CEUR-WS.org/Vol-3201/paper12.pdf

Generating Compressed Combinatory Proof Structures – An Approach to Automated First-Order Theorem Proving

Christoph Wernhard

University of Potsdam, Germany

Abstract

Representing a proof tree by a combinator term that reduces to the tree lets subtle forms of duplication within the tree materialize as duplicated subterms of the combinator term. In a DAG representation of the combinator term these straightforwardly factor into shared subgraphs. To search for proofs, combinator terms can be enumerated, like clausal tableaux, interwoven with unification of formulas that are associated with nodes of the enumerated structures. To restrict the search space, the enumeration can be based on proof schemas defined as parameterized combinator terms. We introduce here this "combinator term as proof structure" approach to automated first-order proving, present an implementation and first experimental results. The approach builds on a term view of proof structures rooted in condensed detachment and the connection method. It realizes features known from the connection structure calculus, which has not been implemented so far.

Keywords

clausal tableaux, combinators, condensed detachment, connection structure calculus, first-order ATP, grammar-based tree compression, proof compression, proof schemas, proof search, proof structures

1. Introduction

Goal-driven first-order provers such as *leanCoP* [1], *PTTP* [2], *SETHEO* [3] and *CMProver* [4], which can be described as based on clausal tableaux [5], the connection method [6, 7] or model elimination [8], in essence enumerate tree-shaped proof structures, interwoven with unification of formulas that are associated with nodes of the structures. While such provers do not compete with state-of-the art systems in the range of solvable problems, they have merits that are relevant in certain contexts: Proofs are typically emitted as data structures of simple and detailed forms, making them suitable as inputs for further processing. The provers facilitate comparing alternate proofs of a problem or influencing the shape of proofs. Interpolation is an example where such features are quite useful [9, 10]. The goal-driven mode of operation and constraining the proof shape can be stretched to using such a prover as processor of a full-fledged programming language (Prolog). In general, through iterative deepening the emitted proofs tend to be short,

PAAR'22: 8th Workshop on Practical Aspects of Automated Reasoning, August 11–12, 2022, Haifa, Israel
EMAIL: info@christophwernhard.com (C. Wernhard)
© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

which again is useful for further processing, including integration with other systems and presentation for humans. Implementations following the approach are typically manageable and small (with *leanCoP* outbidding all others [11]), making them attractive for adaptation to specific logics [12, 13, 14] and novel combinations with other techniques [15, 16, 17, 18, 19, 20]. Another aspect of the approach with potential long-term relevance is its role as a foundation for systematic investigations of first-order ATP, as for example in [6, 21, 5, 22].

Here we aim to preserve the merits of the clausal tableau or conventional connection approach, with respect to theory as well as practice, while moving on to stronger proving capabilities. In particular, we address a crucial limitation of the conventional techniques: the restriction to tree structures in contrast to DAG structures, where nodes with multiple incoming edges correspond to multiply used lemmas. The issue has three aspects: First, DAGs permit to share duplicated subtrees where the conventional handling of variables as rigid (their scope is the whole structure) is not sufficient, because, in general, each use of a lemma takes it in a fresh copy. Second, for DAG enumeration a naturally adequate depth measure for iterative deepening is the DAG size, i.e., the number of inner nodes of the DAG, in contrast to conventionally used measures such as the number of tree nodes or tree height.

The third aspect is to extend the forms of duplication in the proof structure that materialize as duplicated subtrees and are thus shareable in the DAG representation. As an example of this aspect, consider the proof tree

$$2(2(2(2(2(2(2(21))))))). \tag{i}$$

If leaf label 1 represents the axiom $\mathsf{p}(\mathsf{a})$, leaf label 2 the axiom $\mathsf{p}(x) \to \mathsf{p}(\mathsf{f}(x))$ and, given a proof d_1 of $P \to Q$ and a proof d_2 of R, the juxtaposition $d_1 d_2$ represents an inference step by modus ponens with unification, i.e., $d_1 d_2$ is a proof of $Q\sigma$, where σ is the most general unifier of P and R, then (i) is a proof of $\mathsf{p}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a})))))))))$. Although in (i) intuitively the repeated use of axiom 2 constitutes some kind of duplication, there are no duplicated subtrees in (i), except of leaves. The minimal DAG of (i) is identical to the tree. The intuitively observed form of duplication is not captured by duplicated subtrees that could be factored in the DAG representation.

The idea is now to use combinator terms as representations of proof structures, which enrich the possibilities of sharing substructures. A combinator is technically a λ-term without free variables. Combinators form the basis of combinatory logic, introduced in the 1920s by Moses Schönfinkel [23], developed further by Haskell B. Curry [24], and later used for functional programming languages [25] and for an empirical computation theory [26] (see [27] for a comprehensive bibliography). As common in the literature, we notate function application, which now generalizes the modus ponens inference steps considered before, by left associative juxtaposition. The $\mathbf{B}$ combinator, for example, which can be characterized by the equivalence

$$\mathbf{B}xyz \equiv x(yz). \tag{ii}$$

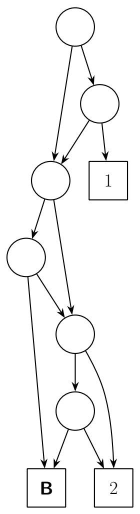
allows to represent (i) by the combinator term

$$\mathbf{B}(\mathbf{B}22)(\mathbf{B}22)(\mathbf{B}(\mathbf{B}22)(\mathbf{B}22)1). \tag{iii}$$

Eliminating in (iii) the occurrences of **B** by rewriting with (ii) results in (i). In (iii) the subtrees **B**22 and **B**(**B**22)(**B**22) occur multiply such that each of them has two incoming edges in the minimal DAG of (iii) (Fig. 1). The number of inner nodes of that DAG is only 6, compared to 8 for (i).

For *proof search*, terms such as (iii), that is, proof structures with combinators and identifiers of proper axioms[1] as constants, can be *enumerated*, just like clausal tableaux. The interwoven formula unification is constrained from the root by the proof goal, from the leaves by copies of axioms and from within the structure by constraints induced by the modus ponens inference rule. A combinator is handled just as an axiom with an implicational formula that is associated in a specific way with its definition, the *principal type* [28] of the combinator.

Figure 1: Term (iii) as DAG.



The proof search is then actually upon compressed structures. On the one hand, these may be shorter than expanded structures but on the other hand, the search space may also be increased because a single expanded structure can be represented by many different compressed structures. For example, (i), an expanded structure because it does not involve a combinator, can be represented in compressed form with the **B** combinator not just by (iii), but also by five further structures with the same size of the minimal DAG, including, e.g., **B**22(**B**22(**B**22(**B**221))).

The balancing between short proofs and the vast number of compression possibilities in proof search with compressed structures is addressed here with a specific form of *proof schemas*, patterns, consisting of a function symbol with arguments and an associated defining combinator term. If search is performed with a specified set such proof schemas, the proof structures are constructed only from constants representing proper axioms and the function patterns. In proofs, instances of the function patterns can be rewritten into their defining combinator terms with at most a linear increase of the DAG size, making the proofs independent from possibly ad-hoc schemas used for search.

In brief, we introduce an approach to first-order theorem proving that has its background in goal-driven clausal tableaux or conventional realizations of the connection method, but differently from these, is characterized by search over combinatory compressed proof structures, with DAGs instead of trees as basic structure for re-usable lemmas and DAG size as basic measure for iterative deepening. A proof schema mechanism based on combinator terms is provided to control search in presence of compression.

In Sect. 2 we flesh out the approach and sketch features of *CCS*, an implemented experimental prover. We then present in Sect. 3 empirical results, so far for the problems in the *TPTP* [29] that are directly expressed using modus ponens and unification, i.e., that are condensed detachment [30] problems. In Sect. 4 we conclude with pointers to related work , next steps, open issues and perspectives. The implemented prover and auxiliary tools are components of the latest version of the *CD Tools* system [31], available

[1]A *proper axiom* is an axiom from the application problem, in contrast to the logical machinery.

as free software from

http://cs.christophwernhard.com/cdtools/.

That website also provides supplementary result tables, including graphical proof visualizations, as well as detailed outputs from the experiments described in the paper.

2. Outline of the Approach

2.1. Tree/Term/DAG Representation of Condensed Detachment Proofs

As basic inference mechanism we consider condensed detachment [32, 33, 34, 30, 22], in other words modus ponens with unification, originally due to Carew A. Meredith. An inference step by condensed detachment can be informally described as deriving from major premise $P \rightarrow Q$ and minor premise R the conclusion Q under the most general unifier of P and R.[2] The structural component of a condensed detachment proof is a full binary tree where the left child of an inner node represents the subproof of the major premise and the right child that of the minor premise. Leaves represent axioms. Such a tree can be conveniently written as a so called D-term, a term with the binary function symbol D for inner nodes and constants identifying axioms as leaves. The following term provides an example. It tree visualization is shown in Fig. 2(a).

$$\mathsf{D}(\mathsf{D}(1,1),\mathsf{D}(\mathsf{D}(1,\mathsf{D}(1,1)),\mathsf{D}(1,\mathsf{D}(1,1)))). \tag{iv}$$

For given axioms associated with the constants, the "most general" formula proven by a D-term, called its *most general theorem (MGT)* [22] (or *principal type* [34, 28]), can be determined through unification as follows: The MGT of leaf, a constant D-term, is a fresh instance of the axiom labeled by the constant. If $P \rightarrow Q$ and R are the MGTs of D-terms d_1 and d_2, respectively, then the MGT of $\mathsf{D}(d_1, d_1)$ is $Q\mathsf{mgu}(\{P, R\})$.[3] If there is no substitution that satisfies these unification constraints for all nodes of a D-term, its MGT is said to be *not defined*.

Notice that each instance of an axiom, associated with a leaf, is before consideration of the unification constraints a fresh copy of the axiom. By the unification constraints, variables are propagated throughout the tree. Hence the variables are rigid, that is, their scope comprises all formulas associated with nodes of the whole structure. For the example axioms in Table 1 it is easy to see that the MGT of the D-term 1 is $\mathsf{P}(\mathsf{a})$, the MGT of 2 is $\mathsf{P}(x) \rightarrow \mathsf{P}(\mathsf{f}(x))$ (modulo renaming x to a fresh variable), the MGT of $\mathsf{D}(2, 1)$ is $\mathsf{P}(\mathsf{f}(\mathsf{a}))$, and the MGT of $\mathsf{D}(2, \mathsf{D}(2, 1))$ is $\mathsf{P}(\mathsf{f}(\mathsf{f}(\mathsf{a})))$.

D is the only non-constant function symbol used in D-terms. Moreover, a condensed detachment step with premises $P \rightarrow Q$ and R may be viewed as *application* of the function

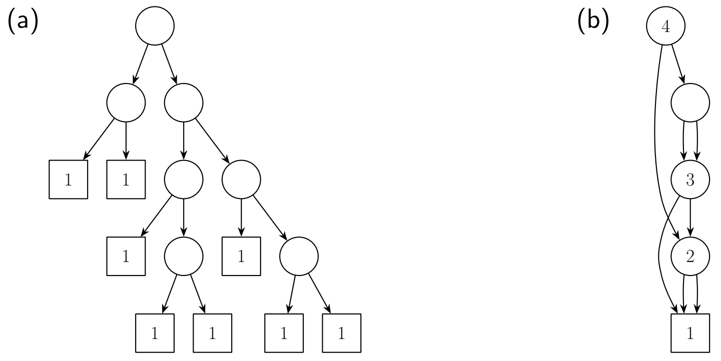
[2]For a recent comprehensive account of condensed detachment that relates to the connection method see [22].

[3]We use the common postfix notation for application of a substitution. If M is a set of pairs of terms that has a unifier, then $\mathsf{mgu}(M)$ denotes the most general unifier of M. If M contains just a single pair $\{t, u\}$, we use $\mathsf{mgu}(\{t, u\})$ as shorthand for $\mathsf{mgu}(\{\{t, u\}\})$. For a precise account of condensed detachment, we assume w.l.o.g. that $\mathsf{mgu}(M)$ has the *clean* property [22].

Table 1
Examples of proper axioms with numbers as identifying constants.

Axiom ID	Axiom formula
1	$\mathsf{P}(\mathsf{a})$
2	$\mathsf{P}(x) \to \mathsf{P}(\mathsf{f}(x))$

Figure 2: D-term (iv), shown in applicative notation in (v), as tree (a) and as DAG (b). In (b) the labels of inner nodes in (b) show the corresponding factor labels from (vi).



$\lambda x.Q\mathsf{mgu}(\{P, x\})$ to R. This suggests to present complex D-terms in the common notation for application in λ-terms and terms of combinatory logic. For example, $\mathsf{D}(\mathsf{D}(1,2), \mathsf{D}(3,4))$ is then written as 12(34), and (iv) as

$$11(1(11)(1(11))). \tag{v}$$

Useful measures for the size of a D-term are *tree size*, i.e., the number of inner nodes, *height*, i.e., the height of the tree, and *compacted size*, i.e., the number of inner nodes of the minimal DAG representing the tree, or, equivalently, the number of distinct compound subterms. For example, the D-term shown in (iv), (v) and Fig. 2 has tree size 7, height 4 and compacted size 4. A DAG can be represented textually in efficient form by a list of factors, paired with labels to express references. The following list of factors gives an example for the minimal DAG of (v) as shown in Fig. 2.(b).

$$[2 = 11,\ 3 = 12,\ 4 = 2(33)]. \tag{vi}$$

In this example numbers are used as factor labels, starting from 2, since 1 was the largest number used as axiom identifier. The presentation order mimics building up the proof by applying condensed detachment to previously obtained lemmas, with the axiom identifiers, just 1 in the example, presupposed. The largest number, 4 in the example, corresponds to the root of the term. The original expanded D-term, (v) in the example, can be obtained from the root factor, 2(33) in the example, by exhaustively replacing labels with their designated factor. If the factor list represents the *minimal* DAG, the compacted size of the represented D-term can be read off by counting the occurrences of application on the right hand sides.

2.2. Combinatory Compression

As running example, we consider proving $\mathsf{P}(\mathsf{f}^8(\mathsf{a}))$, which stands for $\mathsf{P}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a})))))))))$, from the axioms in Table 1.[4] There is a single D-term which proves this, namely

$$2(2(2(2(2(2(2(21))))))). \tag{vii}$$

Its tree size as well as its compacted size are both 8. As indicated by the compacted size, it has 8 distinct compound subterms: 21, 2(21), 2(2(21)), ..., 2(2(2(2(2(2(2(21))))))), each of them with just a single occurrence. Hence its minimal DAG is identical to the tree. That axiom 2 is applied eight times is not reflected in any multiply occurring compound subterm. This can be remedied with the **B** combinator, defined as

$$\mathbf{B} \overset{\text{def}}{=} \lambda xyz\,.\,x(yz). \tag{viii}$$

Occurrences of **B** where it is applied to three arguments, matching the number of arguments of its defining λ-term, can be eliminated by rewriting to the body of that λ-term, i.e., with the equivalence-preserving rewrite rule

$$\mathbf{B}xyz \to_{\text{rew}} x(yz). \tag{ix}$$

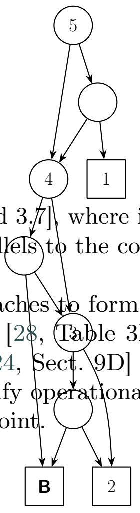
We now generalize D-terms to permit as constants, in addition to the identifiers of proper axioms, also combinators. To emphasize that no combinators occur in a D-term, we call it a *pure D-term*. To emphasize that combinators are permitted, we call it *CL-term*, with CL suggesting *combinatory logic*. If only combinators, in contrast to identifiers of proper axioms, occur as constants, we speak of a *pure CL-term*. For determining the MGT, a combinator is just considered as identifier of an associated axiom, determined as the principal type of the combinator.[5] We can now express (vii) as the CL-term

$$\mathbf{B}(\mathbf{B}22)(\mathbf{B}22)(\mathbf{B}(\mathbf{B}22)(\mathbf{B}22)1). \tag{x}$$

By exhaustively rewriting (x) with (ix) we obtain the pure D-term (vii). We call the CL-term (x) a *combinatory compression* of (vii). Speaking of *compression* is justified, since the (x) has with 6 a smaller compacted size than (vii), whose compacted size is 8. Figure 3 shows the minimal DAG of (x). As list of factors it can be written as

$$[3 = \mathbf{B}22,\ 4 = \mathbf{B}33,\ 5 = 4(41)]. \tag{xi}$$

Figure 3: D-term (x) as DAG.



[4]This example, already sketched in the introduction, is from [35, Sects. 2.10 and 3.7], where it is used to explicate the connection structure calculus. It was chosen here to indicate parallels to the connection structure calculus.

[5]For examples, see Table 5 below. In the literature there are different approaches to formally deal with the relationship of combinators and associated implicational formulas: In [28, Table 3E2a] the formulas are the *principal types* of the combinators or equivalent λ-terms. In [24, Sect. 9D] they are called *functional characters of combinators*. In [36, Chap. 6] they are used to specify operational models for an algebraic view on combinators. In [37] the formulas are taken as starting point.

While $\mathbf{B}$ can be eliminated completely from the combinatory compressed form (x) by rewriting with (ix), the rewrite rule can not be applied in any of the individual factors shown in (xi), because $\mathbf{B}$ occurs in these only with two arguments. The construction of (x) can be generalized to proofs of $\mathsf{P}(\mathsf{f}^{2^n}(\mathsf{a}))$ for arbitrary $n \geq 1$. While the pure D-term then always has compacted size 2^n, the compacted size of the CL-term with $\mathbf{B}$ is just $2n$. For example, for $n = 4$, to prove $\mathsf{P}(\mathsf{f}^{16}(\mathsf{a}))$, the minimal DAG is $[3 = \mathbf{B}22,\ 4 = \mathbf{B}33,\ 5 = \mathbf{B}44,\ 6 = 5(51)]$.

2.3. Proof Search with Compressed Combinatory Proof Structures

Goal-driven clausal tableau provers enumerate proof structures, tableaux, interwoven with unification of formulas associated with nodes of the structures. These structures may be viewed as terms, pure D-terms in the setting of condensed detachment. Now, like tableaux or pure D-terms, also CL-terms can be enumerated as basis for first-order proof search. If we add the combinator $\mathbf{B}$ as axiom identifier that designates its principal type (shown in Table 5 below) to our proper axioms from Table 1, let $\mathsf{P}(\mathsf{f}^8(\mathsf{a}))$ be the goal, and enumerate D-terms (or, more precisely, CL-terms, since now $\mathbf{B}$ is permitted) by iterative deepening upon compacted size, we find no proofs with compacted size less than 6 and six proofs with compacted size 6, specifically:

$$\begin{array}{ll}
\text{(a)} & [3 = \mathbf{B}22,\ 4 = 3(3(3(31)))]. \\
\text{(b)} & [3 = \mathbf{B}2,\ 4 = 32,\ 5 = 34,\ 6 = 4(5(51))]. \\
\text{(c)} & [3 = \mathbf{B}2,\ 4 = 32,\ 5 = 34,\ 6 = 5(5(41))]. \\
\text{(d)} & [3 = \mathbf{B}2,\ 4 = 32,\ 5 = 34,\ 6 = 5(4(51))]. \\
\text{(e)} & [3 = \mathbf{B}2,\ 4 = 3(3(32)),\ 5 = 4(41)]. \\
\text{(f)} & [3 = \mathbf{B}22,\ 4 = \mathbf{B}33,\ 5 = 4(41)].
\end{array} \qquad \text{(xii)}$$

CL-term (f) was shown above as (x). All six CL-terms reduce with the rule (ix) for $\mathbf{B}$ to the same normal form, (vii) and thus represent just different compressions of the same "expanded" proof, the pure D-term (vii). For CL-terms (a) and (f) the factors involve applications of $\mathbf{B}$ to two arguments, and (b)–(e) to a single argument. Since the rewrite rule for $\mathbf{B}$ is only applicable in presence of three arguments, it fails to rewrite any individual factor of these CL-terms. Observing for the example that whether $\mathbf{B}$ is applied in factors to one or two arguments has no influence on the minimal proof size, we wish to restrict the search space by constraining permitted occurrences of $\mathbf{B}$ to the two-argument cases. This can be achieved by using combinators not directly for enumeration, but via parameterized *proof schemas*, whose semantics is specified by a parameterized CL-term. Table 2 gives two examples.

A proof schema is determined by a *schema*, a pattern for use in the construction of proof terms and a *defining CL-term* that specifies a semantics. We call proof terms that are built-up from such schemas (and possibly also application, combinators and identifiers of proper axioms) *PS-terms*. The λ-term in Table 2 is equivalent to the defining CL-term and provides an alternate, sometimes more intuitive, way to specify the semantics. The

Table 2
Proof schemas for use with the axioms from Table 1 and goals $\mathsf{P}(\mathsf{f}^n(\mathsf{a}))$.

Schema	*CL-term*	*λ-term*	*Resolution-like view*
$\mathsf{r}_0(p,q)$	pq	pq	$A{\rightarrow}B,\ D\ \vdash\ B\mathsf{mgu}(\{A,D\})$
$\mathsf{r}_1(p,q)$	$\mathbf{B}pq$	$\lambda x.p(qx)$	$A{\rightarrow}B,\ C{\rightarrow}D\ \vdash\ (C{\rightarrow}B)\mathsf{mgu}(\{A,D\})$

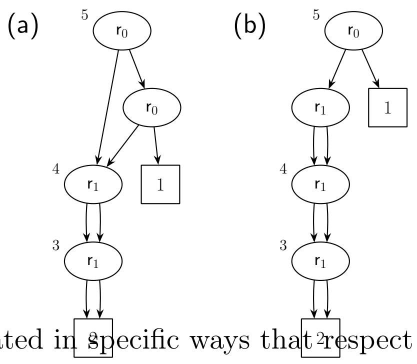
table also shows the effect of the proof schema in terms of a binary resolution step between Horn clauses, indicating a correspondence of proof schemas to restricted forms of lemma computation by resolution.

By enumerating PS-terms constructed from the schemas in Table 2 and the identifiers $1, 2$ of proper axioms in Table 1 with iterative deepening upon compacted size we find the following two proofs with least compacted size, which is 4, shown also in Fig. 4.

$$\begin{array}{ll} \text{(a)} & [3 = \mathsf{r}_1(2,2),\ 4 = \mathsf{r}_1(3,3),\ 5 = \mathsf{r}_0(4, \mathsf{r}_0(4,1))]. \\ \text{(b)} & [3 = \mathsf{r}_1(2,2),\ 4 = \mathsf{r}_1(3,3),\ 5 = \mathsf{r}_0(\mathsf{r}_1(4,4),1)]. \end{array} \qquad \text{(xiii)}$$

By rewriting schema occurrences in the PS-terms of (xiii) to their defining CL-term, i.e., by rules $r_0(p,q) \rightarrow_{\text{rew}} pq$ and $r_1(p,q) \rightarrow_{\text{rew}} \mathbf{B}pq$, we obtain (xii).(a) for (xiii).(a) and (xii).(f) for (xiii).(b). In general, rewriting schema occurrences with their defining CL-terms can be applied individually on each factor, increasing the compacted size only linearly (if duplicate occurrences of schema parameters are not allowed in defining CL-terms) and resulting in a CL-term without heuristically motivated proof schemas, but only combinators in addition to constants representing proper axioms.

Figure 4: Proofs (xiii).(a) and (xiii).(b).



In the proofs (xiii), the schemas occur only instantiated in specific ways that respect an associated arity, which, in the resolution-like view is the number of antecedents of the involved Horn clauses. For example, for $\mathsf{r}_0(p,q)$, the p argument is a term with r_1 as outermost function symbol and the q argument is either axiom 1 or a term with r_0 as outermost function symbol. To make use of this restriction in proof search, the specification of schemas can be supplemented with *arity types* of parameters and values. Also with the axioms and the goal such an arity type is then associated, as shown in Table 3 for our running example, generalized to goals $\mathsf{P}(\mathsf{f}^n(\mathsf{a}))$ for $n \geq 0$. For these problems, the specified arity typing has no effect on the found proofs, but lessens the search effort. In Sect. 3.3 we will see a second purpose of arity types, to disambiguate between object- and meta-level for condensed detachment problems where implicational formulas are reified.

2.4. Practical Realization: The *CCS* Prover as a Component of *CD Tools*

The approach is realized as a component called *CCS* (for *Compressed Combinatory Structures*) of *CD Tools* [31], a Prolog environment for experimenting with first-order ATP where emphasis is on proof structures like D-terms as data objects, which in turn is

Table 3
Axiom, schema and goal specifications supplemented by arity types.

Axiom ID	*Axiom formula*	*Schema*	*Defining CL-term*	*Goal*
1:0	$\mathsf{P}(\mathsf{a})$	$\mathsf{r}_0(p{:}1, q{:}0){:}0$	pq	$\mathsf{P}(\mathsf{f}^n(\mathsf{a})){:}0$
2:1	$\mathsf{P}(x) \to \mathsf{P}(\mathsf{f}(x))$	$\mathsf{r}_1(p{:}1, q{:}1){:}1$	$\mathbf{B}pq$	

embedded in *PIE* [4, 38]. *CD Tools* and *PIE* are free software and run in *SWI-Prolog* [39]. The starting point of *CD Tools* is condensed detachment as a specialization of the connection method [22], providing a simplified variant of first-order ATP that still has many of its essential characteristics and is suitable as basis for the development and study of new techniques. Condensed detachment has dedicated applications [30, 40, 41], reflected in about 200 *TPTP* problems which we so far used in experiments, and can more generally be used as inference rule for first-order Horn problems. *CCS* currently supports enumeration of proof structures of or up to a given compacted size, i.e., DAG enumeration, complementing the *SGCD* [31] prover of *CD Tools*, which enumerates by tree-oriented measures. Of course, unification is woven into the enumeration, such that only proofs of a given goal or, if the goal is unspecified, proofs of lemmas derivable from the axioms are enumerated.

The DAG enumeration method of *CCS* keeps a list of proofs of solved subgoals that are subproofs of the proof under construction. The list is subject to backtracking. A subgoal is solved by first trying the proofs in the list, which corresponds in case of success to attaching a new incoming edge to the DAG node representing the subproof. Then proofs of the subgoal are newly computed and discarded if they already appear in the list. This is slightly different from the *value-number method* [42, 43] for DAG construction, where there is no first step of trying to find a tree in the list and a newly computed tree that is equal to a tree in the list is not discarded but identified with the tree in the list (gets the same "number"). For us, that method seems not suitable, because adequate size restrictions for the newly computed trees are then not available when they are created.

The basic user input to *CCS* is a problem specification along with a specification of the elements to be used for proof structure construction. The problem specification determines a set of Horn clauses as proper axioms and (optionally) a goal atom.[6] It can be provided in various formats, e.g., as identifier of a *TPTP* problem, as file in *TPTP* format, or as implication in *PIE*'s formula syntax. Some formats are subjected to preprocessing, e.g., normal form transformation or conversions to handle a non-atomic goal. For problems that are given as condensed detachment problems, in contrast to general Horn problems, a special translation is available. The elements for proof structure construction are in the simplest "vanilla" case just the binary D function symbol and constants for the proper axioms, sufficient to find pure D-term proofs with minimal compacted size. To make use of compression, combinators and proof schemas can be specified, where a proof schema is characterized by a pattern, optionally with arity types for its variables and result, together with a CL-term or λ-term that specifies its semantics.

To keep the interplay between proper axioms, specification of proof term constructors

[6]It possible to run *CCS* without given goal to generate lemmas.

and handling of lemmas with subproofs for DAG construction manageable, flexibly configurable and efficient, *CCS* is realized as a compiler that, like *PTTP* [44], generates Prolog code. *CD Tools* further includes supplementary functionality for experimenting with the approach, including graph-based normal form conversion for CL-terms and an interface to *TreeRePair* [45], an external tool for advanced tree compression.

3. First Experiments

The following three subsections each describe a series of experiments based on the corpus *TPTPCD2*, which comprises the 196 condensed detachment problems in *TPTP 7.5.0* that remain after excluding from all 206 condensed detachment problems 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.[7] References to the *rating* of a problem refer to the latest rating in *TPTP 7.5.0*. While some of the experiments include proofs for about 25 problems rated between 0.25 and 0.50, thus comparing to results of state-of-the art solvers, others explore what can be reached by exhaustive search upon compacted size, which seems useful, e.g., to find guaranteed shortest proofs, to gain an overview on redundancies in the search space, and as a basis for refinements. For the latter type of experiments the range is around 80–90 proven problems, which actually is at the top of what is known for clausal tableau provers.[8]

All results were obtained on a HPC system with *Intel® Xeon® Platinum 9242 @ 2.30GHz* CPUs, 3.7 GB memory per CPU and 2,400 s time limit per prover run and problem. In their current versions the used provers *CCS* and *SGCD* do not support parallelism or portfolio configurations. If adequate, we consider results of several configurations joined together, picking the best result for each problem, as if obtained in a parallel run. For such combined configurations, each individual configuration was given the 2,400 s time limit.

The *CD Tools* web page shows detailed tables with links to the individual *TPTP* problems and to graphical proof structure presentations. It also archives downloadable logs that contain the proofs from the experiments in Prolog-readable form.

3.1. Pure D-Term Proofs with Minimal Compacted Size

CCS was applied in "vanilla" configurations to determine by exhaustive search as a far as possible for each problem in the corpus the minimal compacted size of a pure D-term proof and, moreover, the number of different such proofs of that size. The results

[7]The restriction on detachment and goals simplifies the problem form and was used in previous experiments [31]. The eight problems that do not match it are suitable as inputs of *CCS* in a mode for general Horn problems.

[8]*leanCoP 2.1* [1] proves 50 problems in the setting of the experiments reported here. The *ProblemAndSolutionStatistics* document of *TPTP 8.0.0* reports the following numbers of proved problems: *SETHEO 3.3* [46]: 65; *S-SETHEO* [47]: 66; *lazyCoP 0.1* [18]: 42; *SATCoP 0.1* [19]: 59; all four together: 76. *CMProver* [4] in various configurations proves 89 problems (http://cs.christophwernhard.com/pie/cmprover/evaluation_201803/tptp_neq.html). All these systems together prove 92 problems.

give an overview on the range of such exhaustive search and provide guiding values for comparison with compressed proofs. Table 4 aggregates the results. *CCS* was used in two configurations that differed in the order in which proper axioms and D are considered at enumeration. The table shows in each row for a set of problems its cardinality and, aggregated as minimum, maximum, average and median over the set, the number of proper axioms (not counting the detachment clause and the negated goal clause), the minimal compacted size of a pure D-term proof, and the number of different pure D-term proofs with that size. The problem sets are as follows: *TPTPCD2* is the corpus of considered problems, *MC* is the set of problems in the corpus for which *CCS* (in at least one of the two configurations) found the minimal compacted size, *MC all* is the subset of *MC* for which *CCS* could determine the number of all proofs of that size, and *Not MC* is *TPTPCD2* without the members of *MC*. For these just lower size bounds could be determined. Three problems in *MC* and two in *MC all* are rated 0.25, the remaining ones in *MC* are rated 0.00.

Table 4
Determined minimal compacted size and number of different proofs of that size for *TPTPCD2*.

Problem set	*#*	*#Axioms*				*Min. comp. size*				*#Proofs of min. comp. size*			
		min	max	avg	med	min	max	avg	med	min	max	avg	med
TPTPCD2	196	1	5	2.46	3								
MC	86	1	5	2.33	2	1	12	6.41	7				
MC all	79	1	5	2.30	2	1	12	6.09	6	1	17,280	368.08	3
Not MC	110	1	4	2.56	3	$\geq$8	$\geq$12	$\geq$9.31	$\geq$9				

3.2. Combinatory Compression of Given Proofs

In this experiment we first converted given proofs of problems from the corpus to compressed tree grammars, with an advanced tool originally targeted at compressing XML trees, and then translated the grammars into CL-terms, via λ-terms and an optimized technique from the implementation of functional programming languages. Specifically, the grammar compression was obtained with *TreeRePair* [45] with option *-optimize edges*. It compresses the given binary tree representation of the proof into an acyclic context-free grammar with exactly one production for each nonterminal. The nonterminals may have parameters, which have only a single occurrence in the right-hand side (i.e., the grammars are *linear*). We translated the productions first into λ-terms and then into CL-terms by the optimized method from [25, Chap. 16], followed by some graph-based simplification. Table 5 shows all combinators considered there.

Our objectives with this experiment were to get an idea about what can be achieved for proofs stemming from applications with compression techniques and to see which combinators emerge, as potential guidance for selecting combinators in proof search. We were interested in the "real" potential of compression, rather than just effects on redundancies that are trivial or better to detect with other means. Hence the given proofs should already be small. For this purpose, they were obtained by *SGCD* [31] and *CCS*.

Table 5
Considered combinators with defining λ-term and axiom formula (principal type). For some, an alternate definition in terms of other combinators is given. The 4-ary combinators $\mathbf{S_4}$, $\mathbf{B_4}$, $\mathbf{C_4}$ appear as $\mathbf{S'}$, $\mathbf{B^*}$,$\mathbf{C'}$ in [25], which clashes with other uses of these names in the literature. **BB** has no widespread name.

	λ-Term	Principal Type	Alt.
$\mathbf{S}$	$\lambda xyz\,.\,xz(yz)$	$(p \to (q \to r)) \to ((p \to q) \to (p \to r))$	
$\mathbf{K}$	$\lambda xy\,.\,x$	$p \to (q \to p)$	
$\mathbf{I}$	$\lambda x\,.\,x$	$p \to p$	
$\mathbf{B}$	$\lambda xyz\,.\,x(yz)$	$(p \to q) \to ((r \to p) \to (r \to q))$	
$\mathbf{C}$	$\lambda xyz\,.\,xzy$	$(p \to (q \to r)) \to (q \to (p \to r))$	
$\mathbf{S_4}$	$\lambda xyzu.x(yu)(zu)$	$(p \to (q \to r)) \to ((s \to p) \to ((s \to q) \to (s \to r)))$	
$\mathbf{B_4}$	$\lambda xyzu.x(y(zu))$	$(p \to q) \to ((r \to p) \to ((s \to r) \to (s \to q)))$	
$\mathbf{C_4}$	$\lambda xyzu.x(yu)z$	$(p \to (q \to r)) \to ((s \to p) \to (q \to (s \to r)))$	
$\mathbf{I'}$	$\lambda xy\,.\,yx$	$p \to ((p \to q) \to q)$	**CI**
$\mathbf{B'}$	$\lambda xyz\,.\,y(xz)$	$(p \to q) \to ((q \to r) \to (p \to r)))$	**CB**
$\mathbf{C^*}$	$\lambda xyz\,.\,yzx$	$p \to ((q \to (p \to r)) \to (q \to r)))$	**CC**
$\mathbf{B''}$	$\lambda xyzu\,.\,xy(zu)$	$(p \to (q \to r)) \to (p \to ((s \to q) \to (s \to r)))$	**BB**

Proofs for 167 problems were found in settings where the size of the CL-term translation is recorded and only a single proof with the smallest CL-translation that was found before the timeout is kept. These proofs were supplemented by further proofs from previous experiments with *SGCD* and *CCS* to form a basis of 217 proofs for 176 problems of the corpus, including 25 rated larger than 0.00, with 5 rated 0.50, but none 1.00. Table 6 shows the combinators, or, more precisely, *maximal pure CL-terms*, i.e., pure CL-terms in occurrences not as strict subterm of another pure CL-term, in the CL-translations obtained in the experiment. Note that **S** and $\mathbf{S_4}$, whose defining λ-term has a variable with two occurrences in its body, do not appear in the proofs, because the underlying grammars were linear.

Table 6
Maximal pure CL-terms occurring in the CL compressed proofs, with the number of proofs in which they occur (among 132 proofs for 86 problems in which combinators occur in the CL compression).

$\mathbf{I'}$	42	$\mathbf{C}$	28	$\mathbf{C_4}$	5	$\mathbf{C^*}$	2	$\mathbf{B_4B_4B}$	2	$\mathbf{C_4C}$	1
$\mathbf{B}$	37	$\mathbf{B_4}$	9	$\mathbf{B'}$	2	$\mathbf{B_4BB}$	2	$\mathbf{C_4B}$	1	$\mathbf{C_4BB}$	1

Tables 7 and 8 summarize the achieved compression effects, where we use the following shorthands for size measures of a proof in its different transformations: the originally given pure D-term, the compressed tree grammar and the compressed CL-term.

LC: Compacted size of the compressed CL-term, where maximal pure CL-terms are valued like constants, reflecting that a pure CL-term may be expressed by a dedicated combinator.

XC: Compacted size of the given pure D-term.

GS: Size of the compressed tree grammar, determined as the sum over the number of edges in all its right-hand sides. (If the grammar has only parameter-free nonterminals, it represents a DAG. *GS* is then the doubled compacted size of the represented term.)

Table 7
Problems with given proofs on which compression to CL-terms and tree grammars had reducing effect.

Problem set	*#*	*LC<XC*	*CL-term has comb.*	*GS<XC∗2*	*Gr. has param.*
TPTPCD2	196				
Proven	176	29	86	110	133
MC	87	1	16	32	45
TPTPCD2, Rtg>0.00	45				
Proven, Rtg>0.00	25	11	21	22	24

Table 8
Compression ratios for problems where the compression is strictly smaller than the source.

Problem set	*#*	min	max	avg	med
Proven, LC<XC	29	1.03	1.17	1.09	1.10
Proven, LC<XC, Rtg>0.00	11	1.03	1.12	1.08	1.07
Proven, GS<XC∗2	110	1.02	1.83	1.26	1.21
Proven, GS<XC∗2, Rtg>0.00	22	1.02	1.56	1.32	1.38

Table 7 shows in each row for a set of problems its cardinality, the number of members with $LC{<}XC$, the number of members where a combinator occurs in the compressed CL-term, the number of members with $GS{<}XC{*}2$, i.e., where the grammar compression is smaller than the DAG considered as grammar, and the number of members where the grammar has a parameterized nonterminal, indicating that the grammar compression did not effect just DAG compression. (The number of parameters of nonterminals in all obtained grammars is between 0 and 3.) The problem sets are as follows: *TPTPCD2* is the corpus of considered problems. *Proven* is the set of problems for which a proof was available. *MC* is the set of problems for which the minimal compacted size of a pure D-term proof is known.[9] *TPTPCD2 Rtg>0.00* and *Proven Rtg>0.00* are like *TPTPCD2* and *Proven*, but restricted to problems rated larger than 0.00. Apparently compressions have on higher rated problems a stronger effect, in particular compared to those where the minimal compacted size of pure D-terms can be determined. A size reduction achieved by the grammar compression (e.g., for 110 of the 176 problems in *Proven*) is not always reflected in the introduction of a combinator (only for 86 problems in *Proven*) and a size reduction of the CL-term through combinators (only for 29 of these 86 problems). This needs further investigation. It may be due to a linear size increase in the grammar to combinator translation and the way combinators themselves are counted. Table 8 shows in each row for a set of problems its cardinality and compression ratios, aggregated as minimum, maximum, average and median over the set. The problem sets are as follows: *Proven* is the set of problems for which a proof is available and it holds that $LC{<}XC$, i.e., the combinatory compression has reducing effect. The considered compression ratio

[9] Compared to the set *MC* of Table 4 it contains one more problem, where the techniques considered there could ascertain only a lower bound of the compacted size and a proof of that size was found with another technique.

for this set is it XC/LC. *Proven, GS<XC∗2* is the set of problems for which a proof is available and it holds that $GS<XC{*}2$, i.e., the grammar compression taken as basis for the combinatory compression has reducing effect. The considered compression ratio is $XC{*}2/GS$. *Proven, LC<XC, Rtg>0.00* and *Proven, GS<XC∗2, Rtg>0.00* are like the other sets, but restricted to problems rated larger than 0.00. The modest compression ratios do not come as a surprise since the problems are not constructed challenges but stem from applications that where previously addressed by ATP systems. As discussed in the context of Table 7, the reducing effects of the grammar compression are not in full reflected in the combinatory translation, which needs further investigation.

3.3. Exhaustive Search with Proof Schemas Involving Combinators

CCS was applied with combinator-based proof schemas and exhaustive enumeration of PS-terms by iterative deepening upon compacted size. The main objective was to see the basic effects of proof search with compressed combinatory structures for problems from applications. Does the implicitly incorporated search for shortest compressions obstruct proof search or does it increase success through the shorter structures? The used configurations of *CCS* were characterized by subsets of the schemas from Table 9. Arity type 0 is used for proper axioms and the goal. $\mathbf{I}$ with arity type 2 together with r_1 expresses detachment: A schema instance $\mathsf{r}_1(\mathbf{I}, p, q)$ expands into $\mathbf{I}pq$, which reduces into pq, or $\mathsf{D}(p, q)$. Similarly, detachment can be expressed with $\mathbf{I}$, r_0 and r_2.

Table 9
Considered proof schemas.

Schema	CL-term	λ-term	Resolution-like view
$\mathbf{I}$:2	$\mathbf{I}$	$\lambda xy.xy$	$\vdash (A{\to}B){\to}(A{\to}B)$
$\mathsf{r}_0(p{:}1, q{:}0){:}0$	pq	pq	$A{\to}B,\ D\ \vdash\ B\mathsf{mgu}(\{A, D\})$
$\mathsf{r}_1(p{:}2, q{:}0, r{:}0){:}0$	pqr	pqr	$A_1{\to}(A_2{\to}B),\ D_1,\ D_2\ \vdash\ B\mathsf{mgu}(\{\{A_1, D_1\}, \{A_2, D_2\}\})$
$\mathsf{r}_2(p{:}2, q{:}0){:}1$	pq	$\lambda x.pqx$	$A_1{\to}(A_2{\to}B),\ D\ \vdash\ (A_2{\to}B)\mathsf{mgu}(\{A_1, D\})$
$\mathsf{r}_3(p{:}2, q{:}0){:}1$	$\mathbf{C}pq$	$\lambda x.pxq$	$A_1{\to}(A_2{\to}B),\ D\ \vdash\ (A_1{\to}B)\mathsf{mgu}(\{A_2, D\})$
$\mathsf{r}_4(p{:}1, q{:}1){:}1$	$\mathbf{B}pq$	$\lambda x.p(qx)$	$A{\to}B,\ C{\to}D\ \vdash\ (C{\to}B)\mathsf{mgu}(\{A, D\})$
$\mathsf{r}_5(p{:}2, q{:}1){:}2$	$\mathbf{B}pq$	$\lambda xy.p(qx)y$	$A_1{\to}(A_2{\to}B),\ C{\to}D\ \vdash\ (C{\to}(A_2{\to}B))\mathsf{mgu}(\{A_1, D\})$
$\mathsf{r}_6(p{:}2, q{:}1){:}2$	$\mathbf{B}(\mathbf{C}p)q$	$\lambda xy.py(qx)$	$A_1{\to}(A_2{\to}B),\ C{\to}D\ \vdash\ (C{\to}(A_1{\to}B))\mathsf{mgu}(\{A_2, D\})$

The enumerated proof structures are PS-terms built-up from constants for proper axioms and the configured schemas. Finally, they were subjected to a simplification where schemas whose definition involves no combinator are rewritten into their definiens, which often leads to a slight size reduction.[10] The final PS-term then may also contain occurrences of application as binary symbol. Tables 10– 12 summarize the outcomes, where we use the following shorthands for size measures of a given proof expressed as PS-term.

SC: Compacted size of the PS-term, i.e., the number of inner nodes of its minimal DAG.

[10]With the rules $\mathsf{r}_0(p, q) \to_{\text{rew}} pq$, $\mathsf{r}_2(p, q) \to_{\text{rew}} pq$, $\mathsf{r}_1(i, p, q) \to_{\text{rew}} pq$, $\mathbf{I}p \to_{\text{rew}} p$. This induces a slight mismatch between the assessed compacted size during enumeration and that of the final proof.

XC: Compacted size of the pure D-term obtained from the PS-term by expanding schemas and rewriting combinators.

MC: Minimal compacted size of a pure D-term that proves the same problem as the given proof, if known.

Table 10 shows in each row for a configuration of *CCS* the number of problems that could be proven, the number of problems with a proof in which r_3, r_4, r_5, or r_6 (i.e., a schema involving a combinator other than **I**) occurs, and two compression ratios, aggregated as minimum, maximum, average and median. The first is the ratio *XC*/*SC* for the proofs involving r_3, r_4, r_5, or r_6. Then the number of problems in this set for which in addition *MC* is known is shown, followed by the ratio *MC*/*SC* upon that set. The considered prover configurations are as follows: *S1*, *S2* and *S3* enumerate upon compacted size, building-up PS-terms from the indicated schemas, **I** and identifiers of proper axioms. *S1+S2+S3* combines the best results from *S1*, *S2* and *S3*, ordered by *SC* and required run time. *M* combines the two "vanilla" configurations described in Sect. 3.1 and is included for comparison. All configurations together can prove 92 problems, 89 rated 0.0 and 3 rated 0.25, proved by both, *S1+S2+S3* and *M*.

Table 10
Number of proven problems, proofs with combinators, and compression ratios.

CCS Configuration	*#Proven*	*#Cmb.*	*XC/SC*				*#Cmb.*	*MC/SC*			
			min	*max*	*avg*	*med*	$\cap$*M*	*min*	*max*	*avg*	*med*
S1+S2+S3	88	25	1.11	1.64	1.37	1.33	19	1.11	1.40	1.28	1.29
S1 ($\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_4, \mathsf{r}_5, \mathsf{r}_6$)	86	20	1.20	1.63	1.37	1.33	16	1.20	1.40	1.30	1.31
S2 ($\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3, \mathsf{r}_4, \mathsf{r}_5, \mathsf{r}_6$)	83	20	1.11	1.57	1.31	1.33	19	1.11	1.40	1.26	1.29
S3 ($\mathsf{r}_0, \mathsf{r}_2, \mathsf{r}_4, \mathsf{r}_6$)	64	30	0.71	2.00	1.17	1.15	29	0.67	1.17	0.96	1.00
M	86										
Total	92										

Of the proof schemas r_3–r_6, schema r_4 is by *S1* and *S2* together used only for 3 problems, but by *S3* for 24 problems; r_3 is used by *S2* for 7 problems and r_5 and r_6 are used by each of the configurations where they are specified for 10–16 problems. For *S3* some compression ratios are less than 1.00, indicating that the compacted size of some PS-terms is larger than that of their pure D-term normalization or the minimal compacted size for the problem. It remains to investigate whether this has a substantial reason or can be remedied by some postprocessing simplification.

S1+S2+S3 failed on four problems proven by *M* but proved six problems on which *M* failed, indicating that for the latter the search for compressed structures was beneficial. They are shown in Table 11, with the number of proper axioms, the *CCS* configuration that provided the proof and parameters of the proof. Table 12 shows those problems for which a proof was found by *S1+S2+S3* that satisfies *SC*<*MC*<*XC*, where *SC* is minimal among the proofs by *S1+S2+S3*. The relationship *SC*<*MC*<*XC* indicates that the minimal compacted size of a pure D-term can be undercut by a PS-term whose pure D-term normalization is larger than the minimal one. All problems in Table 12 are hard for conventional tableau provers.[11]

Table 11
Proven by S1+S2+S3 but not by *M*

Problem	*Rating*	*#Ax*	*Config*	*Time*	*SC*	*XC*	*XC/SC*
LCL080-2	0.00	4	*S1*	2,201 s	7	9	1.29
LCL088-1	0.00	1	*S2*	1,029 s	10	12	1.20
LCL090-1	0.00	1	*S3*	1,254 s	11	18	1.64
LCL366-1	0.00	3	*S1*	1,743 s	8	13	1.63
LCL378-1	0.00	3	*S1*	1,717 s	8	13	1.63
LCL399-1	0.00	3	*S1*	1,912 s	8	12	1.50

Table 12
Proofs where *SC*<*MC*<*XC* and *SC* is minimal among the considered proofs.

Problem	*Rating*	*#Ax*	*Config*	*Time*	*SC*	*MC*	*XC*	*MC/SC*	*XC/SC*
LCL089-1	0.00	1	*S2*	85 s	9	10	13	1.11	1.44
LCL129-1	0.00	1	*S1*	48 s	8	11	12	1.38	1.50
LCL364-1	0.00	3	*S1*	70 s	7	9	11	1.29	1.57
LCL092-1	0.25	1	*S2*	357 s	9	12	13	1.33	1.44
LCL365-1	0.25	3	*S1*	270 s	8	10	12	1.25	1.50

4. Conclusion

A new perspective for clausal tableau techniques in first-order ATP has been shown, where proof search is not performed directly by enumerating tableau structures, but by enumerating more succinct combinator terms that normalize into the tableau structures. Restricting combinator terms to configured patterns provides a proof schema mechanism that constrains proof structures considered at search. Initial experiments indicate feasibility of the approach for problems from applications, and special relevance if the objective is to find particularly short or "good"[12] proofs, or to get an overview on different proofs for a problem.

Of course, it would be very interesting to see the approach with problem series that separate proof systems by their strength. Unfortunately, it seems that, differently from propositional logic, for first-order Horn formulas such problems are hardly available. Our running example was from [21], where it separates conventional connection calculi (including clausal tableaux) on the one hand from resolution and the connection structure

[11] *ProblemAndSolutionStatistics* of *TPTP 8.0.0* does contain for neither of the provers *SETHEO 3.3*, *S-SETHEO*, *lazyCoP 0.1* and *SATCoP 0.1* a success record on these problems. Also *leanCoP 2.1* fails on them with a 2 hour timeout per problem. *CMProver*, in some configurations, can solve LCL364-1, LCL092-1, and LCL365-1.

[12] Aside of measures based directly on size of the proof structure [48, 22, 49], qualities of interest are, for example, short formula size of lemmas; restrictions on the inference rules, e.g., to just condensed detachment [40]; that extracted interpolants are in some specific formula class [50, 9]; or capturing short informal arguments [51].

calculus [52, 21, 35] on the other hand. The presented approach actually emerged from an effort to implement the connection structure calculus, which was never implemented so far. A precise comparison of its labeling techniques with our term-oriented view would, however, need further investigation.

Our proof schemas resemble, at least in part, restricted forms of resolution for Horn clauses. It is indeed straightforward to express inference rules such as hyperresolution or binary resolution for clauses of specific lengths by proof schemas. Determining actual correspondences to known resolution refinements needs further work. More generally, the potential and expressive power of combinator terms as compressed and enumerable proof representations is yet to be explored. Is it possible to capture induction through schemas based on combinators with repeated variables in the defining λ-term? The proof schema mechanism suggests a two-leveled search, some systematic enumeration of sets of permitted schemas and then the actual proof search with these. Also enumeration itself can be addressed, with incomplete ways (e.g., based on the subproof relationship [22, 31]) and driven by learned structure patterns.

The problem of finding a proof with minimal compacted size, considered in Sect. 3.1, is also the subject of [48], where a method based on linked UR-resolution and implemented with *OTTER* [53] is introduced. Apparently, its performance on *TPTP* problems is not known. Grammar compressions are used in first-order proof theory on the formula level [54]. Their use for proof structures seems new. Also the correspondence between grammar compressions and combinator terms seems a new observation. Combinator terms are used for *search* in higher-order unification [55, 56] and a related recent superposition calculus for higher-order logic [57] involves rewrite rules that realize combinator axioms.

So far, the experiments were performed on the condensed detachment problems in the *TPTP*. As indicated with our running example, the approach extends to general first-order Horn problems, which has been already laid out in the implementation. The experiments so far were centered on exhaustive DAG enumeration. On the agenda is to use the combinator approach with features previously explored for a related prover [31], in particular, with blending goal- and axiom-directed search, tree- instead of DAG-oriented enumeration, and heuristic restrictions (e.g., on the size of involved lemma formulas, number of kept lemmas, or by keeping at most a single proof per lemma). Specifying adequate size measures of proof terms may need further investigation: It appears that details which from a complexity point of view play no role show up as confusing when single proofs are considered. Possibly this is a matter of graph-based postprocessing simplifications. From a broader perspective, the approach opens tempting bridges between first-order ATP and neighboring fields, such as type theory [34, 28] and the recent empirical computation theory based on enumerating combinator terms [26].

Acknowledgments

The author thanks Wolfgang Bibel for inspirations and for sharing an idea – more directly based on the connection method – to prove a problem series, which gave impetus to this work. The author also thanks anonymous reviewers for helpful feedback to improve the presentation. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research

Foundation) – Project-ID 457292495. The work was supported by the North-German Supercomputing Alliance (HLRN).

References

[1] J. Otten, Restricting backtracking in connection calculi, AI Communications 23 (2010) 159–182. doi:https://doi.org/10.3233/AIC-2010-0464.

[2] M. E. Stickel, A Prolog technology theorem prover: implementation by an extended Prolog compiler, J. Autom. Reasoning 4 (1988) 353–380. doi:https://doi.org/10.1007/BF00297245.

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

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

[5] 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 Jun 30, 2022.

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

[7] W. Bibel, J. Otten, From Schütte's formal systems to modern automated deduction, in: R. Kahle, M. Rathjen (Eds.), The Legacy of Kurt Schütte, Springer, 2020, pp. 215–249. doi:https://doi.org/10.1007/978-3-030-49424-7_13.

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

[9] A. Hudek, D. Toman, G. Wedell, On enumerating query plans using analytic tableau, in: TABLEAUX 2015, volume 9323 of *LNCS (LNAI)*, Springer, 2015, pp. 339–354.

[10] C. Wernhard, Craig interpolation with clausal first-order tableaux, J. Autom. Reasoning 65 (2021) 647–690. doi:https://doi.org/10.1007/s10817-021-09590-3.

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

[12] J. Otten, MleanCoP: A connection prover for first-order modal logic, in: S. Demri, D. Kapur, C. Weidenbach (Eds.), IJCAR 2014, volume 8562 of *LNCS (LNAI)*, Springer, 2014, pp. 269–276. doi:https://doi.org/10.1007/978-3-319-08587-6_20.

[13] J. Otten, Non-clausal connection-based theorem proving in intuitionistic first-order logic, in: C. Benzmüller, J. Otten (Eds.), ARQNL 2016, volume 1770 of *CEUR Workshop Proc.*, CEUR-WS.org, 2016, pp. 9–20. URL: http://ceur-ws.org/Vol-1770/paper1.pdf.

[14] J. Otten, The nanoCoP 2.0 connection provers for classical, intuitionistic and modal logics, in: A. Das, S. Negri (Eds.), TABLEAUX 2021, volume 12842 of *LNCS (LNAI)*, Springer, 2021, pp. 236–249. doi:https://doi.org/10.1007/978-3-030-86059-2_14.

[15] C. Kaliszyk, J. Urban, FEMaLeCoP: Fairly efficient machine learning connection prover, in: M. Davis, A. Fehnker, A. McIver, A. Voronkov (Eds.), LPAR-20, volume 9450 of *LNCS (LNAI)*, Springer, 2015, pp. 88–96. doi:https://doi.org/10.1007/978-3-662-48899-7_7.

[16] Z. Zombori, J. Urban, C. E. Brown, Prolog technology reinforcement learning prover (system description), in: N. Peltier, V. Sofronie-Stokkermans (Eds.), IJCAR 2020, volume 12167 of *LNCS (LNAI)*, Springer, 2020, pp. 489–507. doi:https://doi.org/10.1007/978-3-030-51054-1_33.

[17] Z. Zombori, J. Urban, M. Olsák, The role of entropy in guiding a connection prover, in: A. Das, S. Negri (Eds.), TABLEAUX 2021, volume 12842 of *LNCS (LNAI)*, Springer, 2021, pp. 218–235. doi:https://doi.org/10.1007/978-3-030-86059-2_13.

[18] M. Rawson, G. Reger, lazyCoP: Lazy paramodulation meets neurally guided search, in: A. Das, S. Negri (Eds.), TABLEAUX 2021, volume 12842 of *LNCS (LNAI)*, Springer, 2021, pp. 187–199. doi:https://doi.org/10.1007/978-3-030-86059-2_11.

[19] M. Rawson, G. Reger, Eliminating models during model elimination, in: A. Das, S. Negri (Eds.), TABLEAUX 2021, volume 12842 of *LNCS (LNAI)*, Springer, 2021, pp. 250–265. doi:https://doi.org/10.1007/978-3-030-86059-2_15.

[20] M. Färber, C. Kaliszyk, J. Urban, Machine learning guidance for connection tableaux, J. Autom. Reasoning 65 (2021) 287–320. doi:https://doi.org/10.1007/s10817-020-09576-7.

[21] E. Eder, Relative Complexities of First Order Calculi, Vieweg, Braunschweig, 1992. doi:https://doi.org/10.1007/978-3-322-84222-0.

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

[23] M. Schönfinkel, Über die Bausteine der mathematischen Logik, Math. Ann. 92 (1924) 305–316. doi:https://doi.org/10.1007/BF01448013.

[24] H. Curry, R. Feys, Combinatory Logic, volume I, North-Holland, 1958.

[25] S. L. Peyton Jones, The Implementation of Functional Programming Languages, Prentice Hall, 1987.

[26] 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.

[27] S. Wolfram, A Bibliography of Combinators, 2021, pp. 325–356. Also available from https://www.wolframcloud.com/obj/sw-writings/Combinators/bibliography.pdf, accessed Jun 30, 2022.

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

[29] G. Sutcliffe, The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0, Journal of Automated Reasoning 59 (2017) 483–502.

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

[31] C. Wernhard, CD Tools – Condensed detachment and structure generating theorem proving (system description), 2022. In preparation, draft available from http://

cs.christophwernhard.com/papers/cdtools.

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

[33] J. A. Kalman, Condensed detachment as a rule of inference, Studia Logica 42 (1983) 443–451. doi:https://doi.org/10.1007/BF01371632.

[34] J. R. Hindley, D. Meredith, Principal type-schemes and condensed detachment, Journal of Symbolic Logic 55 (1990) 90–105. doi:https://doi.org/10.2307/2274956.

[35] W. Bibel, E. Eder, Methods and calculi for deduction, in: D. M. Gabbay, C. J. Hogger, J. A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, volume 1, Oxford Univ. Press, 1993, pp. 67–182.

[36] K. Bimbó, Combinatory Logic: Pure, Applied and Typed, Routledge, 2012.

[37] A. Rezus, On a theorem of Tarski, Libertas Mathematica 2 (1982) 63–97.

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

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

[40] R. Veroff, Challenge problems with condensed detachment, 2011. Online: https://www.cs.unm.edu/~veroff/CD/, accessed Jun 30, 2022.

[41] M. Walsh, B. Fitelson, Answers to some open questions of Ulrich and Meredith (2021). Under review, preprint: http://fitelson.org/walsh.pdf, accessed Jun 30, 2022.

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

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

[44] M. E. Stickel, A Prolog technology theorem prover: A new exposition and implementation in Prolog, Theor. Comput. Sci. 104 (1992) 109–128. doi:https://doi.org/10.1016/0304-3975(92)90168-F.

[45] M. Lohrey, S. Maneth, R. Mennicke, XML tree structure compression using RePair, Inf. Syst. 38 (2013) 1150–1167. doi:https://doi.org/10.1016/j.is.2013.06.006, system available from https://github.com/dc0d32/TreeRePair, accessed Jun 30, 2022.

[46] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, K. Mayr, SETHEO and E-SETHEO – the CADE-13 systems, J. Autom. Reasoning 18 (1997) 237–246. doi:https://doi.org/10.1023/A:1005808119103.

[47] R. Letz, G. Stenz, Model elimination and connection tableau procedures, in: A. Robinson, A. Voronkov (Eds.), Handb. of Autom. Reasoning, volume 1, Elsevier, 2001, pp. 2015–2114.

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

[49] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, A. Kovtunova, Finding good proofs for description logic entailments using recursive quality measures, in:

A. Platzer, G. Sutcliffe (Eds.), CADE 28, volume 12699 of *LNCS (LNAI)*, Springer, 2021, pp. 291–308. doi:https://doi.org/10.1007/978-3-030-79876-5_17.

[50] M. Benedikt, J. Leblay, B. ten Cate, E. Tsamoura, Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation, Morgan & Claypool, 2016.

[51] M. J. H. Heule, B. Kiesl, A. Biere, Clausal proofs of mutilated chessboards, in: J. M. Badger, K. Y. Rozier (Eds.), NFM 2019, volume 11460 of *LNCS*, Springer, 2019, pp. 204–210. doi:https://doi.org/10.1007/978-3-030-20652-9_13.

[52] 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, M. M. Richter (Eds.), CSL '88, volume 385 of *LNCS*, Springer, 1989, pp. 80–98. doi:https://doi.org/10.1007/BFb0026296.

[53] 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 Jun 30, 2022.

[54] S. Hetzl, Applying tree languages in proof theory, in: LATA 2012, volume 7183 of *LNCS*, 2012, pp. 301–312. doi:https://doi.org/10.1007/978-3-642-28332-1_26.

[55] D. J. Dougherty, Higher-order unification via combinators, Theor. Comput. Sci. 114 (1993) 273–298. doi:https://doi.org/10.1016/0304-3975(93)90075-5.

[56] A. Bhayat, G. Reger, Restricted combinatory unification, in: P. Fontaine (Ed.), CADE 27, number 11716 in LNAI, Springer, 2019, pp. 74–93. doi:https://doi.org/10.1007/978-3-030-29436-6_5.

[57] A. Bhayat, G. Reger, A combinator-based superposition calculus for higher-order logic, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), IJCAR 2020, volume 12166 of *LNCS*, Springer, 2020, pp. 278–296. doi:https://doi.org/10.1007/978-3-030-51074-9_16.