CD Tools – Condensed Detachment and Structure Generating Theorem Proving (System Description)

Christoph Wernhard
University of Potsdam, Germany

Abstract
CD Tools is a Prolog library for experimenting with condensed detachment in first-order ATP, which puts a recent formal view centered around proof structures into practice. From the viewpoint of first-order ATP, condensed detachment offers a setting that is relatively simple but with essential features and serious applications, making it attractive as a basis for developing and evaluating novel techniques. CD Tools includes specialized provers based on the enumeration of proof structures. We focus here on one of these, SGCD, which permits to blend goal- and axiom-driven proof search in particularly flexible ways. In purely goal-driven configurations it acts similarly to a prover of the clausal tableaux or connection method family. In blended configurations its performance is much stronger, close to state-of-the-art provers, while emitting relatively short proofs. Experiments show characteristics and application possibilities of the structure generating approach realized by that prover. For a historic problem often studied in ATP it produced a new proof that is much shorter than any known one.

Keywords
clausal tableaux, combining goal- and axiom-driven proof search, condensed detachment, connection method, finding short proofs, first-order ATP, lemmas, Prolog, proof compression, proof structures

1. Introduction
We present CD Tools, a Prolog library for experimenting with condensed detachment (CD) [36] in automated theorem proving (ATP). CD is basically a framework for first-order reasoning about propositional logics, introduced by Carew A. Meredith [28]. Its inference rule, intuitively modus ponens with unification, may be described as positive hyperresolution with the single non-unit clause

\[
\text{Thm}(y) \leftarrow \text{Thm}(x \Rightarrow y) \land \text{Thm}(x). \tag{Det}
\]

The propositional formulas to reason about are represented there by terms, with \( \Rightarrow \) as function symbol for implication, wrapped in the unary predicate \( \text{Thm} \).\(^1\) In an application of \( \text{(Det)} \), \( \text{Thm}(y) \) is called conclusion, \( \text{Thm}(x \Rightarrow y) \) major premise and \( \text{Thm}(x) \) minor premise. A CD proof is a full binary tree or term (term now understood not as formula constituent but rather on the meta or proof-structure level) such that for inner nodes \( D(d_1, d_2) \) the arguments \( d_1 \) and \( d_2 \) are the subproofs of the major and minor premise, and leaves or constants are labels of proper axioms of the considered propositional object logics, unit clauses, e.g., \( \text{Thm}(p \Rightarrow (q \Rightarrow p)) \), \( \text{Thm}(\neg p) \lor \text{Thm}(\neg q) \lor \text{Thm}(q) \lor \text{Thm}(p) \).

---

1General first-order proving for Horn problems is possible with very similar techniques. For examples, see [42] and http://cs.christophwernhard.com/cdtools/exp-ccs-2022-06/table_4.html.

2Our presentation of the clause \( \text{(Det)} \) is oriented at the proof structures considered with CD, discussed below. In a more conventional ATP-oriented form, with the function symbol for implication in prefix notation, \( \text{(Det)} \) might be written as \( \neg \text{Thm}(x) \lor \neg \text{Thm}(\text{implies}(x, y)) \lor \text{Thm}(y) \).
where \( p, q \) are variables. We call these proof structure representations *D-terms*. See [43] for a precise account.

CD is attractive as a basis for research on first-order ATP because it allows to express application problems, including hard proving problems and questions for deriving theorems that render mathematical intuition, while requiring only a simplified setting of first-order ATP, with a single predicate, a single non-unit clause which is Horn, and no equality. Actually, research with CD belongs to the first and most successful applications of ATP [22] and numerous techniques for OTTER [20] emerged from it [49, 45, 22, 46, 47, 37, 8, 48]. A dedicated system for CD is described in [9]. Prover9 [21] appears to handle inputs that represent a CD problem in a special way by selecting hyperresolution as inference rule such that proofs may be viewed as CD proofs. Recently [43] it was observed that the connection method [3, 5] suggests an ATP-oriented view on CD that is not centered on associating with \((\text{Det})\) an inference rule but on the proof structure, the D-term, as a whole. A D-term represents the structure formed by the connections. A proof consists of a D-term together with a substitution on the terms of conclusion formulas \( \text{Thm}(y) \) associated with each node. Informally, this substitution is the most general one that is constrained by copies of axioms at the leaves and copies of \((\text{Det})\) at the inner nodes.

*CD Tools* is motivated by exploring that view. It provides functionality to inspect, relate and simplify given proofs in various ways and aims to be a practical basis for developing new first-order ATP techniques. It currently included two experimental provers, *CCS* (Compressed Combinatory Proof Structures) [42] and *SGCD* (Structure Generation for Condensed Detachment). *SGCD* is specialized for CD problems and can (in the union of four different configurations) solve 93% of the CD problems in the TPTP [35] that are solvable by any prover at all. Its paradigm may be called *structure generating*, because it basically operates, like goal-driven provers describable in terms of the connection method, model elimination or clausal tableaux [33, 15, 13, 25, 39], by enumerating proof structures, which here means D-terms. A caching or lemma mechanism permits to configure the prover in the space between the extremes of purely goal- and axiom-driven proof search.

The paper is organized as follows: In Sect. 2 we overview the system and specify the considered problem corpus, in Sect. 3 the included prover *SGCD* is described, and in Sect. 4 seven subsections present experiments that indicate characteristics and application possibilities of *SGCD* and its approach. Section 5 concludes the paper. The second included prover, *CCS*, mentioned here on occasion, is described in [42]. The system is available as free software from

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

That website also provides detailed result tables, full output logs, including Prolog-readable proof terms, as well as reproduction instructions for the experiments described in the paper. For some experiments, also graph visualizations of proofs are shown there.
2. Overview on the System

CD Tools is implemented in SWI-Prolog [44], a free well maintained modern Prolog system and uses PIE [39, 40] as basic support for first-order ATP.\(^4\) It comprises about 260 user predicates, modularized in a fine-grained way to keep tight track of dependencies, into 56 modules, which in turn are grouped into 14 sets of modules with related or similar functionality.\(^5\) Prolog is based on unification, Horn clauses, and realizing the execution of a nondeterministic program as an enumeration of proofs through backtracking. These features are quite close to CD and the approach to theorem proving with enumerating proof structures, which is utilized in CD Tools by building on the Prolog-provided unification and means for structure enumeration.

The system provides interfaces to several external worlds: It supports conversion to and from Jan Łukasiewicz’s Polish notation typically used in the literature on applications of CD, where the axiom \(\text{Thm}(p \Rightarrow (q \Rightarrow p))\), in Prolog notation \(\text{thm}(p \Rightarrow (q \Rightarrow p))\), is represented by \(CpCqp\). Predicates are provided to test TPTP problems whether they represent a CD problem, and to canonicalize their vocabulary. TPTP 8.0.0 (and TPTP 7.5.0) contains 206 CD problems, all in the LCL domain. CD Tools includes a definition of a slightly smaller problem corpus called TTPCDT2, comprising the 196 CD problems in TPTP 8.0.0 that remain after excluding 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. The restriction on detachment and goals simplifies the problem form.

Operations on D-terms in CD Tools are based on the formal treatment and concepts from [43]. An important basic way to associate an atomic formula with a D-term is its most general theorem (MGT) (aka principal type scheme [12, 11]) with respect to given axioms. It is the most general (with respect to subsumption) atomic formula that is proven with the D-term for the axioms. Figure 1 shows its implementation in CD Tools. The occurs check is realized by

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{MGT_computation.png}
\caption{Implementation of MGT computation in CD Tools.}
\end{figure}

\begin{verbatim}
d_mgt(DXY, DX, FY) :-
  !,
  d_mgt(DXY, (FX=>FY)),
  d_mgt(DX, FX).
d_mgt(D, F) :-
  id_axiom(D, F),
  acyclic_term(F).
\end{verbatim}

the SWI-Prolog library predicate acyclic_term/1. The given axioms are referenced via the predicate id_axiom/2. If this is defined as \(\text{id\_axiom}(1, P\Rightarrow(Q\Rightarrow P))\), then the query \(?- d_mgt(d(1,1), X)\) succeeds with \(X\) bound to \(P\Rightarrow(Q\Rightarrow(R\Rightarrow Q))\), representing the atomic formula \(\text{Thm}(p \Rightarrow (q \Rightarrow (r \Rightarrow q)))\), where \(p, q, r\) are variables. Another way to use the

---

\(^4\) 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.

\(^5\) See http://cs.christophwernhard.com/cdtools/overview.html.

\(^6\) TPTP 8.0.0 and TPTP 7.5.0 contain the same CD problems, with the same difficulty ratings.
The `d_mgt/2` predicate is to verify that the MGT subsumes a proof goal given as ground term, as for example in `?- d_mgt(d(1,1), p=>((q=>r)=>(p=>(q=>r))))`. The in-place theorem (IPT), where also substitution constraints induced by a context D-term are taken into account is a second important way to associate an atomic formula with a D-term, which is also supported in CD Tools.

The predicate `id_axiom/2` used in the implementation of `d_mgt/2` represents a global mapping from axiom identifiers to axioms that is quietly assumed at experimenting, e.g., when computing the MGT of a given D-term. It is one of the few dynamic (state dependent) predicates of CD Tools and can be [re-]initialized with an interface predicate. A pre-defined association of about 170 common names for about 120 well-known formulas, e.g., from [29, 36], helps to specify these as values of `id_axiom/2` and to detect them among computed lemmas.

CD Tools includes implementations of many concepts and operations introduced or discussed in [43], including D-term comparison by $\geq_c$, various notions of regularity, the organic property (based on calling a SAT solver 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 the experiments MiniSat [7] was used.

D-terms allow precise proof size measures that are supported by CD Tools. We represent here the dimensions of a D-term by a triple

$$\langle c, t, h \rangle$$

that gathers three of its measures: Its compacted size $c$, i.e., the number of inner nodes of the unique minimal DAG representing the tree, its tree size $t$, i.e., the number of inner nodes (occurrences of the D symbol, copies of (Det) used in the proof), and the height $h$ of the tree. For a multiset of D-terms, $c$ is the number of inner nodes of the minimal DAG representing the set of its members, and $t$ is the sum and $h$ the maximum of the respective values of its members.

CD Tools provides various predicates that enumerate all D-terms of or up to some given size measure, together with the respective MGT, where D-terms which have no MGT because the constraints on the substitution are not satisfiable are omitted. For small values, these predicates can be used to obtain information about all proofs of a given size. They also can serve as simple theorem provers, forming the basis of the approach discussed in the next section and developed further in [42].

3. The SGCD Prover

CD Tools includes SGCD, a highly configurable structure generating prover for CD problems. The prover is parameterized by a generator, a predicate that enumerates all D-terms paired with MGTs at a given level. As an example of such a level, consider the tree size of the D-term. The generator can be used in two modes: goal-driven, where the MGT argument is considered as input and instantiated with a ground term, the goal to prove, and axiom-driven, where the MGT argument is a variable that is successively bound to derived lemmas.

In goal-driven mode, the system acts similar to “SETHEO-like” provers [33, 15, 13, 25, 39]. Invoking the generator at increasing levels realizes iterative deepening, the usual way of
operation of such systems. In axiom-driven mode, the generator would be invoked at increasing levels, where found solutions are cached. Later invocations at higher levels, which involve enumeration of solutions at lower levels as subproblems, then access these from the cache instead of recomputing them.

SGCD combines both modes in the following way: Basically it operates in the axiom-driven mode, but before a new level is computed and cached, it invokes the goal-driven mode, at the new level and, depending on the configuration, possibly at a number of increasingly higher levels. The goal-driven mode has there access to the cache, which it uses for subproblems that involve finding solutions below the new level.

For experimenting, various generators have been implemented, for tree size, height, and other measures, also in variations concerning the order in which structures appear. Each structure appears only once, although it is of course possible that different structures prove the same lemma (i.e., have the same MGT with respect to the given axioms). The prover can stop after a proof of a given goal has been found or enumerate alternate proofs. Extreme configurations realize pure axiom-driven lemma computation and pure goal-driven proof search. The cached solutions are triples of MGT (a lemma), D-term (its proof) and the level at which the proof was found. After the axiom-driven mode has completed a level, the cache is updated with the solutions newly found at the level, which is configurable in many ways: New solutions whose MGT is subsumed by some other solution can be deleted. The number of cached solutions can be given a limit, where the solutions to delete in order to stay within the limit are determined by an ordering of the union of the old cache and the new solutions, e.g., according to some size measure of the MGT. If a solution is deleted from a cached level, it does no longer appear when that level is accessed for a subproblem of a problem at a higher level. However, deleted solutions can be kept in an extra store, which can be useful for experimenting and for applications with axiom-driven theorem generation.

4. Experiments

4.1. SGCD and the Condensed Detachment Problems in the TPTP

Table 1 summarizes the performance of SGCD on the TPTPDT2 problems in comparison to state-of-the-art provers as represented by the TPTP rating value. A value of 1.00 indicates that the problem is most difficult, i.e., can not be solved by any state-of-the-art prover (assuming some reasonable resource constraints). Also own experiments with E 2.6 [32] and Prover9 [21]. The latter plays a special role, because for CD problems it chooses by default hyperresolution as inference rule and thus emits, like SGCD, actual CD proofs. Moreover, its output format permits easy translation to D-terms, which is implemented in CD Tools. As indicated in [38], CD proofs, in contrast to proofs of other calculi, may be explicitly desirable in applications.

Each row in the table shows for a set of problems its cardinality. The first five rows describe the corpus and the performance of state-of-the art provers on it: Corpus TPTPDT2 is the whole

---

9That each structure appears only once is not an essential precondition for SGCD. The level characterizations considered so far in experiments could all be easily implemented such that each structure is returned only once.

9TPTP rating values considered in this paper refer to the latest values in TPTP 8.0.0 or in TPTP 7.5.0, which both provides the same values for the considered problems.
Table 1
The performance of SGCD in configurations that blend goal- and axiom-driven search.

| Corpus TPTPCDT2               | 196          |
|-------------------------------|--------------|
| Corpus TPTPCDT2, Rating < 1.00| 189          |
| Corpus TPTPCDT2, Rating > 0.00| 45           |
| Corpus TPTPCDT2, Rating > 0.25| 20           |
| E 2.6                         | 185          |

| SGCD-1                        | 165          |
|-------------------------------|--------------|
| SGCD-1+2+3+7, Rating > 0.00   | 25           |
| SGCD-1+2+3+7, Rating > 0.25   | 5            |

| Prover9                       | 168          |
|-------------------------------|--------------|
| SGCD-1+2+3+7 \ Prover9        | 13           |
| Prover9 \ SGCD-1+2+3+7        | 5            |

corpus, the subsequent three rows show its cardinality under restrictions of the problem rating. Row E 2.6 shows the number of problems in the corpus solved by E 2.6 [32].

SGCD was tested in about two dozens configurations of which SGCD-1 could solve the most problems. There the generator is over tree size, goal-driven search is in two levels, solutions where the dimensions of the MGT exceeds 5 times the maxima in the input are deleted, solutions with MGTs subsumed by previously found solutions are deleted, the cache is ordered according to height and size of the MGTs and its size is limited to 1000 solutions. Row SGCD-1 shows the number of problems solved by SGCD in this configuration.

In general, the performance of a first-order prover on a given problem depends strongly on parameter settings, such that mature systems are typically invoked in a parallel or portfolio mode with a scheduled pattern of alternate settings. Since SGCD does currently not support such a parallel or portfolio-like invocation, it seems adequate to consider its results for different configurations grouped together. SGCD-2, SGCD-3 and SGCD-7 supplement configuration SGCD-1 with slightly varied settings, in particular a generator over height, a larger cache size of 3000, and differently characterized limitations of the MGT dimensions. Row SGCD-1+2+3+7 shows the number of problems solved in at least one of these four configurations, each tried with a timeout of 2400 s. Further rows show the number of the solved problems whose rating is above specific thresholds. None of them has rating 1.00. Row Prover9 shows the number of problems solved by Prover9 in default configuration. Two further rows indicate the numbers of problems that can be solved either by SGCD in at least one of the four considered configurations or by Prover9, but not both. The set of problems solved by E 2.6 includes those solved by SGCD as well as those solved by Prover9.

---

10 On a HPC system with Intel® Xeon® Platinum 9242 @ 2.30GHz CPUs, 3.7 GB memory per CPU and 600 s time limit per problem. Option settings were --auto-schedule --cpu-limit=600. A time limit of 2400 s with --cpu-limit=2400 lead to longer proving times but not to more solved problems.

11 All results for SGCD reported in this subsection were obtained with TPTP 8.0.0 on a HPC system with Intel® Xeon® Platinum 9242 @ 2.30GHz CPUs, 3.7 GB memory per CPU and 2400 s time limit per prover run and problem. If results for different configurations are considered together, each of these was given the 2400 s time limit.

12 Performed on the same hardware as those with SGCD, also with a timeout of 2400 s for each problem, supplemented for problems LCL020-1 and LCL021-1 with results obtained on a notebook with 16 GB RAM and Intel® Core™ i7-8550U @ 1.80GHz CPU, because these caused memory exhaustion on the HPC system.
In summary, the experiment shows that from the TPTPCDT2 problems SGCD can solve more 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., 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.

4.2. SGCD in Purely Goal-Driven Mode and Clausal Tableaux

SGCD stems from goal-driven provers that can be characterized in terms of the connection method, model elimination or clausal tableaux and operate in essence by enumerating proof structures. As Table 2 verifies, in purely goal-driven configurations SGCD is indeed very similar to these provers and can roughly solve the same problems as several of them taken together.

| SGCD-G1    | 81 |
| SGCD-G2    | 65 |
| SGCD-G1+G2 | 89 |
| TABX       | 92 |
| TABX \ SGCD-G1+G2 | 3 |
| CMProver   | 89 |
| SETHEO 3.3 | 65 |
| S-SETHEO   | 66 |
| lazyCoP 0.1| 42 |
| SATCoP 0.1 | 59 |

The table shows, like Table 1 the cardinalities of subsets of problems in corpus TPTPCDT2. The value for SGCD-1+2+3+7, the set of problems solvable by SGCD in at least one of four configurations that blend goal- and axiom-driven operation, is transferred from Table 1. Rows SGCD-G1 and -G2 show the number of solved problems for two configurations of SGCD that are purely goal-driven with generators over tree size and height, respectively. Row SGCD-G1+G2 shows the number of problems from that was solved by at least one of these configurations. The set SGCD-1+2+3+7 includes all of these.

Row TABX shows for comparison the number of problems that was solved by at least one of the following provers: CMProver \[39, 40\] in at least one of several considered configurations, SETHEO 3.3 \[24\], S-SETHEO \[14\], lazyCoP 0.1 \[30\] and SATCoP 0.1 \[31\]. The next two rows indicate the numbers of problems that can be solved either by SGCD in at least one of the two goal-driven configurations or by at least one of the five contributors to TABX, but not both.

\[13\]With hardware and timeout settings as described above in footnote 11.

\[14\]For CMProver according to http://cs.christophwernhard.com/pie/cmprover/evaluation_201803/tptp_neq.html, for the other systems according to the ProblemAndSolutionStatistics document of TPTP 8.0.0.
The remaining rows show the number of problems solved individually by the contributors to TABX.\textsuperscript{15}

In summary, from the viewpoint of the goal-driven provers based on clausal tableaux or the conventional connection method, SGCD in purely goal-driven configurations is very similar to a strong prover of that family and the possibility to blend goal- and axiom-driven proceeding in SGCD is a substantial improvement, roughly doubling the number of TPTPCDT2 problems that can be solved.

4.3. 68 Theses by Łukasiewicz in a Single Axiom-Driven SGCD Run

68 of the TPTPCDT2 problems are from the same source: Theses that follow from an axiomatization of propositional logic with three axioms by Łukasiewicz and are proven in his book [19]. They were used extensively with OTTER [20] by Larry Wos [45, 22, 46, 47]. In a purely axiom-driven configuration SGCD finds these 68 theses all together in about 2.5 min. As parameters that relate to the theses, the configuration only considers maximal term dimensions among all theses. The generator is over tree size and the cache size is limited to 3000 solutions, trimmed according to dimensions of the MGTs. The procedure exhausts after level (i.e., tree size) 240. All theses were found at levels below 150, 56 of them among the 3000 cached solutions and the remaining 12 ones among 126,839 residual deleted solutions. OTTER solved the problems in a single run after the introduction of weight templates [45, 46]. The focus in [47] was to find short overall proofs for subsets of the theses that represent axiom systems. As shown in Table 3, our initial results can not compete with the carefully developed proofs from [47], except with respect to tree size.

Table 3
Dimensions of the combined proofs of axiom systems (after [47]). In [47] compacted size is termed length and height level.

| Axiom System   | Proof by SGCD | Proof from [47] |
|----------------|---------------|-----------------|
| Church         | (43, 125, 19) | (21, 218, 15)   |
| Frege          | (62, 277, 19) | (28, 473, 17)   |
| Hilbert        | (44, 196, 17) | (23, 483, 15)   |
| Alt. Łukasiewicz | (48, 138, 17) | (24, 277, 15)   |
| Wos            | (57, 170, 18) | (25, 335, 16)   |

4.4. Dimensions of Proofs Returned by SGCD

As already mentioned, the proofs for CD problems by Prover9 in its default configuration can be easily translated to D-terms, allowing direct size comparisons. Table 4 shows for the 163 TPTPCDT2 problems solvable by both SGCD-1+2+3+7 (as specified in Sect. 4.1) and Prover9 the compacted size, tree size and height of the proofs by SGCD and Prover9, aggregated as average and median values. For SGCD, for each problem and size measure the minimal value of all the

\textsuperscript{15} Also leanCoP 2.1 [25] was tried. With a single exception, all TPTPCDT2 problems are provided in the TPTP CNF format, which is not accepted by leanCoP. Experiments in the same settings as SGCD on conversions to TPTP FOF format (with axioms and goal conjecture distinguished) lead to 50 solved problems, subsumed by those of TABX.
up to four proofs obtained with the four configurations of SGCD-1+2+3+7 is considered. The \( n \)-superscript indicates values after n-simplification [43], which has a strong effect on the proofs by Prover9 but only a very minor effect on those by SGCD. In summary, the table indicates that SGCD returns 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.

### Table 4
Size of CD proofs obtained by SGCD and Prover9.

|                | C-avg | C-med | T-avg | T-med | H-avg | H-med |
|----------------|-------|-------|-------|-------|-------|-------|
| SGCD-1+2+3+7\(^n\) | 15.80 | 12    | 29.36 | 17    | 8.52  | 6     |
| SGCD-1+2+3+7  | 15.83 | 13    | 29.36 | 17    | 8.52  | 6     |
| Prover9\(^n\)  | 23.09 | 18    | 19501.99 | 40 | 13.69 | 12 |
| Prover9        | 28.37 | 21    | 194736.83 | 93 | 16.90 | 13 |

4.5. PSP Structure Enumeration and Proofs with Small Compacted Size

We consider the objective to find proofs with small compacted size. CCS can be applied in a configuration that returns proofs with ascertained minimal compacted size by exhaustive search [42]. This succeeds for about 44% problems of the TPTPCDT2 corpus. The remaining problems have to be tackled with techniques that may not yield proofs of ascertained minimal compacted size, but typically proofs with small compacted size. Aside of the configurations of SGCD based on enumeration by tree size or height discussed in the previous sections, there is another prospective candidate, a novel proof enumeration strategy for SGCD called PSP (indicating Proof-SubProof), specified as follows, where level is understood in the sense of Sect. 3.

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

PSP was motivated by an analysis [43] of Meredith’s variant [23] of Łukasiewicz’s completeness proof for his shortest single axiom for implicational logic [18], where it was observed that most steps in Meredith’s proof can be ascribed the relationship between levels that underlies PSP. The experiments summarized in Table 5 evaluate the potential of SGCD in configurations with PSP and with other structure generation methods for finding proofs, with emphasis on small compacted size.

SGCD was there considered in five configurations with PSP structure generation and in eight “NonPSP” configurations, that is, configurations with structure generation over tree size or height, which included those of SGCD-1+2+3+7 considered in Sect. 4.1. Conditions were as described in footnote 11. Each row in Table 5 shows for a set of problems from TPTPCDT2 its cardinality, as well as the number of members with rating \( > 0.00 \) and the number of members with rating \( > 0.25 \). CCS-MinC is the set of those problems for which a proof with ascertained compacted size could be found with CCS [42, Sect. 3.1]. SGCD-PSP is the set of problems for which a proof could be found by SGCD in a PSP configuration, SGCD-NonPSP is the set of problems for which a proof could be found by SGCD in a NonPSP configuration. The four rows
Table 5
Proofs with small compacted size by $SGCD$ and $CCS$.

|                        | $>0.00$ | $>0.25$ |
|------------------------|---------|---------|
| Corpus TPTPCDT2        | 196     | 45      | 20      |
| $CCS$-$MinC$           | 86      | 3       | 0       |
| $SGCD$-$PSP$           | 153     | 14      | 2       |
| $SGCD$-$NonPSP$        | 176     | 25      | 5       |
| $CCS$-$MinC$ $\cap$ $SGCD$-$PSP$ | 85   | 3       | 0       |
| $CCS$-$MinC$ $\cap$ $SGCD$-$NonPSP$ | 86 | 3       | 0       |
| $CCS$-$MinC$ $\cap$ ($SGCD$-$PSP$ $=$ $CCS$-$MinC$) | 60 | 1       | 0       |
| $CCS$-$MinC$ $\cap$ ($SGCD$-$NonPSP$ $=$ $CCS$-$MinC$) | 59 | 1       | 0       |
| $\neg$ $CCS$-$MinC$ $\cap$ $SGCD$-$PSP$ | 68 | 11      | 2       |
| $\neg$ $CCS$-$MinC$ $\cap$ $SGCD$-$NonPSP$ | 90 | 22      | 5       |
| $\neg$ $CCS$-$MinC$ $\cap$ ($SGCD$-$PSP$ $<$ $SGCD$-$NonPSP$) | 44 | 5       | 1       |
| $\neg$ $CCS$-$MinC$ $\cap$ ($SGCD$-$NonPSP$ $<$ $SGCD$-$PSP$) | 43 | 17      | 4       |
| $\neg$ $CCS$-$MinC$ $\cap$ ($SGCD$-$PSP$ $=$ $SGCD$-$NonPSP$) | 3 | 0       | 0       |

prefixed by “$CCS$-$MinC$ $\cap$” relate $SGCD$-$PSP$ and $SGCD$-$NonPSP$ to $CCS$-$MinC$. The intersection symbol is to be read as set intersection. ($SGCD$-$PSP$ $=$ $CCS$-$MinC$) denotes the subset of problems from $SGCD$-$PSP$ for which a PSP configuration returned a proof with the ascertained minimal compacted size. The ($SGCD$-$NonPSP$ $=$ $CCS$-$MinC$) is the analog for the NonPSP configurations.

The five rows prefixed by “$\neg$ $CCS$-$MinC$ $\cap$” relate $SGCD$-$PSP$ and $SGCD$-$NonPSP$ to the set of problems for which the minimal compacted size could not be determined by exhaustive search with $CCS$, which are the problems that actually benefit from the enumeration methods and heuristic restrictions offered by $SGCD$. The five rows indicate the contribution of $SGCD$ in PSP and NonPSP configurations to increase the number of solvable problems, taking into account the compacted size of proofs. $\neg$ $CCS$-$MinC$ denotes the complement of $CCS$-$MinC$ with respect to the corpus TPTPCDT2. ($SGCD$-$PSP$ $<$ $SGCD$-$NonPSP$) denotes the subset of the problems of $SGCD$-$PSP$ that could either not be proven in the NonPSP configurations or could be proven in a NonPSP configuration only with a compacted size that is larger than the smallest compacted size obtained with a PSP-based configuration. ($SGCD$-$NonPSP$ $<$ $SGCD$-$PSP$) is analogous, with the roles of $SGCD$-$PSP$ and $SGCD$-$NonPSP$ switched. ($SGCD$-$PSP$ $=$ $SGCD$-$NonPSP$) denotes the set of problems in both $SGCD$-$PSP$ and $SGCD$-$NonPSP$ such that the smallest compacted size of proofs by the PSP configurations and the smallest compacted size of proofs by the NonPSP configurations are equal.

In summary, the table shows that the problems for which the required minimal compacted proof size can be determined successfully by exhaustive search can (with one exception) also be proven by $SGCD$ in both PSP and NonPSP configurations. The ascertained minimal size is achieved there with $SGCD$ for about 70% of the problems in either family of configurations. For problems where determining the minimal compacted proof size by exhaustive search with $CCS$ failed, configurations of both families succeed in many cases, roughly doubling the number of proven problems compared to those for which the required minimal compacted size can be determined. PSP configurations succeed there for a subset of about 3/4 of the problems on
which the NonPSP configurations succeed. When the compacted size of the proofs is taken into account, the contributions of both configuration families are of the same order, each family yielding for about one half of the solvable problems a proof with smaller compacted size than the smallest obtained with the other family.

For PSP, the experiments show that – although motivated by observations on a proof by a human for a particular problem – it is applicable as a general structure generation technique to prove a large portion of problems from corpus TPTP CD2 and, in particular, is useful to complement other structure generation techniques when the objective is to find proofs with small compacted size.

4.6. A Short Proof for Łukasiewicz’s Shortest Axiom

The completeness of Łukasiewicz’s shortest single axiom for the implicational propositional calculus has been proven originally [18] with dimensions ⟨34, 585, 29⟩ and in [23] refined to ⟨33, 669, 29⟩ [43]. Representations of these historic formal proofs by logicians are included in CD Tools. Guided by an analysis of them, slightly shorter proofs were obtained in [43]. With SGCD, the much shorter proof shown in Fig. 2 was found, whose dimensions are ⟨29, 92, 22⟩.

It was obtained by some interaction with the CD Tools system as follows: First, SGCD in a PSP-based configuration returned a relatively short proof of the most difficult of the three involved subproblems, LCL038-1, with dimensions ⟨22, 64, 22⟩, which compare to ⟨31, 491, 29⟩ in [23]. This proof was then supplemented with proofs of the other two subproblems, which were found through enumerating by tree size and selecting according to the compacted size of all three proofs taken together. The structure of the PSP-based proof of LCL038-1 with dimensions ⟨22, 64, 22⟩, step 7 in Fig. 2, is shown as DAG in Fig. 3.

Figure 2: ⟨29, 92, 22⟩-proof of the completeness of Łukasiewicz’s axiom in the Meredith’s notation [23].

1. $\text{CCCpq} \text{CCrpCsp}$
2. $\text{CCCCpq} \text{Cpr} \text{Cqs CtCqs} = \text{D11}$
3. $\text{CCCPq} \text{Cq} \text{Cqs Ctq CCsqCtq} = \text{D12}$
4. $\text{CCCCpq} \text{Cr} \text{Cst CCq Cst} = \text{DDDD1D1D1D1D1D1D1n11n1n1}$
5. $\text{CCCCpq} \text{Cq} \text{CCp} \text{Crq} \text{CCp} \text{Crq} \text{Ctq} \text{Ctq} = \text{D1D414}$
6. $\text{CCCPq} \text{Cr} \text{Cr} = \text{DD31n}$
*7. $\text{CCpq} \text{CCq} \text{Cr} \text{Cpr} = \text{DDDD1DD55n1n1n1}$
*8. $\text{CCpq} \text{pp} = \text{DD426n}$
*9. $\text{CpCq} \text{r} = \text{DD26n}$

4.7. Proof Structure Compression by Tree Grammars and by Combinators

Meredith’s notation used in Fig. 2 can be read as a tree grammar where the line numbers except of 1 for the axiom are nonterminals. For each nonterminal the grammar generates exactly one D-term. The grammar represents the proof as a DAG, where the number of written Ds is the compacted size. Stronger compressions can be achieved by grammars where the nonterminals are permitted to contain parameters [16], which has been considered before in proof theory for term substitutions in formulas [10]. SGCD supports experimenting with such
Figure 3: The structure of the proof of step 7 from Fig. 2 (LCL038-1) as DAG. Node numbers indicate the correspondence to that figure. A dashed arrow indicates that the actual formula used as minor premise plays no role to determine the conclusion, which is indicated by “n” in Meredith’s notation. (See also the discussion of n-simplification in [43].)
compressions applied to the proof structure through an interface to TreeRePair [17], an advanced tree compression system originally targeted at XML document trees. The right column of Fig. 4 shows a grammar obtained with this system for the subproof of 7 from Fig. 2.

**Figure 4:** A tree grammar compression of the subproof of step 7 from Fig. 2 (LCL038-1) (with \( n \) replaced by 1) obtained by TreeRePair invoked via CD Tools.

1. \( CCCpqrCCrpCsp \)  
2. \( CCCCpqrCCrpCsp \rightarrow t \)  
3. \( CCpqr \rightarrow CCrpCsp \)  
4. \( CCpqr \rightarrow CCCspCrtr \)  
5. \( CCCpqrCCrpCspCCCtuvCCvtCwtCCCxyzCCzxCaxb \)  
6. \( CCCpqrCCrpCspCCqtCst \)  
7. \( CCCpqCCpCqrCstCCqtCst \)  
8. \( CCCpqCCpCqrCptCstuCCuvtCCCxyzzCCzxCaxb \rightarrow b \)

The left column of Fig. 4 shows lemmas that correspond to the respective nonterminals. For a parameter-free nonterminal it is the MGT of the generated D-term, for a nonterminal with parameters it is a Horn clause. To represent these Horn clauses we extend the Polish notation for the lemmas corresponding to the proof steps. For example, \( CCpqr \rightarrow CCrpCsp \) of step 3 reads in our first-order representation as \( \text{Thm}( (p \Rightarrow q) \Rightarrow r) \rightarrow \text{Thm}( (r \Rightarrow p) \Rightarrow (s \Rightarrow p)) \). A common size measure for such grammars is the sum of the number of edges on the right-hand sides of the productions, 24 in the example. It compares to the doubled value of the compacted size, \( 2 \times 2 = 44 \), because each inner node of the DAG underlying the compacted size has two outgoing edges.

CD Tools also supports a novel way of proof structure compression introduced in [42], which works by permitting combinators in the D-terms. It can be used as basis for proof search over compressed structures and it lets uncompressed proof structures appear as normal forms, in the technical sense of combinator reduction, of compressed proof structures. Tree grammar compressions can be converted to combinator compressions with techniques from functional programming languages [27], which is described in [42] and implemented in CD Tools. Figure 5 shows an example, the combinatorial compression obtained from the grammar compression shown in Fig. 4. Its dimensions are \( \langle 19, 119, 15 \rangle \), which compare to \( \langle 22, 64, 22 \rangle \), the dimensions of the expanded structure shown in Fig. 3. The involved combinators are associated with the following rewriting rules.

\[
\begin{align*}
D(D(I', x), y) & \rightarrow_{\text{rew}} D(y, x) \\
D(D(D(B, x), y), z) & \rightarrow_{\text{rew}} D(x, D(y, z)) \\
D(D(D(B_4, x), y), z), u) & \rightarrow_{\text{rew}} D(x, D(y, D(z, u)))
\end{align*}
\]

Rewriting the tree representation of Fig. 5 with these rules yields the tree representation of Fig. 3. Combinatory compression as provided by CD Tools is discussed in more depth in [42].
5. Conclusion

A Prolog library for experimenting with CD in ATP has been presented. It puts an ATP-oriented formal view on CD that is centered around proof structures represented as terms [43] into practice. SGCD, an included first-order prover is specialized on CD problems and competes for these with the lower end of general state-of-the-art provers. It is based on a simple but so far not fully explored paradigm, enumerating proof structures in a way that combines two modes: a goal-driven mode, as known from systems based on the connection method or clausal tableaux, and an axiom-driven mode that generates lemmas in a level-by-level way. In generic configurations it produces comparatively short proofs. With special configurations very short proofs of difficult problems can be found, for which the systematic detection of shortest proofs seems too hard.
We conclude the presentation with some remarks on related work and perspectives. Similarly to SGCD, an implementation [26] of hypertableaux [2] creates lemmas axiom-driven level-by-level, however much less flexibly configurable and incorporating goal-driven phases only in a rudimentary way, by immediate termination before a level is completed if the goal is generated. A recent enhancement of E by machine learning supplements a classification based on derivation history as clause selection guidance [34]. This may be viewed as introducing into E a bit of the structure generating approach, where the derivation history in form of D-terms is the main guidance. That the approach of SGCD yields particularly small proofs should be of general relevance for applications that are based on proof structures emitted by reasoners, for example to produce “good” explanations [1] and for interpolation [41]. In extreme configurations, SGCD acts purely goal-driven or purely axiom-driven. Purely goal-driven configurations perform very similarly to systems based on the connection method or clausal tableaux, which, however, are for the CD problems much weaker than state-or-the-art provers. Purely axiom-driven configurations may be useful as a basis for finding mathematically interesting theorems from axioms. CD Tools is a versatile platform for experimental studies of various kinds such as, for example, investigating grammar- and combinator-based compressions of proof structures, which are both stronger than just compressing trees into DAGs. Combinator-based compression is not only applicable to shorten a given proof, but also for practical proof search, realizing features known from the connection structure calculus [6, 4] but not implemented previously. CD Tools made the initial elaboration and evaluation of this observation possible [42]. This involved writing a second structure generating prover, CCS, which is now also included in the system.

Acknowledgments

The author thanks Wolfgang Bibel for inspiration, and him as well as anonymous reviewers of a previous version for helpful suggestions 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] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, and A. Kovtunova. Finding good proofs for description logic entailments using recursive quality measures. In A. Platzer and G. Sutcliffe, editors, CADE 28, volume 12699 of LNCS (LNAI), pages 291–308. Springer, 2021. https://doi.org/10.1007/978-3-030-79876-5_17.
[2] 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. https://doi.org/10.1007/3-540-61630-6_1.
[3] W. Bibel. Automated Theorem Proving. Vieweg, Braunschweig, 1982. https://doi.org/10.1007/978-3-322-90102-6. Second edition 1987.
[4] W. Bibel and E. Eder. Methods and calculi for deduction. In D. M. Gabbay, C. J. Hogger, and
[5] 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. https://doi.org/10.1007/978-3-030-49424-7_13.

[6] 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. https://doi.org/10.1007/3-540-56296.

[7] 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. https://doi.org/10.1007/10710078_37.

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

[9] D. Fuchs and M. Fuchs. CODE: A powerful prover for problems of condensed detachment. In W. McCune, editor, CADE-14, pages 260–263. Springer, 1997. https://doi.org/10.1007/3-540-61324-1_25.

[10] S. Hetzl. Applying tree languages in proof theory. In A.-H. Dediu and C. Martin-Vide, editors, LATA 2012, volume 7183 of LNCS, pages 301–312, 2012. https://doi.org/10.1007/978-3-642-28332-1_26.

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

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

[13] R. Letz. Tableau and Connection Calculi. Structure, Complexity, Implementation. Habilitationsschrift, TU München, 1999. Available from http://www2.tcs.tu-muenchen.de/~letz/habil.ps, accessed Jun 30, 2022.

[14] R. Letz and G. Stenz. Model elimination and connection tableau procedures. In A. Robinson and A. Voronkov, editors, Handb. of Autom. Reasoning, volume 1, pages 2015–2114. Elsevier, 2001.

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

[16] M. Lohrey. Grammar-based tree compression. In I. Potapov, editor, DLT 2015, volume 9168 of LNCS, pages 46–57. Springer, 2015. https://doi.org/10.1007/978-3-319-21500-6_3.

[17] M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. Inf. Syst., 38(8):1150–1167, 2013. https://doi.org/10.1016/j.is.2013.06.006. System available from https://github.com/dc0d32/TreeRePair, accessed Jun 30, 2022.

[18] 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.

[19] J. Łukasiewicz. Elements of Mathematical Logic. Pergamon Press, 1963. English translation of the second edition (1958) of Elementy logiki matematycznej, PWM, Warszawa.

[20] 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.

[21] W. McCune. Prover9 and Mace4. http://www.cs.unm.edu/~mccune/prover9, 2005–2010.

[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. https://doi.org/10.1007/3-540-55602-8_167.

[23] 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. https://doi.org/10.1305/ndjfl/1093957574.

[24] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and E-SETHEO – the CADE-13 systems. J. Autom. Reasoning, 18(2):237–246, 1997. https://doi.org/10.1023/A:1005808119103.

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

[26] 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. https://doi.org/10.1007/978-3-540-73595-3_37.

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

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

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

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

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

[32] S. Schulz, S. Cruanes, and P. Vukmirović. Faster, higher, stronger: E 2.3. In P. Fontaine, editor, CADE 27, number 11716 in LNAI, pages 495–507. Springer, 2019. https://doi.org/10.1007/978-3-030-79876-5_31.

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

[34] 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, 2019. https://doi.org/10.1007/978-3-030-79876-5_31.

[35] G. Sutcliffe. The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning, 59(4):483–502, 2017.

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

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

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

[39] C. Wernhard. The PIE system for proving, interpolating and eliminating. In P. Fontaine,
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. https://doi.org/10.1007/978-3-030-46714-2_11.

C. Wernhard. Craig interpolation with clausal first-order tableaux. J. Autom. Reasoning, 65(5):647–690, 2021. https://doi.org/10.1007/s10817-021-09590-3.

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, CEUR Workshop Proc. CEUR-WS.org, 2022. Preprint available from http://cs.christophwernhard.com/papers/ccs/.

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. https://doi.org/10.1007/978-3-030-79876-5_4.

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

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. https://doi.org/10.1007/978-94-011-3488-0_15.

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

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

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

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. https://doi.org/10.1007/3-540-52885-7_109.