vanilla
LCL054-1	0.14	33/283/21	218.11	Prover9
LCL060-1	0.14	36/76/17	0.41	CCS-Vanilla-LIN*
LCL084-2	0.14	n 53/87/25	80.13	SGCD-GNN*
LCL084-3	0.14	52/98/23	295.65	SGCD-LIN-GNN*
LCL100-1	0.14	18/44/11	187.43	CCS-Vanilla-LIN*
LCL122-1	0.14	25/142/10	13.46	SGCD-LIN*
LCL166-1	0.14	51/155/18	182.67	SGCD
LCL369-1	0.14	22/34/16	2.82	SGCD
LCL370-1	0.14	27/39/13	242.93	SGCD
LCL371-1	0.14	27/39/13	233.94	SGCD
LCL373-1	0.14	37/68/14	708.81	SGCD
LCL382-1	0.14	29/53/18	6.21	SGCD
LCL384-1	0.14	13/23/5	683.01	CMProver
LCL390-1	0.14	31/45/14	281.13	SGCD
LCL403-1	0.14	40/94/16	30.54	SGCD-LIN*
LCL032-1	0.29	n 67/15362/35	106.73	Prover9
LCL099-1	0.29	20/41/6	459.30	SGCD
LCL105-1	0.29	37/109/11	90.54	Prover9-LIN*
LCL119-1	0.29	83/28624/27	76.07	Prover9
LCL365-1	0.29	10/15/9	429.17	CCS-Vanilla
LCL368-1	0.29	21/32/16	2.10	SGCD
LCL372-1	0.29	27/46/13	12.87	SGCD
LCL383-1	0.29	33/52/15	41.99	SGCD
LCL391-1	0.29	40/161/20	65.93	SGCD
LCL392-1	0.29	30/52/14	26.83	SGCD
LCL393-1	0.29	37/87/17	46.13	SGCD
LCL020-1	0.43	106/24989/37	21.65	Prover9-LIN*
LCL028-1	0.43	34/67/15	295.28	SGCD
LCL061-1	0.43	39/92/16	87.96	SGCD
LCL062-1	0.43	44/115/21	285.10	SGCD-LIN*
LCL124-1	0.43	27/130/10	76.25	SGCD-LIN*
LCL125-1	0.43	33/460/16	33.14	Prover9
LCL167-1	0.43	48/265/22	27.53	SGCD-GNN*
LCL374-1	0.43	33/77/17	42.47	SGCD
LCL375-1	0.43	43/103/20	56.44	SGCD-LIN*
LCL376-1	0.43	30/76/15	58.17	SGCD-GNN*
LCL394-1	0.43	41/81/17	267.22	SGCD
LCL875-1	0.43		0.298	Vampire
LCL037-1	0.57	n 72/45359/39	172.29	Prover9
LCL074-1	0.57	n 50/136/18	998.93	SGCD
LCL377-1	0.57	38/78/15	62.71	SGCD
LCL395-1	0.57	45/112/20	140.94	SGCD
LCL428-1	0.57		0.227	E
LCL109-1	0.86	72/348/22	226.55	Prover9-HEU-1*
LCL417-1	0.86		647.386	Vampire-HEU-2*
LCL422-1	0.86			
LCL876+1	0.93	70/396/22	227.17	Prover9-HEU-1*
LCL063-1	1.00		943.481	E
LCL073-1	1.00	46/3276/40	16.55	SGCD-HEU-3*
LCL418-1	1.00			
LCL419-1	1.00			
LCL420-1	1.00			
LCL421-1	1.00			
LCL425-1	1.00			
LCL426-1	1.00			

F Our Proof of LCL073-1 with MGTs

The following figures present our proof of LCL073-1 (Sect. 4) together with its lemma formulas, the MGTs of subproofs, in the notation for CD applications used in the classical literature.

1. $CCCCCpqCNrNsrtCCtpCsp$
2. $CCCppqCrq = \mathtt{D1D1D11}$
3. $CpCqCrr = \mathtt{D22}$
4. $CCCpCqqrCsr = \mathtt{D13}$
5. $CCCpqrCqr = \mathtt{D14}$
6. $CpCCpqCrq = \mathtt{D51}$
7. $CpCCCqprCsr = \mathtt{D56}$
8. $CCCpqrCCCqsCNrNpr = \mathtt{DDD1D176}\mathrm{n}$
9. $CCCpqCNCCCrpsCtsNrCCCrpsCts = \mathtt{D86}$
10. $CCCpqCNCrpNCCCsCptuCvuCrp = \mathtt{D8D19}$
11. $CpCCqrCCNqNpr = \mathtt{DD1D1D4.10.1}$
12. $CCCpNqCCqrCsrCtCCqrCsr = \mathtt{D1D6D1DD1D9D9DD11.341}$
13. $CCpCqrCCCsCtNprCqr = \mathtt{DDD12.D5D8.12.}\mathrm{n}\mathtt{7}$
14. $CCCCCpqrstCCCqrst = \mathtt{D1D13.D1D13.5}$

*15. $CCpqCCqrCpr = \mathtt{DD1D13.DDDD13.69.11.10.D14.D14.1}$

Fig. 3. Our proof of LCL073-1 in the notation of Meredith [41,77]. For each factor of the overall D-term the argument term of its MGT is there shown in Łukasiewicz's notation.

1. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,q),\mathtt{i}(\mathtt{n}(r),\mathtt{n}(s))),r),t),\mathtt{i}(\mathtt{i}(t,p),\mathtt{i}(s,p))))$
2. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,p),q),\mathtt{i}(r,q))) = \mathtt{D}(1,\mathtt{D}(1,\mathtt{D}(1,1)))$
3. $\mathtt{P}(\mathtt{i}(p,\mathtt{i}(q,\mathtt{i}(r,r)))) = \mathtt{D}(2,2)$
4. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,\mathtt{i}(q,q)),r),\mathtt{i}(s,r))) = \mathtt{D}(1,3)$
5. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,q),r),\mathtt{i}(q,r))) = \mathtt{D}(1,4)$
6. $\mathtt{P}(\mathtt{i}(p,\mathtt{i}(\mathtt{i}(p,q),\mathtt{i}(r,q)))) = \mathtt{D}(5,1)$
7. $\mathtt{P}(\mathtt{i}(p,\mathtt{i}(\mathtt{i}(\mathtt{i}(q,p),r),\mathtt{i}(s,r)))) = \mathtt{D}(5,6)$
8. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,q),r),\mathtt{i}(\mathtt{i}(\mathtt{i}(q,s),\mathtt{i}(\mathtt{n}(r),\mathtt{n}(p))),r))) = \mathtt{D}(\mathtt{D}(\mathtt{D}(1,\mathtt{D}(1,7)),6),\mathrm{n})$
9. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,q),\mathtt{i}(\mathtt{n}(\mathtt{i}(\mathtt{i}(\mathtt{i}(r,p),s),\mathtt{i}(t,s))),\mathtt{n}(r))),\mathtt{i}(\mathtt{i}(\mathtt{i}(r,p),s),\mathtt{i}(t,s)))) = \mathtt{D}(8,6)$
10. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,q),\mathtt{i}(\mathtt{n}(\mathtt{i}(r,p)),\mathtt{n}(\mathtt{i}(\mathtt{i}(\mathtt{i}(s,\mathtt{i}(p,t)),u),\mathtt{i}(v,u))))),\mathtt{i}(r,p))) = \mathtt{D}(8,\mathtt{D}(1,9))$
11. $\mathtt{P}(\mathtt{i}(p,\mathtt{i}(\mathtt{i}(q,r),\mathtt{i}(\mathtt{i}(\mathtt{n}(q),\mathtt{n}(p)),r)))) = \mathtt{D}(\mathtt{D}(1,\mathtt{D}(1,\mathtt{D}(4,10))),1)$
12. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,\mathtt{n}(q)),\mathtt{i}(\mathtt{i}(q,r),\mathtt{i}(s,r))),\mathtt{i}(t,\mathtt{i}(\mathtt{i}(q,r),\mathtt{i}(s,r)))))$
    $= \mathtt{D}(1,\mathtt{D}(6,\mathtt{D}(1,\mathtt{D}(\mathtt{D}(1,\mathtt{D}(9,\mathtt{D}(9,\mathtt{D}(\mathtt{D}(11,3),4)))),1))))$
13. $\mathtt{P}(\mathtt{i}(\mathtt{i}(p,\mathtt{i}(q,r)),\mathtt{i}(\mathtt{i}(\mathtt{i}(s,\mathtt{i}(t,\mathtt{n}(p))),r),\mathtt{i}(q,r)))) = \mathtt{D}(\mathtt{D}(\mathtt{D}(12,\mathtt{D}(5,\mathtt{D}(8,12))),\mathrm{n}),7)$
14. $\mathtt{P}(\mathtt{i}(\mathtt{i}(\mathtt{i}(\mathtt{i}(\mathtt{i}(p,q),r),s),t),\mathtt{i}(\mathtt{i}(\mathtt{i}(q,r),s),t))) = \mathtt{D}(1,\mathtt{D}(13,\mathtt{D}(1,\mathtt{D}(13,5))))$

*15. $\mathtt{P}(\mathtt{i}(\mathtt{i}(p,q),\mathtt{i}(\mathtt{i}(q,r),\mathtt{i}(p,r))))$
    $= \mathtt{D}(\mathtt{D}(1,\mathtt{D}(13,\mathtt{D}(\mathtt{D}(\mathtt{D}(\mathtt{D}(13,6),9),11),10))),\mathtt{D}(14,\mathtt{D}(14,1)))$

Fig. 4. Our proof of LCL073-1 with the ATP-oriented notation for formulas and D-terms of the paper, arranged such that it mimics Meredith's notation.