Structure-Generating First-Order Theorem Proving

Christoph Wernhard

University of Potsdam, Germany

Abstract

Provers based on the connection method can be much stronger than currently believed. We substantiate this thesis with certain generalizations of know techniques. In particular, we generalize proof structure enumeration interwoven with unification – the proceeding of goal-driven connection and clausal tableaux provers – to an interplay of goal- and axiom-driven processing. It makes heuristic restrictions known from saturating provers applicable. Proof structure terms, proof objects that allow to specify and implement various ways of building proofs, are central there. As Meredith's condensed detachment represents the prototypical base case of such proof structure terms, applying to a subclass of first-order Horn problems, we focus on this class. Experiments show that the approach keeps up with state-of-the-art first-order provers, leads to short proofs, solves an ATP challenge problem, and is useful in machine learning for ATP. A general aim is to make ATP more accessible to systematic investigations in the space between calculi and implementation aspects.

1. Introduction

Our thesis is that provers based on the connection method can be much stronger than currently believed. We substantiate this with certain generalizations of know techniques.

Current realizations of the CM operate by enumerating proof structures, interwoven with unification of atomic formulas that are associated with nodes in the structure. The scope of variables in these formulas includes the whole structure; in technical terms, they are *rigid* [15]. Failure of unification on a partially built-up structure effects backtracking. All those structures that would extend this partial structure are abandoned at once.

As the proof structures are often represented in a tree form as clausal tableaux [17], we call provers that proceed in this way *CM-CT provers*, for *Connection Method/Clausal Tableaux*. Examples are the Prolog Technology Theorem Prover [39], SETHEO [18], leanCoP [28, 27] and CMProver [7, 44, 45]. Aside of the CM and clausal tableaux also model elimination [19] is a conceptual model that covers such provers. The underlying principle of enumerating proof structures in combination with unification is already present in [30, 31], as a progress compared to enumerating formula instantiations. In [12, Sect. 3] resolution theorem proving is contrasted as a formula enumeration process with tableau enumeration. With techniques based on proof structure enumeration, each proof structure usually appears at most once, while, e.g., in proof search by resolution the same proof may appear several times.

Typically, CM-CT provers perform the structure enumeration wrapped in iterative deepening upon the maximal value of some structure size measure. A basic motivation there is completeness while avoiding the storage requirements of breadth-first search [39]. In addition, alternate depth measures are explored as heuristics variations, e.g., [18, Sect. 6.2].

info@christophwernhard.com (C. Wernhard)

CM-CT provers proceed goal-driven through an initially instantiated goal formula at the top of the proof structure, e.g., with a ground clause obtained from Skolemizing universal variables in the original goal formula.

Structure-generating first-order theorem proving as discussed here generalizes these principles. One major influence is the consideration of *proof structure terms* in Carew A. Meredith's condensed detachment (CD) [32]. These are full binary trees,[1] called D-terms [49], referring to their convenient presentation as terms with the binary function symbol D. A D-term permits to associate a most general formula that is proven by it if the constants or leaf nodes are associated with given axioms. See, e.g., [24] for original examples. D-terms allow to specify many useful operations on proof structures in immediate and straightforward ways. For example, notions of proof size. Or proof composition: proofs of lemmas can simply be inserted into an overall proof term. This is, for example, necessary to obtain an overall proof in proving with precomputed and preselected lemmas [34]. Also decomposition of proofs is straightforward: Each subterm of a D-term is itself a D-term that represents the proof of a lemma.

CD plays a core role in *witness theory* [36] and is known [37] as the first technical approach to the Curry-Howard correspondence or "formulas as types" [14, 13]. The problems covered by CD form a subclass of first-order Horn problems. Although many historic successes in ATP were with CD problems [56, 52, 22, 53, 54, 43, 10, 55, 42], CD was considered in ATP mostly just as a restricted form of hyperresolution. Emphasis on its proof structures became in ATP prevalent only later, based on observing parallels to the CM [48, 49].

Thus our investigations focuses on CD problems, for which these D-terms are readily available, and we call our prototypical system to investigate the approach *SGCD – Structure-Generating theorem proving for Condensed Detachment*.

Our starting point is enumerating proof structures, interwoven with unification, as performed by the CM-CT provers. SGCD emits the proofs in increasing *levels*, characterized for example by some size measure of the D-terms, e.g., number of nodes or height, similar to the iterative deepening in CM-CT provers. The simple form of the D-terms, just full binary trees, allows to find precise upper bounds of the search space growth for increasing levels in the *The On-Line Encyclopedia of Integer Sequences* https://oeis.org/ (*OEIS*) [26]. Also measures that were so far hardly considered for CM-CT provers can be taken into account. For example compacted size, that is, the number of nodes of the minimal DAG representation of the proof tree. Or we can characterize sets of structures level by level in some inductive way. With one specific such characterization, the so-called PSP-level (*PSP* for *Proof-SubProof*), SGCD finds a strikingly short proof of a historic problem by Łukasiewicz [49] and a proof of a problem that was solved by Meredith but was so far very hard for ATP [34, 55].

Another issue to address is the purely goal-driven operation of the CM-CT provers. Their depth-first search effects that subgoals are re-proven again and again. Moreover, information about the goal in form of the initial goal instantiation is often propagated only to a few subgoals that are very close to the overall goal.

From our enumeration view, instantiating the top node with a goal is just an option. We can also leave it with variables and collect their bindings in each enumerated structure. Under such a binding, the goal is a lemma formula that can be proven by the structure in a given level from

[1] A binary tree is *full* if each node has 2 or 0 children.

given axioms. We call this mode *axiom-driven*. SETHEO was used for bottom-up preprocessing in this way [38]. Hypertableaux [3] may be viewed as proving by maintaining such derived lemmas in a specific way. SGCD can be invoked in a loop where goal-driven phases alternate with axiom-driven phases, for increasing levels. The parameters that control this loop are highly configurable. Lemmas obtained by the axiom-driven phases are cached and available in later phases, both axiom- and goal-driven. They can be deleted on the basis of heuristics that are well-known from saturating theorem proving, for example deleting lemmas whose term height exceeds a threshold. Such heuristics are usually not available in CM-CT provers, because the atom instances maintained by them are deeper instantiated, impacted always from the overall proof structure in contrast to just the substructure that proves the lemma.

In the following Sect. 2 we provide more background on condensed detachment. Then, in Sect. 3 we explicate our approach with presenting the system *SGCD (Structure-Generating theorem proving for Condensed Detachment)*, covering the range from general concepts to implementation aspects and experimental results. In Sect. 4 we discus some particular open issues and speculative ideas concerning the approach. Section 5 concludes the paper.

Detailed descriptions of experiments are provided in [47] and [34].

The system is available as free software from http://cs.christophwernhard.com/cdtools/, where also additional data and presentations for the experiments can be found, including HTML-formatted comprehensive result tables and full logs.

2. Background: Condensed Detachment for ATP

CD was developed in the mid-1950s by Carew A. Meredith as an evolution of the method of substitution and detachment practiced by Łukasiewicz [32, 16, 33, 25]. Reasoning steps are by detachment, or modus ponens, under implicit substitution by most general unifiers. Its basic application field is the investigation of axiomatizations of propositional logics from a first-order meta level. Some of the most advanced formal proofs ever developed by humans were such applications of CD by Meredith and some other researchers. From a first-order ATP perspective, a *CD problem* consists of *proper axioms* (briefly *axioms*), that is positive unit clauses, a *goal theorem*, that is a single negative ground unit clause (representing a universally quantified atomic goal theorem after Skolemization) and the following ternary Horn clause that models detachment.

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

The premises of *Det* are called *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 considered as an *inference rule*. With this view, the most straightforward way to describe a *CD inference step* from an ATP perspective is to consider it as a positive hyperresolution step such that from *Det* and two positive unit clauses, used as major and minor premise, a third positive unit clause is inferred. A *CD proof* is a proof of a CD problem constructed with the CD inference rule, in contrast to, for example, a proof involving binary resolution steps involving

Det that yield non-unit clauses. Prover9 actually by default chooses positive hyperresolution as inference rule for CD problems and thus produces CD proofs for these.

The structure of CD proofs can be represented in a very simple and convenient way as full binary trees or terms, which can be directly taken into account as objects in proving. In ATP we find this aspect in the connection methods (CM) [4, 5, 6], where the proof structure as a whole is in the focus, in contrast to extending a set of formulas by inferred formulas. This view of CD has been made precise and elaborated in [48, 49].

As already mentioned, we call the structure representations of CD proofs *D-terms*. A D-term is a term built from D as binary function symbol and atom labels as constant symbols or *primitive* D-Terms, labels of axioms. In other words, it is a full binary tree where the leaf nodes are labels of axioms. The following line shows four example D-terms for axiom labels $1, 2$.

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

The mapping between the representation of CD proofs in terms of CD inference steps and D-terms is straightforward: The use of an axiom corresponds to a primitive D-term. A CD inference step corresponds to a D-term $\mathsf{D}(d_1, d_2)$ where d_1 is the D-term that proves the unit clause for the major premise and d_2 is the D-term that proves the unit clause for the minor premise.

A D-term taken together with an *axiom assignment*, that is, a mapping of primitive D-terms to axioms, represents a proof in full.

With a D-term a most general formula that is proven by it from the axioms is associated through unification, constrained by the axioms for the leaf nodes and the requirements imposed by *Det* for the inner nodes.[2] Following [49], we call this formula the *most general theorem (MGT)* of the D-term with respect to the axiom assignment. The MGT is unique, modulo renaming of variables. Of course, for a given axiom assignment not all D-terms necessarily have an MGT. If the unification constraints can not be satisfied, we say the D-term has no MGT. And, of course, also depending on the axiom assignment, it is well possible that two different D-terms have the same MGT or that the MGT of one is subsumed by the MGT of the other. A CD problem also includes a goal theorem, a ground atom. A D-term is a proof of the problem if its MGT with respect to the axioms subsumes the goal theorem.

Example 1. *Consider the axiom assignment* $\alpha \stackrel{\text{def}}{=} \{1 \mapsto \mathsf{P}(\mathsf{i}(x,\mathsf{i}(x,x)))\}$ *declaring the primitive D-term* 1 *as label of an axiom known as* Mingle *[42]. The MGT of the primitive D-term* 1 *is then just this axiom, modulo renaming of variables. The MGT of the D-term* $\mathsf{D}(1,1)$ *is* $\mathsf{P}(y)\sigma$ *where* σ *is the most general unifier of the set of pairs*

$$\{\{\mathsf{P}(\mathsf{i}(x,y)),\ \mathsf{P}(\mathsf{i}(x',\mathsf{i}(x',x')))\},\ \{\mathsf{P}(x),\ \mathsf{P}(\mathsf{i}(x'',\mathsf{i}(x'',x'')))\}\}.$$

The most general unifier of these pairs is $\sigma = \{x \mapsto \mathsf{i}(x'',\mathsf{i}(x'',x'')),\ x' \mapsto \mathsf{i}(x'',\mathsf{i}(x'',x'')),\ y \mapsto \mathsf{i}(\mathsf{i}(x'',\mathsf{i}(x'',x'')),\mathsf{i}(x'',\mathsf{i}(x'',x'')))\}$. *Hence, after renaming variables (recall that the MGT is just characterized modulo renaming of variables) the MGT of* $\mathsf{D}(1,1)$ *is* $\mathsf{P}(\mathsf{i}(x,\mathsf{i}(x,x)),\mathsf{i}(x,\mathsf{i}(x,x)))$. *A D-term proves as goal theorems all ground instances of its MGT, which is defined with respect to an axiom assignment. So here* $\mathsf{D}(1,1)$ *proves* $\mathsf{P}(\mathsf{i}(\mathsf{a},\mathsf{i}(\mathsf{a},\mathsf{a})),\mathsf{i}(\mathsf{a},\mathsf{i}(\mathsf{a},\mathsf{a})))$*, and also, for example* $\mathsf{P}(\mathsf{i}(\mathsf{i}(\mathsf{a},\mathsf{b}),\mathsf{i}(\mathsf{i}(\mathsf{a},\mathsf{b}),\mathsf{i}(\mathsf{a},\mathsf{b}))),\mathsf{i}(\mathsf{i}(\mathsf{a},\mathsf{b}),\mathsf{i}(\mathsf{i}(\mathsf{a},\mathsf{b}),\mathsf{i}(\mathsf{a},\mathsf{b})))))$.

[2] There is actually some fine-print here with respect the usual notion of most general unifier [48, 49, 14].

D-terms, 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 by ATP systems in general, for D-terms there are three immediate size measures:

- The *tree size* of a D-term is the number of its inner nodes.
- The *height* of a D-term is the number of edges of the longest downward path from the root to a leaf.
- 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.[3]

In the literature compacted size is also called *length*, height *level* and tree size *CDcount* [43].

Example 2. *The D-term* $\mathsf{D}(\mathsf{D}(1,\mathsf{D}(1,1)),\mathsf{D}(\mathsf{D}(1,\mathsf{D}(1,1)),\mathsf{D}(\mathsf{D}(1,1),1)))$ *has tree size* 8*, height* 4 *and compacted size* 5*. If we replace the second occurrence of* 1 *from left by* 2 *we obtain*

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

Tree size and height remain unaltered, but the compacted size is then 7*.*

CD proofs suggest and allow a specific form of lemmas, which we called *unit/subtree* lemmas, reflecting two views: 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[4]) of the overall D-term. If they occur multiply there, they would be factored in the minimal DAG of the overall D-term. Both views are linked in that the formula of a lemma is the MGT of its D-term. The most adequate size measure for D-terms that takes the compression achieved by unit/subtree lemmas into account is compacted size. From the perspective of proof structure compression methods, unit/subtree lemmas have the property that the compression target is unique, simply because each tree is represented by a unique minimal DAG.

A textual representation of a D-term that respects its compacted size, the DAG compression by unit/subtree lemmas, is provided by a list of factor equations. There, distinct subproofs with multiple incoming edges in the DAG receive numeric labels by which they are referenced. Meredith uses such a form with Polish notation for D-terms and their MGTs, e.g., in [24], which is discussed in [49].

Example 3. *Consider the D-term* $\mathsf{D}(\mathsf{D}(1,\mathsf{D}(1,1)),\mathsf{D}(\mathsf{D}(1,\mathsf{D}(1,1)),\mathsf{D}(\mathsf{D}(1,1),1)))$ *from Examp. 2. As a list of factor equations it can be represented as follows.*

$$\begin{array}{rcl} 2 & = & \mathsf{D}(1,1) \\ 3 & = & \mathsf{D}(1,2) \\ 4 & = & \mathsf{D}(3,\mathsf{D}(3,\mathsf{D}(2,1))) \end{array}$$

Here we assume that labels 2–4 *are not used for axioms and thus are free for use as reference labels. Label* 4*, which is not referenced, is associated with the root or the overall D-term.*

[3] It is well known that a tree or set of trees is represented by a unique (modulo isomorphism) *minimal* DAG, which is maximally factored (it has no multiple occurrences of the same subtree) or, equivalently, is minimal with respect to the number of nodes.

[4] We use *subtree* with the meaning common in computer science and matching the notion of *subterm*: A subtree of a tree T is a tree consisting of a node in T and all of its descendants in T.

Compared to general first-order ATP problems, CD problems are restricted: Only positive unit clauses; a single negative ground clause; and a single ternary Horn clause. Also equality is not explicitly considered. Nevertheless, core characteristics of general first-order ATP problems are present: first-order variables, a binary function symbol and cyclic predicate dependency. The generalization to arbitrary Horn problems is actually not difficult [46] but not considered here.

3. SGCD – Structure-Generating Theorem Proving for Condensed Detachment

We explicate our approach by means of describing the SGCD system, which is implemented in SWI-Prolog [50]. There we enter the space between calculi and "system descriptions". In a sense, the calculus (e.g., condensed detachment as some inference system) just tells us in an inductive way how proof structures are built-up and relate to proven formulas or MGTs. Although the inductive specification suggests a construction, for theorem proving in practice, there are many ways in which the proof structures can be actually built. We think, this is not merely the domain of ad-hoc low-level heuristics, but a potentially large field that should be systematically approached as well.

The programming language Prolog is inherently rather close to proof structure enumeration with incorporated unification. In fact, it expresses arbitrary computations according to this principle. A predicate (or nondeterministic procedure) enumerates alternate bindings of output variables, each binding corresponding to a different (implicitly represented) proof structure (or way to evaluate the predicate by recurring to further nondeterministic predicates). Thus, Prolog is here not just suitable as an implementation language, but also to abstractly describe the involved methods.

3.1. The Core Enumeration Predicate, its Modes and the Cache

We assume CD problems and D-terms as proof structures. Further we assume that the set of D-terms is grouped into *levels*, possibly but not necessarily disjoint, where the strict subterms of D-terms in a higher level are members of some lower level. The set of the sets of all D-terms with a specific tree size would be an example of such a level grouping. We then say that the level is *characterized* by the tree size. At the core of SGCD is a ternary predicate that, for a given level, enumerates all pairs of a D-term in the level and a formula such that the formula is the MGT of the D-term, with respect to some assumed axioms that do not explicitly appear as parameter.

`enum_dterm_mgt_pairs(`*+Level*`,` *?D-term*`,` *?Formula*`)`

Depending on the mode in which `enum_dterm_mgt_pairs` is invoked, that is, which of the *D-term* and *Formula* arguments are inputs or outputs, it realizes different functionalities.

- If both are input arguments, then the predicate *verifies* that *D-term* is a proof of *Formula*. The predicate then either succeeds (enumerates the empty binding as sole value) or fails.

- If only *D-term* is an input and *Formula* an output, it *computes the MGT* of *D-term*. The predicate then either enumerates a single value for *Formula* or, in case *D-term* has no MGT, fails.
- If only *Formula* is an input and *D-term* is an output, it acts as a *goal-driven prover*, enumerating proofs of *Formula*.
- If both *D-term* and *Formula* are outputs, it enumerates pairs of a proof and its MGT, that is, it generates lemmas with proofs in an *axiom-driven* way.

Invoked goal-driven, i.e., with *Formula* as input and *D-term* as output, for increasing values of *Level* it is in essence a CM-CT prover with iterative deepening that is specialized to CD problems.

To process the lemmas enumerated by *axiom-driven* invocations, i.e., with both *D-term* and *Formula* as outputs, SGCD maintains a *cache* of $\langle$*Level*, *D-term*, *Formula*$\rangle$ triples, solutions of `enum_dterm_mgt_pairs(`*Level*`, `*D-term*`, `*Formula*`)`.

If lemmas are computed and cached for increasing levels, this cache can be used within the implementation of `enum_dterm_mgt_pairs` to solve subproblems with a lower level than that of the top call from the cache, instead of recomputing them. This is possible not just within axiom-driven top calls, but also within goal-driven invocations. A part of the proof search is then replaced by retrieval from the cache.[5].

Moreover, the cache can be subjected to heuristic restrictions based on formulas, the MGTs, as known from resolution-based (or more generally, *saturating*) provers. There, for example, lemmas with formulas that are subsumed by an already present lemma or whose term height exceeds some threshold are discarded. Such heuristic restrictions are not possible with conventional CM-CT provers, because these do not have MGTs at hand, just formulas that are due to rigid variables more deeper instantiated, in dependence of the context where they occur in the proof (this is discussed in more depth as IPT vs. MGT in [49]). With such heuristic restrictions, the replacement of search by lemmas realized with SGCD's cache then effects an actual modification, a heuristically motivated restriction, of the explored search space.

3.2. Level Characterizations

SGCD can be used with alternate versions of `enum_dterm_mgt_pairs` for different level characterizations, including by tree size, height, maximal tree size and maximal height. The number of distinct D-terms for increasing values of some size measure gives an upper bound of the number of trees to consider in proof search by enumerating D-terms level-by-level. This number is just an upper bound, because it does not take into account that D-term enumeration is interwoven with unification. Through the unification, which is constrained by given axioms and possibly a given goal, fragments of D-terms for which unifiability fails are immediately discarded and D-terms extending them are skipped in the enumeration. Heuristic restrictions may further limit the number of enumerated structures.

The number of distinct D-terms for increasing values of some size measure also indicates a measure-specific size value up to which it is easily possible to compute for given axioms all proofs, together with the lemma formulas proven by them.

[5]This realizes a *replacing* form of lemma application [2, 34]

Table 1
The numbers of distinct D-terms for a single axiom (or full binary trees) of given size n for different size measures.

n		0	1	2	3	4	5	6
Tree size	oeis:A000108	1	1	2	5	14	42	132
Height	oeis:A001699	1	1	3	21	651	457,653	210,065,930,571
Compacted size	oeis:A254789	1	1	3	15	111	1,119	14,487

Table 2
The numbers of distinct D-terms for a single axiom (or full binary trees) in PSP-level n and of compacted size n.

n		0	1	2	3	4	5	6	7	8
$\lvert\mathcal{PSPL}evel(n)\rvert$	oeis:A001147	1	1	3	15	105	945	10,395	135,135	2,027,025
Compacted size	oeis:A254789	1	1	3	15	111	1,119	14,487	230,943	4,395,855

If we assume a single axiom such that we can identify D-terms with full binary trees without any additional labeling, the sequences of the number of distinct D-terms for increasing tree size, height or compacted size are well-known and can be found in the *OEIS*, with identifiers A000108, A001699, and A254789, respectively, as shown in Table 1.

Aside of these "conventional" criteria, our use of proof structure terms also permits to use inductive structure specifications as level characterizations. Particularly prospective there is the *PSP-level* (indicating *Proof-SubProof*), specified as follows.

(i) Structures in PSP-level 0 are the axiom identifiers.
(ii) Structures in PSP-level $n+1$ are all structures $\mathsf{D}(d_1, d_2)$ and $\mathsf{D}(d_2, d_1)$ where d_1 is a structure in PSP-level n and d_2 is a (not necessarily strict) subterm of d_1 or an axiom identifier.

The PSP-level was motivated by an analysis [49] of Meredith's variant [24] of Łukasiewicz's completeness proof for his shortest single axiom for implicational logic [20], where it was observed that most steps in Meredith's proof can be ascribed the relationship between levels that underlies the PSP-level. PSP-levels are disjoint. All D-terms in PSP-level n have compacted size n. However, the cardinality of D-terms at PSP-level n grows slower than that of D-terms of compacted size n, according to th oeis:A001147 in contrast to oeis:A254789. Table 2 compares the initial values of both sequences. It follows that the enumeration of D-terms according to the PSP-level is "incomplete", that is, there are D-terms that are not a member of any PSP-level.

Experimental successes involving enumeration by PSP-level include relatively short proofs for problems where exhaustive search for a shortest proof seems unfeasible [47, Sect. 4.5], a short proof for Łukasiewicz's shortest axiom, LCL038-1 [47, Sect. 4.6][49], and an automatically found proof of a challenge problem, the "Meredith single axiom" LCL073-1 [34].

3.3. Combining Goal- and Axiom-Driven Reasoning

SGCD combines goal- and axiom-driven invocation of `enum_dterm_mgt_pairs` by proceeding in two nested loops, as shown in Fig. 1. The goal formula is given as input parameter g.

```
Cache := ∅;
for l := 0 to maxLevel do begin
    for m := l to l + preAddMaxLevel do begin
        enum_dterm_mgt_pairs(m, d, g);
        throw proof_found(d)
    end;
    News := {⟨l, d, f⟩ | enum_dterm_mgt_pairs(l, d, f)};
    if News = ∅ then throw exhausted;
    Cache := merge_news_into_cache(News, Cache)
end
```

Figure 1: The nested loops of the SGCD theorem proving method.

The parameters $maxLevel$ and $preAddMaxLevel$ are specified with the configuration. We assume that the core predicate `enum_dterm_mgt_pairs` is for some particular configured level characterization. It enumerates argument bindings nondeterministically. If it succeeds once in the inner loop, the exception `proof_found(`d`)` returns the bound D-term d.

The procedure `merge_news_into_cache(`$News$, $Cache$`)` merges the newly generated

$$\langle \textit{Level}, \textit{D-term}, \textit{Formula} \rangle$$

triples $News$ with the cache $Cache$, optionally taking into account configured heuristics. For example, sorting the union of the triples according to increasing term height of the lemma formulas and truncating the sorted list after, say, 1,000 entries.

If $maxLevel$ is configured as 0, the method proceeds purely in goal-driven mode with the inner loop performing iterative deepening upon the level m from 0 up to $preAddMaxLevel$. The similarity to CM-CT provers can even be shown empirically by comparing the sets of solved TPTP problems (see Sect. 3.5). If combined with axiom-driven lemma generation, generally successful configurations of $preAddMaxLevel$ typically have values of 0–3.

Of course, also variations of the basic schema of Fig. 1 can be useful. For example to enumerate alternate proofs instead of exiting with the first found proof through the **throw** statement.

For lemma generation [34], SGCD can be run with some resource bound, e.g. a timeout or a Prolog inference limit, or until axiom-driven enumeration exhausts by heuristic limitations. After SGCD terminates, the content of the cache represents a set of generated lemmas. Also lemmas deleted at `merge_news_into_cache` may be maintained in a separate "passive" store and be used for example in lemma generation [34] and theorem finding (Sect. 3.5, [47, Sect. 4.3]).

"Hybrid" configurations are possible, where different level characterizations are used for the goal- and axiom-driven phases. The cache has initially not to be empty: It is possible to initialize it with externally generated lemmas (possibly by another invocation of SGCD) and modify the initial settings of the l and m parameters. We call this *lemma injection*. Search at lower levels is then completely replaced by retrieval from these external lemmas. SGCD with lemma injection was used successfully for lemmas selected with the help of machine learning [34].

3.4. Implementation Aspects and the CD Tools Environment

SGCD is integrated in *CD Tools*, an environment to experiment with CD, which is embedded in SWI-Prolog[6] and utilizes *PIE* [44, 45] as basic support for first-order ATP. For experimenting with machine learning [34] SGCD was invoked via Bash scripts, also in portfolio settings where different configurations are tried in parallel.

CD Tools provides various interfaces to external worlds. It supports, for example, conversion to and from Łukasiewicz's Polish notation. There are means to enumerate and test *TPTP* problems whether they represent a CD problem and to canonicalize the vocabulary of such CD problems. A pre-defined association of about 170 common names for about 120 well-known formulas and combinators, e.g., from [33, 42], helps to specify these as axioms and to detect them among computed lemmas. Also some support for handling combinators and λ-terms is implemented, which is used, e.g., for the second included prover CCS [46].

CD Tools includes implementations of many concepts and operations introduced or discussed in [49], including D-term comparison by $\geq_c$, various notions of regularity, variations of the *organic* property (based on calling a SAT solver[7] or based on previously proven lemmas), the *prime* property, and *n-simplification*, that is, replacing complex subproofs of minor premises whose goal is irrelevant for the conclusion with a primitive subproof.

Many of these operations are available to SGCD for heuristic restrictions that can be optionally configured to be applied either immediately to the results of axiom-driven invocations of `enum_dterm_mgt_pairs` or when merging new results into the cache.

Attempting to introduce at least a little bit of goal-direction in the axiom-driven mode, an experimental heuristic restriction takes the number of distinct subterms in a formula that do not appear in the goal as a negative criterion.

For handling large (millions of tree nodes) proof terms, CD Tools provides an implementation of MGT computation and verification that transparently operates on a factored representation.

3.5. Experimental Results

SGCD was evaluated in a number of experiments, most of them on the 196 pure CD problems in *TPTP 8.0.0* (there are 10 further CD problems in the *TPTP* with status *satisfiable* or a more complex problem form.) We call this problem corpus *TPTPCD*. The report [47] describes various experiments with SGCD on that corpus. Additional experiments with machine learning for ATP and proving a challenge problem for ATP, the "Meredith single axiom" LCL073-1, are described in [34]. Experiments with CCS, the second prover included in CD Tools, are reported in [46]. Comprehensive result tables and log files can be found at http://cs.christophwernhard.com/cdtools/. Here we give brief summaries of the experiments with SGCD.

SGCD solves more *TPTPCD* problems than Prover9, which, like SGCD, creates CD proofs. However, SGCD fails on five problems that Prover9 can solve. It solves no new problems (i.e.,

[6] No efforts were made to support other Prolog systems. Brief tries with free systems reckoned as particularly fast, notably *YAP* and *Eclipse*, unfortunately gave the impression that these are broken in their recent versions. SWI-Prolog provides some library predicates that proved very useful, for example `variant_sha1/3` or `term_factorized/3`.

[7] Via PIE, which provides a call interface to SAT and QBF solvers by translating between PIE's Prolog term representation of formulas and DIMACS or QDIMACS files, commonly used by the solvers. In experiments we used *MiniSat* [9].

problems rated with 1.00), but 93% of the problems that are solvable at all by a first-order prover and 95% of the problems that can be solved by the *E* prover. See [47, Sect. 4.1].

These 176 solutions are obtained as the joint results of SGCD in four configurations. Key settings of these configurations are as follows. Configuration 1: level characterization by tree size; maximal 1000 cache entries; goal-driven phases upon 2 levels (*preAddMaxLevel* = 1); 165 solutions. Configuration 2: similar to configuration 1 but height as level characterization; 109 solutions. Configuration 3: similar to configuration 1 but maximal 3000 cache entries; 161 solutions. Configuration 4: similar to configuration 1 but goal-driven phases just upon 1 level (*preAddMaxLevel* = 0) and strong limitations of the formula size of the cached lemmas just slightly above the maximal sizes found in the input problems; 80 solutions.

In specialized settings with lemma injection, SGCD solves more difficult problems, including the 1.00-rated LCL073-1 [34]. With or without lemma injection it outperforms Vampire (without lemma injection 285 vs. 263 solutions for the 312 problems considered in [34]; with lemma injection 289; Vampire, when given lemmas obtained from SGCD is boosted to 285 solutions). For an overall comparison of SGCD with and without lemma injection with various provers, invoked with and without lemmas provided by SGCD, see http://cs.christophwernhard.com/cdtools/exp-lemmas/lemmas.html. An excerpt of this table is also reproduced in [35, App. E].

In purely goal-driven configurations SGCD is indeed very similar to the CM-CT provers and can roughly solve the same problems as several of them taken together. See [47, Sect. 4.2] and http://cs.christophwernhard.com/cdtools/exp-tptpcd-2022-07/table_3.html.[8]

In purely axiom-driven configurations SGCD can perform theorem finding. It finds proofs for Łukasiewicz's *68 Theses* in a single run in 2.5 minutes. These theses from a textbook by Łukasiewicz were used extensively with OTTER [21] by Larry Wos [52, 22, 53, 54]. They could be solved by OTTER in a single run after the introduction of weight templates [52, 53]. The focus in [54] was to find short overall proofs for subsets of the theses that represent axiom systems. Our initially obtained proofs can not compete in compacted size with the carefully developed proofs from [54], but with respect to tree size. See [47, Sect. 4.3].

Prover9 computes for CD problems CD proofs that can be represented by D-terms. Hence we can directly compare Prover9's proofs with the D-terms output by SGCD. In general, SGCD finds smaller proofs than Prover9 in its default mode, moderately smaller with respect to compacted size and height, and drastically smaller with respect to tree size. See [47, Sect. 4.4].

CCS can be applied in a configuration that returns proofs with ascertained minimal compacted size by exhaustive search [46]. This succeeds for about 44% problems of the *TPTPCD* corpus. Enumeration by PSP-level is particularly useful to find proofs with small compacted size in cases where the exhaustive search for ascertained minimal compacted size appears to be unfeasible. See [47, Sect. 4.5].

With enumeration by PSP-level, SGCD finds a short proof of LCL038-1, the most difficult of the three subproblems of proving completeness of *Łukasiewicz's shortest single axiom for implicational logic*. Short extensions of this proof that solve the other two easier subproblems were then performed with naive structure enumeration predicates provided by CD Tools.

[8] Also *leanCoP 2.1* [27] was tried. With a single exception, all *TPTPCD* problems are provided in the *TPTP* CNF format, which is not accepted by *leanCoP*. Experiments with a 1 hour timeout on conversions to *TPTP* FOF format (with axioms and goal conjecture distinguished) led to 72 or 51 solved problems, depending on whether the major or minor subgoal appears first in axiom *Det*.

SGCD's proof of LCL038-1 is drastically shorter than proofs by other ATP systems and even substantially shorter than the proofs by Łukasiewicz and Meredith. The combined proof for all three subproblems is a few steps shorter than the proofs by these logicians. See [47, Sect. 4.6] and also [49].

Enumeration by PSP-level for lemma generation led in general to good results in the machine learning scenario. It appears to supply to other provers useful lemmas that would not be found by their own calculi. For proving LCL073-1, enumeration by PSP-level was an essential component, applied in a hybrid way just for the axiom-driven phases, where the goal-driven phases were by height as level characterization [34].

4. Some Issues Requiring Further Research

The approach of structure-generating theorem proving currently raises questions that seem to require further investigations. We outline some of these, organized into two major topics.

4.1. Stronger Proof Compressions

SGCD so far focuses on unit/subtree lemmas, which correspond to the sharing of subtrees in DAGs. There are stronger proof compressions that correspond to shared trees with "holes". Technically this can be expressed with binary resolution that results in Horn clauses (the body atoms correspond to the "holes"), with tree grammar compressions where nonterminals may contain variables (the "holes"), or with combinators, where the "holes" are mapped to subtree sharing in DAGs (by getting rid of the variables through combinators) [46]. The combinator representation is in particular suitable for proof structure enumeration, interwoven with unification, initialized with a goal, as in SGCD when operated goal-driven. This is realized with *CCS* [46], our second experimental prover in CD Tools. This prover may be viewed as implementing the connection structure calculus [8], restricted to CD problems. Depending on the configuration, it can simulate other calculi, in particular goal-driven versions of binary resolution.

The most immediate level characterization for enumeration with the combinator representation is compacted size. We then obtain in essence DAG enumeration, with iterative deepening upon the DAG size. Allowing combinators to enter the D-terms is only justified if they permit proofs with smaller compacted size. That is, if the proving D-term with combinators has smaller compacted size than those without combinators. Systematic enumeration of proof DAGs is not as simple as tree enumeration because it requires to maintain subtrees and the lemmas proven by them which already appear in the proof under construction.[9] Rigid variables are not sufficient, some copying or forgetting of variables [8] is required.

So far, CCS has been tried only in goal-driven mode. It is not clear, how enumeration by compacted size transfers to SGCD with its axiom-driven mode. The level characteristics for SGCD are determined for a given proof structure "globally", independently from context: tree size, height, PSP-level. During enumeration by compacted size, a given structure has to be

[9] As outlined in [46], the DAG enumeration by CCS is with a certain variation of the well-known *value-number method* [1, 11].

assessed differently, context depending: if it already occurred in the partially built-up structure, its value is zero (it can be attached again without increasing the overall compacted size). Is the PSP-level a version of compacted size that is limited in a certain sense? If so, is it a distinguished best version within that limitation?

CCS performs roughly comparable with CM-CT provers. If configured without combinators, it yields as a bonus proofs with guaranteed minimal compacted size. While stronger compressions certainly can achieve in principle drastic savings, it is currently not clear which role this may play in practice. In experiments with *TPTP* Horn problems, we observed that the involvement of a compressing combinator in the proof by CCS was often an indicator for the problem stemming from a crafted series.

4.2. Systematization of Level Characterizations

It seems desirable to formally specify and systematize criteria and possibilities of level characterizations for SGCD. Levels may be disjoint (e.g., tree size, height, compacted size, PSP-level) or cumulative (e.g., tree size less or equal than some value). There is an interplay with the subterm relationship among proof terms. There may be gaps with respect to given axioms: some level may have no member with MGT, but the next level has members with MGT. A method that incrementally generates levels then may exhaust too early. The level characterization may be incomplete, as observed for the PSP-level in Sect. 3.2. As noted in Sect. 4.1, compacted size is in a sense context dependent and seems to bear a certain relationship to PSP-level enumeration.

How can different level characterizations be combined? Abstractly into a single new level characterization? The experiment with LCL073-1 shows that "hybrid" configurations with different level characterizations for the goal- and axiom-driven phases are important. A trivial pragmatic approach to combine different level characterizations is invoking prover instances for each of them in portfolio mode. Also "heuristic" limitations like, e.g., maximal term height of lemma formulas, which are so far handled in SGCD orthogonally to the level characterizations, might be incorporated into level characterizations.

In semi-naive evaluation (e.g., [41]) certain recursive invocations of enumeration predicates are replaced with accesses to "delta" predicates that represent results newly obtained in the immediately preceding round. This realizes a forward-reasoning trigger effect: a delta predicate is evaluated first, only for its new results the remaining subgoals are considered. A related effect can be observed in some cases for our proof/formula pair enumeration predicate, where recursive invocations are for the given level minus one. But so far, such effects and their consequences for improving SGCD's enumeration algorithms have not been systematically studied.

One may speculate that the grouping into levels suggests another view on the problem of first-order ATP, a view that may be of interest with respect to abstractly explaining the success of the axiom-driven proceeding, and possibly relate to dynamic programming. While the problem of proving a subgoal can be broken down into proving several further subgoals, these may share variables and are thus not independent. If we consider instead of proving a single problem the problem of finding *all lemmas in a level*, depending on the level characterization, the problem may recur into subproblems of finding all lemmas in lower levels. These subproblems, however, can be solved independently from each other.

The view of enumeration or generation by level suggests to try in ATP research also an "empirical" style of pattern-observing, inspired by physics, as exemplified for combinatory logic in [51].

5. Conclusion

We have generalized proof structure enumeration interwoven with unification performed by goal-driven connection and clausal tableaux provers to an interplay of goal- and axiom-driven processing. It makes heuristic restrictions known from saturating provers applicable.

Proof structure terms, proof objects that allow to specify and implement various ways of building proofs, were a central tool. As Meredith's condensed detachment represents the prototypical base case of such proof structure terms, applying to a subclass of first-order Horn problems, so far we focused on this class. An expressive generalization of CD to first-order Horn problems is already considered and implemented in a related system [46]. Toward first-order logic in general, there are theoretical results [36]. Possibly also [23]. There is the CM and the connection structure calculus. Other possible ingredients to generalization are the simulation of factoring with combinators and ongoing work on transforming resolution proofs such that they lead to a consequence, a clause that subsumes a given goal, instead of the empty clause. It appears that the axiom-driven operation permits incorporation of dedicated equality reasoning without resorting to *lazy* variants of paramodulation.

Concerning related work, similarly to SGCD an implementation [29] of hypertableaux [3] creates lemmas axiom-driven level-by-level. However much less flexibly configurable and incorporating goal-driven phases only in a rudimentary way, by termination before a level is completed as soon as a clause subsuming the goal is generated. A recent enhancement of E by machine learning supplements a classification based on *derivation history* as clause selection guidance [40]. Since in structure-generating proving the D-term itself, or, so-to-speak, the derivation history, provides the main guidance, this enhancement might possibly be viewed as introducing a little bit of structure-generating proving into into E.

Experiments have shown that on CD problems the structure-generating approach keeps up with state-of-the-art first-order provers, leads to short proofs, solves an ATP challenge problem, and is useful in machine learning for ATP. Open issues that seem to require further research include the incorporation of stronger proof compressions and a systematic investigation of the possibilities to group proof structures into ordered "levels" governing the enumeration order of SGCD.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers – Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[2] O. L. Astrachan and M. E. Stickel. Caching and lemmaizing in model elimination theorem provers. In D. Kapur, editor, *CADE-11*, pages 224–238. Springer, 1992. doi: 10.1007/3-540-55602-8_168.

[3] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In J. J. Alferes, L. M. Pereira, and E. Orlowska, editors, *JELIA'96*, volume 1126 of *LNCS (LNAI)*, pages 1–17. Springer, 1996. doi: 10.1007/3-540-61630-6_1.

[4] W. Bibel. *Automated Theorem Proving*. Vieweg, Braunschweig, 1982. doi: 10.1007/978-3-322-90102-6. Second edition 1987.

[5] W. Bibel. *Deduction: Automated Logic*. Academic Press, 1993.

[6] W. Bibel and J. Otten. From Schütte's formal systems to modern automated deduction. In R. Kahle and M. Rathjen, editors, *The Legacy of Kurt Schütte*, chapter 13, pages 215–249. Springer, 2020. doi: 10.1007/978-3-030-49424-7_13.

[7] I. Dahn and C. Wernhard. First order proof problems extracted from an article in the Mizar mathematical library. In M. P. Bonacina and U. Furbach, editors, *FTP'97*, RISC-Linz Report Series No. 97–50, pages 58–62, Linz, 1997. Joh. Kepler Univ. URL https://www.logic.at/ftp97/papers/dahn.pdf.

[8] E. Eder. A comparison of the resolution calculus and the connection method, and a new calculus generalizing both methods. In E. Börger, H. Kleine Büning, and M. M. Richter, editors, *CSL '88*, volume 385 of *LNCS*, pages 80–98. Springer, 1989. doi: 10.1007/BFb0026296.

[9] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003. doi: 10.1007/978-3-540-24605-3_37.

[10] B. Fitelson and L. Wos. Missing proofs found. *J. Autom. Reasoning*, 27(2):201–225, 2001. doi: 10.1023/A:1010695827789.

[11] A. Genitrini, B. Gittenberger, M. Kauers, and M. Wallner. Asymptotic enumeration of compacted binary trees of bounded right height. *J. Comb. Theory, Ser. A*, 172:105177, 2020. doi: 10.1016/j.jcta.2019.105177.

[12] C. Goller, R. Letz, K. Mayr, and J. Schumann. SETHEO V3.2: recent developments – system abstract. In A. Bundy, editor, *CADE-12*, volume 814 of *LNCS (LNAI)*, pages 778–782, 1994. doi: 10.1007/3-540-58156-1_59.

[13] J. R. Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997. doi: 10.1017/CBO9780511608865.

[14] J. R. Hindley and D. Meredith. Principal type-schemes and condensed detachment. *J. Symbolic Logic*, 55(1):90–105, 1990. doi: 10.2307/2274956.

[15] R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handb. of Autom. Reasoning*, volume 1, chapter 3, pages 101–178. Elsevier, 2001. doi: 10.1016/b978-044450813-3/50005-9.

[16] E. J. Lemmon, C. A. Meredith, D. Meredith, A. N. Prior, and I. Thomas. Calculi of pure strict implication. In J. W. Davis, D. J. Hockney, and W. K. Wilson, editors, *Philosophical Logic*, pages 215–250. Springer Netherlands, 1969. doi: 10.1007/978-94-010-9614-0_17. Reprint of a technical report, Canterbury University College, Christchurch, 1957.

[17] R. Letz. *Tableau and Connection Calculi. Structure, Complexity, Implementation*. Habilitationsschrift, TU München, 1999. Available from http://www2.tcs.ifi.lmu.de/~letz/habil.ps, accessed Jul 19, 2023.

[18] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high-performance theorem prover. *J. Autom. Reasoning*, 8(2):183–212, 1992. doi: 10.1007/BF00244282.

[19] D. W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, 1978.

[20] J. Łukasiewicz. The shortest axiom of the implicational calculus of propositions. In

Proc. of the Royal Irish Academy, volume 52, Sect. A, No. 3, pages 25–33, 1948. URL http://www.jstor.org/stable/20488489.

[21] W. McCune. OTTER 3.3 Reference Manual. Technical Report ANL/MCS-TM-263, Argonne National Laboratory, 2003. https://www.cs.unm.edu/~mccune/otter/Otter33.pdf, accessed Jul 19, 2023.

[22] W. McCune and L. Wos. Experiments in automated deduction with condensed detachment. In D. Kapur, editor, *CADE-11*, volume 607 of *LNCS (LNAI)*, pages 209–223. Springer, 1992. doi: 10.1007/3-540-55602-8_167.

[23] N. D. Megill. A finitely axiomatized formalization of predicate calculus with equality. *Notre Dame J. of Formal Logic*, 36(3):435–453, 1995. doi: 10.1305/ndjfl/1040149359.

[24] C. A. Meredith and A. N. Prior. Notes on the axiomatics of the propositional calculus. *Notre Dame J. of Formal Logic*, 4(3):171–187, 1963. doi: 10.1305/ndjfl/1093957574.

[25] D. Meredith. In memoriam: Carew Arthur Meredith (1904–1976). *Notre Dame J. of Formal Logic*, 18(4):513–516, Oct 1977. doi: 10.1305/ndjfl/1093888116.

[26] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. http://oeis.org, 2021.

[27] J. Otten. Restricting backtracking in connection calculi. *AI Communications*, 23(2-3): 159–182, 2010. doi: 10.3233/AIC-2010-0464.

[28] J. Otten and W. Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003. doi: 10.1016/S0747-7171(03)00037-3.

[29] B. Pelzer and C. Wernhard. System description: E-KRHyper. In F. Pfenning, editor, *CADE-21*, volume 4603 of *LNCS (LNAI)*, pages 503–513. Springer, 2007. doi: 10.1007/978-3-540-73595-3_37.

[30] D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.

[31] D. Prawitz. Advances and problems in mechanical proof procedures. *Machine Intelligence*, 4:59–71, 1969. Reprinted with author preface in J. Siekmann, G. Wright (eds.): *Automation of Reasoning, vol 2: Classical Papers on Computational Logic 1967–1970, Springer, 1983, pp. 283–297.*

[32] A. N. Prior. Logicians at play; or Syll, Simp and Hilbert. *Australasian Journal of Philosophy*, 34(3):182–192, 1956. doi: 10.1080/00048405685200181.

[33] A. N. Prior. *Formal Logic*. Clarendon Press, Oxford, 2nd edition, 1962. doi: 10.1093/acprof:oso/9780198241560.001.0001.

[34] M. Rawson, C. Wernhard, Z. Zombori, and W. Bibel. Lemmas: Generation, selection, application. In R. Ramanayake and J. Urban, editors, *TABLEAUX 2023*, LNCS (LNAI). Springer, 2023. Preprint (extended version): https://arxiv.org/abs/2303.05854.

[35] M. Rawson, C. Wernhard, Z. Zombori, and W. Bibel. Lemmas: Generation, selection, application. *CoRR*, abs/2303.05854, 2023. doi: 10.48550/arXiv.2303.05854.

[36] A. Rezuş. *Witness Theory – Notes on λ-calculus and Logic*, volume 84 of *Studies in Logic*. College Publications, 2020.

[37] A. Rezuş. The Curry-Howard Correspondence revisited (2019b). In *Witness Theory – Notes on λ-calculus and Logic* Rezuş [36], pages 209–214.

[38] J. M. P. Schumann. DELTA — A bottom-up preprocessor for top-down theorem provers. In *CADE-12*, volume 814 of *LNCS (LNAI)*, pages 774–777. Springer, 1994. doi: 10.1007/3-540-58156-1_58.

[39] M. E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog

compiler. *J. Autom. Reasoning*, 4(4):353–380, 1988. doi: 10.1007/BF00297245.

[40] M. Suda. Improving ENIGMA-style clause selection while learning from history. In A. Platzer and G. Sutcliffe, editors, *CADE 28*, volume 12699 of *LNCS (LNAI)*, pages 543–561. Springer, 2021. doi: 10.1007/978-3-030-79876-5_31.

[41] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I: Classical Database Systems. Computer Science Press, Rockville, Maryland, 1988.

[42] D. Ulrich. A legacy recalled and a tradition continued. *J. Autom. Reasoning*, 27(2):97–122, 2001. doi: 10.1023/A:1010683508225.

[43] R. Veroff. Finding shortest proofs: An application of linked inference rules. *J. Autom. Reasoning*, 27(2):123–139, 2001. doi: 10.1023/A:1010635625063.

[44] C. Wernhard. The PIE system for proving, interpolating and eliminating. In P. Fontaine, S. Schulz, and J. Urban, editors, *PAAR 2016*, volume 1635 of *CEUR Workshop Proc.*, pages 125–138. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1635/paper-11.pdf.

[45] C. Wernhard. Facets of the PIE environment for proving, interpolating and eliminating on the basis of first-order logic. In P. Hofstedt et al., editors, *DECLARE 2019*, volume 12057 of *LNCS (LNAI)*, pages 160–177, 2020. doi: 10.1007/978-3-030-46714-2_11.

[46] C. Wernhard. Generating compressed combinatory proof structures – an approach to automated first-order theorem proving. In B. Konev, C. Schon, and A. Steen, editors, *PAAR 2022*, volume 3201 of *CEUR Workshop Proc.* CEUR-WS.org, 2022. https://arxiv.org/abs/2209.12592.

[47] C. Wernhard. CD Tools — Condensed detachment and structure generating theorem proving (system description). *CoRR*, abs/2207.08453, 2023. doi: 10.48550/arXiv.2207.08453.

[48] C. Wernhard and W. Bibel. Learning from Łukasiewicz and Meredith: Investigations into proof structures. In A. Platzer and G. Sutcliffe, editors, *CADE 28*, volume 12699 of *LNCS (LNAI)*, pages 58–75. Springer, 2021. doi: 10.1007/978-3-030-79876-5_4.

[49] C. Wernhard and W. Bibel. Investigations into proof structures. *CoRR*, abs/2304.12827, 2023. doi: 10.48550/arXiv.2304.12827. submitted.

[50] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. doi: 10.1017/S1471068411000494.

[51] S. Wolfram. *Combinators – A Centennial View*. Wolfram Media Inc, 2021. Accompanying webpage: https://writings.stephenwolfram.com/2020/12/combinators-a-centennial-view/, accessed Jun 30, 2022.

[52] L. Wos. Automated reasoning and Bledsoe's dream for the field. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 297–345. Kluwer Academic Publishers, 1991. doi: 10.1007/978-94-011-3488-0_15.

[53] L. Wos. The resonance strategy. *Computers Math. Applic.*, 29(2):133–178, 1995. doi: 10.1016/0898-1221(94)00220-F.

[54] L. Wos. The power of combining resonance with heat. *J. Autom. Reasoning*, 17(1):23–81, 1996. doi: 10.1007/BF00247668.

[55] L. Wos. Conquering the Meredith single axiom. *J. Autom. Reasoning*, 27(2):175–199, 2001. doi: 10.1023/A:1010691726881.

[56] L. Wos, S. Winker, W. McCune, R. Overbeek, E. Lusk, R. Stevens, and R. Butler. Automated reasoning contributes to mathematics and logic. In M. E. Stickel, editor, *CADE-10*, pages 485–499. Springer, 1990. doi: 10.1007/3-540-52885-7_109.