Exploring Metamath Proof Structures: Progress Report

Christoph Wernhard [1] Zsolt Zombori [2]

[1]University of Potsdam [2]HUN-REN Alfréd Rényi Institute of Mathematics and Eötvös Loránd University

AITP 2025

Aussois, September 1, 2025

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 457292495; Funded by the Hungarian Artificial Intelligence National Laboratory Program (RRF-2.3.1-21-2022-00004) as well as the ELTE TKP 2021-NKTA-62 funding scheme; Based upon work from the action CA20111 EuroProofNet supported by COST

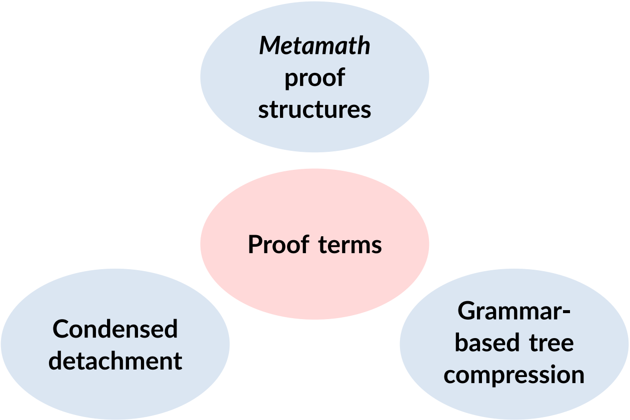
Exploring Metamath Proof Structures: Progress Report

1. Introduction
2. *Metamath* KBs as Grammar-Compressed Proof Terms
3. Some Subtleties of our ”Proof Theory”
4. Grammar-Compressing Proof Terms
5. Expriments and Some Potential Application Contexts
6. Conclusion

Approach

We structure large proofs into manageable lemmas

- **driven by proof structures considered as compressed terms**
- **lemma *formulas* then come second, determined by substructures of the compression**



Particular Theses

We structure large proofs into manageable lemmas

- **driven by proof structures considered as compressed terms**
- **lemma *formulas* then come second, determined by substructures of the compression**

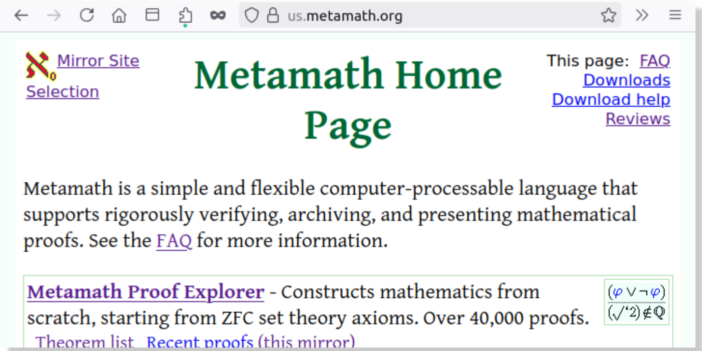
I. ***Compression of proof structures* is a suitable approach to *lemma synthesis***
   - Quality of a lemma is indicated by its effects on proof structure

II. ***For lemma synthesis not only compression "from scratch" can be useful but also further compression applied to already compressed structures***
   - E.g., to let machine suggest improvements of given human structurings

III. ***Structuring of mathematical knowledge by human experts, as with Metamath, is worth systematic investigation for understanding human reasoning***
   - How far can human structurings be modeled by mechanical compression methods?

IV. ***A mathematical KB with proofs in an ATP format helps to advance ATP***
   - These proofs provide examples of the desired ATP results from which ATP may "learn"

Metamath



- By Norman Megill, started early 1990s; contributors include David A. Wheeler, Mario Carneiro
- "Formalizing 100 Theorems": Isabelle 92; HOL Light 89; Coq 79; Lean 79; **Metamath 74**; Mizar 69
- ***Metamath Proof Explorer*** aka ***set.mm***

Topic	*1st Thm*
Propositional calculus	1
Predicate calculus	1,744
Zermelo-Fraenkel set theory	2,650
The axiom of replacement	5,086
The axiom of choice	9,916
Tarski-Grothendieck set theory	10,157
Real and complex numbers	10,304
Elementary number theory	15,391
Basic structures	16,243
Basic category theory	16,695
Basic order theory	17,238
Basic algebraic structures	17,517
Basic linear algebra	19,918
Basic topology	20,936
Basic real and complex analysis	23,316
Basic real and complex functions	23,897
Elementary geometry	25,398
Graph theory	25,912
Guides, miscellanea, examples	27,236
Deprecated material	27,321
70 mathboxes	29,111
Last Thm	43,920

Metamath

- "Metavariable mathematics" – use of **metavariables over an object logic**
- **Simplest** framework that allows essentially all of mathematics to be expressed with absolute rigor
  - All statements treated as mere **sequences of symbols**, i.e., constant and variable tokens

(	ph	->	(	ps	->	ph	)	)

  - Metamath just knows how to **substitute strings of symbols for the variables**, based on instructions you provide it in a proof, subject to constraints you specify for the variables
- **No particular set of axioms**, axioms are defined in a DB
- **Almost no hard-wired syntax**; syntax also defined via substitution rules in the DB
  - Parsing is done **within proofs**, based on declarations in the DB
  - It is easy to **strip off the "syntactic" parts** from proofs; tools by default do not show them
- **Specification** and introduction: **Metamath book** (free PDF)
  [Megill, Wheeler: *Metamath – A Computer Language for Mathematical Proofs, 2nd. ed*, 2019]
- No single canonical tool: **many verifiers and proof assistants**, with *metamath.exe* as a reference
  - *metamath.exe* verifies *set.mm* (44,000 theorems) in **7.5 s**, an optimized system in **0.2 s**

The *CD Tools* Environment for Experimenting with Condensed Detachment ...

- Written in ***SWI-Prolog***
- Extends the *PIE* (*Proving*, *Interpolating*, *Eliminating*) environment [W, 2016; 2020]
  - Provides interfaces to *TPTP* and many first-order provers
- Includes structure-generating provers for CD and Horn problems: *SGCD*, *CCS* [W 2022; Rawson, W, Zombori, Bibel 2023; W 2024; W, Bibel 2024]
- **New: methods and support for grammar-based tree compression**
- **New: *Metamath* interface**, written from scratch in *SWI-Prolog*
  - Also **proofs are translated to Prolog terms**, with various options
    - With and without *Metamath*'s “syntactic” steps
    - Inference of “syntactic” steps that meet disjoint-variable restrictions
    - Compatible with other proof terms in *CD Tools*
  - Prolog fact base generated from *set.mm* in 2 min; after compilation it loads in 0.5 s
- So, now we assume we have read-in *set.mm* into our *SWI-Prolog* ...

The Logic Behind set.mm: First-Order Horn Logic with a Single Predicate "is_theorem"

A theorem statement in *set.mm*

```
${
  notbid.1 $e |- ( ph -> ( ps <-> ch ) ) $.
  $( Deduction negating both sides of a logical equivalence.  (Contributed by
     NM, 21-May-1994.) $)
 notbid $p |- ( ph -> ( -. ps <-> -. ch ) ) $=
    wph wps wn wch wn wph wps wch wps wn wn wch wn wn notbid.1 wps notnotb
    wch notnotb 3bitr3g con4bid $.
$}
```

Converted representation as first-order definite clause

```
is_theorem(X=>wb(n(Y),n(Z))) <- is_theorem(X=>wb(Y,Z))
```

We may omit the "`is_theorem`" predicate

```
(X=>wb(n(Y),n(Z))) <- (X=>wb(Y,Z))
```

Pre-view: the proof as tree grammar production

```
notbid(V) -> con4bid(3bitr3g(V, notnotb, notnotb))
```

Proof Terms: Primitives, Most General Theorem (MGT)

[Megill: *A Finitely Axiomatized Formalization of Predicate Calculus with Equality*, 1995]

There are two primitive proof term constructor functions:

- **Condensed detachment** (modus ponens, modulo **most general unification**)

```
If           A : is_theorem(X=>Y)
and          B : is_theorem(X)
then    d(A,B) : is_theorem(Y)
```

- Condensed generalization

```
If           A : is_theorem(X)
then      g(A) : is_theorem(forall(Y, X))
```

For given axioms, a proof term proves its most general theorem (MGT), or its MGT is undefined

```
        ax1 :: is_theorem(X=>(Y=>X))              Simp, K
d(ax1,ax1)  :  is_theorem(X=>(Y=>(Z=>Y)))
```

Towards Compressing Proof Terms: A Proof Term

Given axiom

```
ax1                                  :: (X=>(Y=>X))
```

Proof term built from primitives d, g and axiom constants

```
d(ax1,                                : (X=>(Y=>(Z=>(U=>Z))))
  d(ax1,
    d(d(ax1, ax1),
      d(ax1, ax1))))
```

DAG-Compressed Proof Terms

Given axiom

```
ax1                                   :: (X=>(Y=>X))
```

Proof term built from primitives d, g and axiom constants

```
d(ax1,                                 : (X=>(Y=>(Z=>(U=>Z))))
  d(ax1,
    d(d(ax1, ax1),
      d(ax1, ax1))))
```

Compression to minimal DAG – factoring repeated subtrees

```
p1    -> d(ax1, ax1)                   : (X=>(Y=>(Z=>Y)))
start -> d(ax1, d(ax1, d(p1, p1)))     : (X=>(Y=>(Z=>(U=>Z))))
```

Grammar-Compressed Proof Terms

Given axiom

```
ax1                                                  :: (X=>(Y=>X))
```

Proof term built from primitives d, g and axiom constants

```
d(ax1,                                               : (X=>(Y=>(Z=>(U=>Z))))
  d(ax1,
    d(d(ax1, ax1),
      d(ax1, ax1))))
```

Compression to minimal DAG – factoring repeated subtrees

```
p1    -> d(ax1, ax1)                                 : (X=>(Y=>(Z=>Y)))
start -> d(ax1, d(ax1, d(p1, p1)))                   : (X=>(Y=>(Z=>(U=>Z))))
```

Grammar compression – factoring repeated patterns

```
p1(V) -> d(ax1, V)                                   : (Y=>X) <- X
p2    -> p1(ax1)                                     : (X=>(Y=>(Z=>Y)))
start -> p1(p1(d(p2, p2)))                           : (X=>(Y=>(Z=>(U=>Z))))
```

The MGT of a Proof Term with Parameters

Given axiom

```
ax1                                       :: (X=>(Y=>X))
```

Proof term built from primitives d, g and axiom constants

```
d(ax1,                                     : (X=>(Y=>(Z=>(U=>Z))))
  d(ax1,
    d(d(ax1, ax1),
      d(ax1, ax1))))
```

Compression to minimal DAG – factoring repeated subtrees

```
p1    -> d(ax1, ax1)                       : (X=>(Y=>(Z=>Y)))
start -> d(ax1, d(ax1, d(p1, p1)))         : (X=>(Y=>(Z=>(U=>Z))))
```

Grammar compression – factoring repeated patterns

```
p1(V) -> d(ax1, V)                         : (Y=>X) <- X
p2    -> p1(ax1)                           : (X=>(Y=>(Z=>Y)))
start -> p1(p1(d(p2, p2)))                 : (X=>(Y=>(Z=>(U=>Z))))
```

- MGT of proof term with parameters: a definite clause with a body atom for each parameter

The Grammar-MGT of a LHS of a Proof Grammar

Given axiom

```
ax1                                         :: (X=>(Y=>X))
```

Proof term built from primitives d, g and axiom constants

```
d(ax1,                                      : (X=>(Y=>(Z=>(U=>Z))))
  d(ax1,
    d(d(ax1, ax1),
      d(ax1, ax1))))
```

Compression to minimal DAG – factoring repeated subtrees

```
p1    -> d(ax1, ax1)                        : (X=>(Y=>(Z=>Y)))
start -> d(ax1, d(ax1, d(p1, p1)))          : (X=>(Y=>(Z=>(U=>Z))))
```

Grammar compression – factoring repeated patterns

```
p1(V) -> d(ax1, V)                          : (Y=>X) <- X
p2    -> p1(ax1)                            : (X=>(Y=>(Z=>Y)))
start -> p1(p1(d(p2, p2)))                  : (X=>(Y=>(Z=>(U=>Z))))
```

- **MGT of proof term with parameters: a definite clause with a body atom for each parameter**
- **Grammar-MGT – MGT computation on compressed structure**

Excerpt of an Actual Proof from set.mm

```
a1i(V)             -> d(ax1, V)                 : (Y=>X) <- X
a2i(V)             -> d(ax2, V)                 : ((X=>Y)=>(X=>Z)) <- (X=>(Y=>Z))
con4               -> ax3                       : (n(X)=>n(Y))=>(Y=>X)
mp2(V1, V2, V3)    -> d(d(V3, V1), V2)          : Z <- X, Y, (X=>(Y=>Z))
con4i(V)           -> d(con4, V)                : (Y=>X) <- (n(X)=>n(Y))
                    ⋮
bitrid(V1, V2)     -> bitrd(a1i(V1), V2)        : (Z=>wb(X,U)) <- wb(X,Y), (Z=>wb(Y,U))
bitr3id(V1, V2)    -> bitrid(bicomi(V1), V2)    : (Z=>wb(Y,U)) <- wb(X,Y), (Z=>wb(X,U))
3bitr3g(V1, V2, V3) -> bitrdi(bitr3id(V2, V1), V3)
                                    : (X=>wb(U,W)) <- (X=>wb(Y,Z)), wb(Y,U), wb(Z,W)
notbid(V)          -> con4bid(3bitr3g(V, notnotb, notnotb))
                                                : (X=>wb(n(Y), n(Z))) <- (X=>wb(Y,Z))
```

Dimensions

Number of productions:	103
$\lvert G \rvert$, grammar size, total number of edges of RHSs:	274
Size of expansion built from d and ax1-3:	398,932
Size of minimal DAG:	550

Exploring Metamath Proof Structures: Progress Report

User-Specified MGT Instantiations in *Metamath*

Proof grammar		*MGT*	*User-specified instance*
`p1(V) -> d(ax1, V)`	:	`(Y=>X) <- X`	`(Y=>X) <- X`
`p2 -> p1(ax1)`	:	`(X=>(Y=>(Z=>Y)))`	`(X=>(Y=>(X=>Y)))`
`start -> p1(p1(d(p2, p2)))`	:	`(X=>(Y=>(Z=>(U=>Z))))`	`(X=>(Y=>(Z=>((U=>(W=>(U=>W)))=>Z))))`

A linear proof term and its MGT

```
d(V1, d(V2, ax1))        :        Y <- (X=>Y), ((Z=>(U=>Z))=>X)
```

A similar but non-linear proof term and its MGT

```
d(V, d(V, ax1))          : (X=>(Y=>X)) <- ((X=>(Y=>X))=>(X=>(Y=>X)))
```

- MGT requirements induced for **all occurrences** of a parameter are unified
- For non-linear proof terms two ways to determine the MGT of $d[V \mapsto d']$ diverge
  1. MGT after performing the substitution: $\mathrm{mgt}(d[V \mapsto d'])$
  2. MGT determined from MGT of $\mathrm{mgt}(d[V])$ and MGT of d':
     $A\sigma$, where $\mathrm{mgt}(d[V]) = A \leftarrow B$, and $\sigma = \mathrm{mgu}(B, \mathrm{mgt}(d'))$

$A\sigma$ is a (possibly strict) instance of $\mathrm{mgt}(d[V \mapsto d'])$

Features of our "Proof Theory" – A Generalization of Condensed Detachment

- The "proves" relation between proof terms and formulas is specified with an inference system

$$\dfrac{\mathtt{d} :: y \leftarrow (x \Rightarrow y) \land x \qquad \dfrac{\mathtt{ax1} :: (x \Rightarrow (y \Rightarrow x))}{\mathtt{ax1} : (x' \Rightarrow (y' \Rightarrow x'))}\,\text{APP} \qquad \dfrac{\mathtt{ax1} :: (x \Rightarrow (y \Rightarrow x))}{\mathtt{ax1} : (x'' \Rightarrow (y'' \Rightarrow x''))}\,\text{APP}}{\mathtt{d}(\mathtt{ax1}, \mathtt{ax1}) : (y' \Rightarrow (x'' \Rightarrow (y'' \Rightarrow x'')))}\,\text{APP}$$

- **Reified proof terms** (not just implicitly formed graphs)
- "Efficiency" not addressed in the spec: **proof search is building proof terms**, in whatever ways
- **MGT: the unique most general formula proven by a proof term**
  - Determined via **unification**
  - A **definite clause**, body atoms corresponding to **parameters in the proof term**
  - For a **nonlinear proof terms**, formulas for **all occurrences of a parameter** are unified
- **Proof grammar**: compressed representation of a proof tree or a set of proof trees
  - **Proofs of lemmas correspond to grammar productions**
  - Grammar-MGTs determine the MGTs **efficiently directly on the grammar compression**
- Theorems can be **user-specified strict instances of their proof's MGT**

Combinator Term DAGs as an Alternative to Grammars

Given proof term

```
d(d(ax1, ax1),
  d(d(d(ax1, ax1),
      ax1),
    ax1))
```

Grammar compression

```
p1(V) -> d(V, ax1)
p2    -> p1(ax1)
start -> d(p2, p1(p1(p2)))
```

Combinator DAG in D-term syntax

```
F1 = d(d(C, I), ax1)
F2 = d(F1, ax1)
F3 = d(F2, d(F1, d(F1, F2)))
```

Combinator DAG

$$f_1 = \mathbf{CI}\mathsf{a}_1$$
$$f_2 = f_1\mathsf{a}_1$$
$$f_3 = f_2(f_1(f_1 f_2))$$

Involved combinators

$$\mathbf{C} = \lambda xyz.xzy$$
$$\mathbf{I} = \lambda x.x$$
$$\mathbf{CI} = \lambda xy.yx$$

Combinator term

$$\mathbf{CI}\mathsf{a}_1\mathsf{a}_1(\mathbf{CI}\mathsf{a}_1(\mathbf{CI}\mathsf{a}_1(\mathbf{CI}\mathsf{a}_1\mathsf{a}_1)))$$

1. Introduction
2. *Metamath* KBs as Grammar-Compressed Proof Terms
3. Some Subtleties of our ”Proof Theory”
4. **Grammar-Compressing Proof Terms**
5. Expriments and Some Potential Application Contexts
6. Conclusion

The Save-Value of a Production

$$\text{save-value}_G(\mathit{Production}) \;:=\; |G \text{ after eliminating } \mathit{Production}| - |G|$$

- **Indicates a production's contribution to the compression** [Lohrey et al., 2013]
- Can be positive, zero, or negative

Let G =

```
p1(V) -> d(ax1, V)                  : (Y=>X) <- X
   p2 -> p1(ax1)                    : (X=>(Y=>(Z=>Y)))
start -> p1(p1(d(p2, p2)))          : (X=>(Y=>(Z=>(U=>Z))))
```

After **eliminating (unfolding and removing)** `p1` we obtain

```
   p2 -> d(ax1, ax1)                : (X=>(Y=>(Z=>Y)))
start -> d(ax1, d(ax1, d(p2, p2)))  : (X=>(Y=>(Z=>(U=>Z))))
```

We have

$$\begin{array}{lcll} |G| & = & 2+1+4 & = 7 \\ |G \text{ after eliminating p1}| & = & 2+6 & = 8 \\ \text{save-value}_G(\text{p1}) & = & 8-7 & = 1 \end{array}$$

[Lohrey, Maneth, Mennicke: *XML Tree Structure Compression using RePair*, 2013]

Phase 1: Replacement

Input: A term (may be represented as DAG)

Loop:
- Find a repeated pattern $f(\_, g(\_), \_)$
- Generate a production $h(\_)$ -> $f(\_, g(\_), \_)$
- In the term, fold into the production

```
d(ax1,
  d(ax1,
    d(d(ax1, ax1),
      d(ax1, ax1))))
```

Generated: `p1(V) -> d(ax1, V)`

```
p1(p1(d(p1(ax1),
        p1(ax1))))
```

Generated: `p2 -> p1(ax1)`

```
p1(p1(d(p2,
        p2)))
```

Output: A grammar: the generated productions and a start production to the final main term

Phase 2: Pruning

Eliminate productions with save-value ≤ 0

All stages are sensitive to configuration and heuristics

Output of replacement

```
p1(V) -> d(ax1, V)
p2    -> p1(ax1)
start -> p1(p1(d(p2, p2)))
```

Proof Compression Workflow

	Processing stage	*Kind*	*Source*	$\|G\|$	$\#Prod(G)$
	Initial set of trees			5×10^{22}	17
	Initial set of trees as DAG			21,472	927
1.	TreeRePair replacement phase	Structural	[Lohrey et al., 2013]	9,739	4,153
2.	TreeRePair pruning phase	Structural	[Lohrey et al., 2013]	3,683	905
3.	Nonlinear compression	Structural		3,204	604
4.	Same-value reduction	Structural		3,174	593
5.	MGT-based reduction	Formula-related		3,017	534

- **Nonlinear compression:** introduce nonlinear productions for RHS occurrences of a nonterminal with repeated arguments
- **Same-value reduction:** eliminate multiple nonterminals with the same expansion
- **MGT-based reduction:** eliminate productions for which the grammar-MGT is subsumed by that of another production
- Subtleties
  - Consideration of parameters modulo permutation
  - Configuration such that specified **top-level theorems** still have productions

Comparing human and machine compression – for 17 selected theorems (MINISET)

- Human compression ($|G|$ = 2,302) still **better than machine compression** ($|G|$ = 3,017) – why?
- 29% of MGTs in machine compression are **also in the human compression**; even 34% in set.mm – **many of the synthesized lemmas seem useful**

Compressing a given human grammar further

- Reduces $|G|$ of MINISET from 2,302 to 1,831
- Works for **large subsets of set.mm (mathematical topics)**; grammar-size reduction 4%–30%
- Result lemmas often look nice

Core part of set.mm as grammar

- 28% of the productions are **nonlinear**
- 8.4% of the theorems are a **strict instance of the MGT**
- 10% of productions have **save-value** 0; 12% < 0 – purposes of these redundancies?
- 3.1% are **duplicate theorems**; 3.9% subsumed – purposes of these redundancies?

The dependency graph as complex network: **edges $p \to q$ for each occurrence of q in the RHS for p**

- Found to be **scale-free**, for both human and machine compression

Grammar-based proof compression for lemma synthesis
[Vyskocil, Stanovský, Urban: *Automated Proof Compression by Invention of New Definitions*, 2010]
[Hetzl: *Applying Tree Languages in Proof Theory*, 2012]

- **Compression applied there to formulas – here to proof terms**

Structuring ATP proofs [Schulz: *Analyse und Transformation von Gleichheitsbeweisen*, 1993]

- Structure-based criteria; special cases of standard grammar measures like save-value
- *Isolated proof segments*: important *for given proof* if used often within it but rarely from outside

Premise selection [Kaliszyk, Urban: *Learning-Assisted Theorem Proving with Millions of Lemmas*, 2015]

- Relevant are not only named theorems, but also **lemmas used implicitly in proofs**
- Such lemmas can be taken into account at **different levels**: kernel/tactics
- **Here: same language for all levels; levels formally related by lossless compression**

Hammering [Carneiro, Brown, Urban: *Automated Theorem Proving for Metamath*, 2023]

- We obtain same FO-formulas; no tree expansion of proofs; inference of “syntactic” proof parts

1. Introduction
2. *Metamath* KBs as Grammar-Compressed Proof Terms
3. Some Subtleties of our ”Proof Theory”
4. Grammar-Compressing Proof Terms
5. Expriments and Some Potential Application Contexts
6. **Conclusion**

Conclusion – Looking Back at the Specific Main Theses

We structure large proofs into manageable lemmas

- **driven by proof structures considered as compressed terms**
- **lemma *formulas* then come second, determined by substructures of the compression**

I. ***Compression of proof structures* is a suitable approach to *lemma synthesis***
   - We introduced grammar compression of proof terms, productions representing lemma proofs
   - First experiments give apparently useful lemmas

II. ***For lemma synthesis not only compression "from scratch" can be useful but also further compression applied to already compressed structures***
   - First experiments show some scalability and give apparently useful lemmas

III. ***Structuring of mathematical knowledge by human experts, as with Metamath, is worth systematic investigation for understanding human reasoning***
   - Grammar translation of *set.mm* and machine compressions allow precise comparisons

IV. ***A mathematical KB with proofs in an ATP format helps to advance ATP***
   - Grammar translation of *set.mm* proofs should provide a suitable form

Exploring Metamath Proof Structures: Progress Report

References I

[Albert and Barabási, 2002] Albert, R. and Barabási, A.-L. (2002).
Statistical mechanics of complex networks.
Reviews of Modern Physics, 74(1):47–97.

[Bibel, 1987] Bibel, W. (1987).
Automated Theorem Proving.
Vieweg.
First edition 1982.

[Bibel and Otten, 2020] Bibel, W. and Otten, J. (2020).
From Schütte's formal systems to modern automated deduction.
In Kahle, R. and Rathjen, M., editors, *The Legacy of Kurt Schütte*, chapter 13, pages 215–249. Springer.

[Blanchette et al., 2016] Blanchette, J. C., Kaliszyk, C., Paulson, L. C., and Urban, J. (2016).
Hammering towards QED.
J. Formaliz. Reason., 9(1):101–148.

References II

[Boolos, 1987] Boolos, G. (1987).
A curious inference.
J. Philos. Logic, 16:1–12.

[Broido and Clauset, 2019] Broido, A. D. and Clauset, A. (2019).
Scale-free networks are rare.
Nature Communications, 10(1):1017.

[Carneiro, 2014] Carneiro, M. (2014).
Conversion of HOL Light proofs into Metamath.
CoRR, abs/1412.8091.

[Carneiro, 2020] Carneiro, M. (2020).
Metamath zero: Designing a theorem prover prover.
In Benzmüller, C. and Miller, B., editors, *CICM 2020*, volume 12236 of *LNCS (LNAI)*, pages 71–88. Springer.

[Carneiro et al., 2023] Carneiro, M., Brown, C. E., and Urban, J. (2023).
Automated theorem proving for Metamath.
In Naumowicz, A. and Thiemann, R., editors, *ITP 2023*, volume 268 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Clauset et al., 2009] Clauset, A., Shalizi, C. R., and Newman, M. E. J. (2009).
Power-law distributions in empirical data.
SIAM Review, 51(4):661–703.

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

[Dahn and Wernhard, 1997] Dahn, I. and Wernhard, C. (1997).
First order proof problems extracted from an article in the Mizar mathematical library.
In Bonacina, M. P. and Furbach, U., editors, *FTP'97*, RISC-Linz Report Series No. 97–50, pages 58–62, Linz. Joh. Kepler Univ.

References IV

[Denzinger and Schulz, 1994] Denzinger, J. and Schulz, S. (1994).
Analysis and Representation of Equational Proofs Generated by a Distributed Completion Based Proof System.
Seki-Report SR-94-05, Universität Kaiserslautern.
Revised September 1997.

[Eder, 1985] Eder, E. (1985).
Properties of substitutions and unification.
J. Symb. Comput., 1(1):31–46.

[Hetzl, 2012] Hetzl, S. (2012).
Applying tree languages in proof theory.
In Dediu, A.-H. and Martín-Vide, C., editors, *LATA 2012*, volume 7183 of *LNCS*, pages 301–312.

[Hindley, 1997] Hindley, J. R. (1997).
Basic Simple Type Theory.
Cambridge University Press.

References V

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

[Kaliszyk and Urban, 2015] Kaliszyk, C. and Urban, J. (2015).
Learning-assisted theorem proving with millions of lemmas.
J. Symb. Comput., 69:109–128.

[Larsson and Moffat, 2000] Larsson, N. J. and Moffat, A. (2000).
Off-line dictionary-based compression.
Proc. IEEE, 88(11):1722–1732.

[Lohrey, 2015] Lohrey, M. (2015).
Grammar-based tree compression.
In Potapov, I., editor, *DLT 2015*, volume 9168 of *LNCS*, pages 46–57. Springer.

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

[Loveland, 1968] Loveland, D. W. (1968).
Mechanical theorem proving by model elimination.
JACM, 15(2):236–251.

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

[Megill and Wheeler, 2019] Megill, N. and Wheeler, D. A. (2019).
Metamath: A Computer Language for Mathematical Proofs.
lulu.com, second edition.
Online https://us.metamath.org/downloads/metamath.pdf.

[Megill,] Megill, N. D.
Proof Explorer – Home Page – Metamath: A Theorem Sampler.
Online: `https://us.metamath.org/mpeuni/mmset.html#theorems`, accessed Jan 10, 2025.

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

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

[Newman, 2018] Newman, M. (2018).
Networks.
Oxford Univ. Press, second edition.

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

References VIII

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

[Rawson et al., 2023] Rawson, M., Wernhard, C., Zombori, Z., and Bibel, W. (2023).
Lemmas: Generation, selection, application.
In Ramanayake, R. and Urban, J., editors, *TABLEAUX 2023*, volume 14278 of *LNAI*, pages 153–174.

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

[Schulz, 1993] Schulz, S. (1993).
Analyse und Transformation von Gleichheitsbeweisen.
Projektarbeit in informatik, Fachbereich Informatik, Universität Kaiserslautern.
(German Language).

[Schönfinkel, 1924] Schönfinkel, M. (1924).
Über die Bausteine der mathematischen Logik.
Math. Ann., 92(3–4):305–316.

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

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

[Vyskocil et al., 2010] Vyskocil, J., Stanovský, D., and Urban, J. (2010).
Automated proof compression by invention of new definitions.
In Clarke, E. M. and Voronkov, A., editors, *LPAR-16*, volume 6355 of *LNCS*, pages 447–462. Springer.

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

[Wernhard, 2024] Wernhard, C. (2024).
Structure-generating first-order theorem proving.
In Otten, J. and Bibel, W., editors, *AReCCa 2023*, volume 3613 of *CEUR Workshop Proc.*, pages 64–83. CEUR-WS.org.

[Wernhard and Bibel, 2021] Wernhard, C. and Bibel, W. (2021).
Learning from Łukasiewicz and Meredith: Investigations into proof structures.
In Platzer, A. and Sutcliffe, G., editors, *CADE 28*, volume 12699 of *LNCS (LNAI)*, pages 58–75. Springer.

[Wernhard and Bibel, 2024] Wernhard, C. and Bibel, W. (2024).
Investigations into proof structures.
J. Autom. Reasoning, 68(24).

References XI

[Wernhard and Zombori, 2025] Wernhard, C. and Zombori, Z. (2025).
Mathematical knowledge bases as grammar-compressed proof terms: Exploring Metamath proof structures.
CoRR, abs/2505.12305.

[Wos, 2001] Wos, L. (2001).
Conquering the Meredith single axiom.
J. Autom. Reasoning, 27(2):175–199.