The *Metamath* Interface of *CD Tools* – Draft September 5, 2024–

Christoph Wernhard

info@christophwernhard.com

1 Introduction

Condensed detachment (CD) is an inference mechanism devised by Carew A. Meredith in the mid-1950s. It is based on detachment (modus ponens) combined with most general unification. Proof structures can be considered as terms with detachment steps represented by the binary function symbol D, which is applied to the subproofs of the major and the minor premise as arguments. These proof terms may straightforwardly be viewed as binary trees or as DAGs.

From the viewpoint of first-order logic, there is only a single unary predicate P involved, representing something like *provable*. The actual application formulas are expressed as first-order terms. Common traditional applications of CD are the investigation of axiomatizations of propositional logics. In particular, showing completeness by proving that from given such axioms, all axioms from a known complete set can be proven via CD steps. For example, that Łukasiewicz's single axiom for the implicational calculus is complete, since it entails a known axiomatization with three axioms known as *Simp*, *Peirce* and *Syll*, can be expressed as the following first-order entailment problem.

$$\begin{array}{ll} & \forall pq\,((\mathsf{P}(p \Rightarrow q) \wedge \mathsf{P}(p)) \rightarrow \mathsf{P}(q)) \wedge \\ & \forall pqrs\,\mathsf{P}((((p \Rightarrow q) \Rightarrow r) \Rightarrow ((r \Rightarrow p) \Rightarrow (s \Rightarrow p)))) \\ \models & \forall pq\,\mathsf{P}(p \Rightarrow (q \Rightarrow p)) \wedge \\ & \forall pq\,\mathsf{P}(((p \Rightarrow q) \Rightarrow p) \Rightarrow p) \wedge \\ & \forall pqr\,\mathsf{P}((p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))). \end{array}$$

Here $\Rightarrow$, representing implication for the modeled propositional logic, is a first-order function symbol. The first formula on the left side of the entailment $\models$ is a Horn clause that models condensed detachment in first-order logic.

Metamath is based on CD [5, 4]. Like the propositional logics in the mentioned CD applications, the logics and concepts of the mathematical applications are axiomatized on the user level with what technically can be considered as first-order terms. To handle quantification, *Metamath* joins the D proof construction function with a second one, G, for *condensed generalization* [5]. *Metamath* theorems have verified proofs expressed in terms of D (called `ax-mp` in `set.mm`, with arguments reversed to minor first, major second), G (called `ax-gen` in `set.mm`), user-specified axioms, and previously verified theorems. If references to previously verified theorems are replaced by their proofs, the ultimate result is an expanded proof that is just built from user-specified axioms, D and G.

CD Tools is a Prolog library for experimenting with condensed detachment in automated first-order theorem proving. Aside of various utilities related to CD it includes two specialized provers: SGCD, based on the enumeration of proof structures and combining goal- with axiom-driven search and CCS to experiment with the approach of compressed combinatory proof structures and DAG enumeration.

The *Metamath* interface of *CD Tools* allows to read-in a *Metamath* database file such as `set.mm`, also known as *Metamath Proof Explorer*, and convert it to a Prolog representation, which in turn offers various particular translations. In addition to formulas of axioms and theorems in the *Metamath* database also proofs are translated, optionally to forms that are directly supported by the utilities of *CD Tools* for CD proofs.

The interface is written from scratch in SWI-Prolog. For all formulas in `set.mm` it can be verified that the translations exactly correspond to the first-order translation obtained with another tool, *mm-hammer* [2], in a different workflow.

2 Syntax: Sequences versus Terms

Simple by design, *Metamath*'s substitution mechanism is not just used to implement semantics but also syntax. The syntax of the mathematical expressions is defined in a *Metamath* database from scratch at the user level. Mathematical expressions appear as *sequences* of constant and variable symbols, separated as tokens by spaces or line breaks. Substitution binds variables in a sequence to sequences, making them sub-sequences of the first sequence. Figure 1 shows syntax definitions and two axiom definitions from `set.mm`. The symbols beginning with `$` are built into the *Metamath* language. First some constants are declared: The parentheses, the symbol `->` for implication, the type symbol `wff` for *well-formed formula*, and the turnstile symbol `|-` representing *provable*. Then `ph`, `ps`, `ch`, mimicking ϕ, ψ, χ, are declared as variables of type `wff`. The line beginning with `wi $a` defines the syntax of implication as a sequence of an opening parenthesis, a variable, the `->` symbol, another variable and a closing parenthesis. It associates `wi` as a symbolic label for this.

The *CD Tools-Metamath* interface reads-in a *Metamath* database file and converts it to a sequence to so-called frames [4, Sect. 4.7.2], represented as Prolog terms. A frame gathers related statements. The Prolog representation offers a variety of options, reflecting to take account of frames and extended frames [4]. Figure 2 shows a Prolog representation of the frame for the `wi $a` statement from Fig. 1. The frame is for an `$a` statement (*axiomatic assertion*), reflected in the `a` functor. The implication expression appears at this stage also in Prolog as a sequence of tokens. The relevant variable declarations, `$f` statements, are contained in the second argument of the `frame` term.

For Prolog as well as for interfacing with automated provers a representation of logic expressions by *terms* rather than sequences is more common and convenient. We associate with each defined and named sequence template in

```
$c ( $.
$c ) $.
$c -> $.
$c wff $.
$c |- $.
$v ph $.
$v ps $.
$v ch $.
wph $f wff ph $.
wps $f wff ps $.
wch $f wff ch $.
wi $a wff ( ph -> ps ) $.
ax-1 $a |- ( ph -> ( ps -> ph ) ) $.
ax-2 $a |- ( ( ph -> ( ps -> ch ) ) ->
             ( ( ph -> ps ) -> ( ph -> ch ) ) ) $.
```

Fig. 1. Excerpts from `set.mm`: syntax definitions and two axioms.

```
frame(a(wi, wff, ['(', ph, ->, ps, ')']),
      [f(wph, wff, ph), f(wps, wff, ps)],
      [])
```

Fig. 2. Prolog representations of the frame for `wi`.

Metamath a function symbol whose arguments correspond to the variables in the order in which they appear in the sequence. Thus, the sequence `( ph -> ps )` is represented by the term `wi(ph, ps)`. From the frames we generate definite clause grammars (DCGs) that parse the sequences given as Prolog lists into such terms. Figure 3 gives an example of a DCG that is sufficient for the formulas of the axioms `ax-1` and `ax-2` shown in Figure 1. The formula of `ax-1`, for example, is parsed by this DCG to `wi(ph,wi(ps,ph))`. In a further step we convert the *Metamath* variables to Prolog variables, obtaining `wi(P,wi(Q,P))`[1]

```
wff(wi(A, B)) --> ['('], wff(A), [->], wff(B), [')'], !.
wff(var(A)) --> var_wff(A), !.
var_wff(ch) --> [ch], !.
var_wff(ph) --> [ph], !.
var_wff(ps) --> [ps], !.
```

Fig. 3. DCG rules for parsing implications in Prolog form.

[1] In Prolog syntax symbols starting with a capital letter are variables. Constants or function symbols (*atoms* and *functors* in Prolog jargon) starting with a capital letter have to be written in single quotes.

For parsing we use the DCG support that comes with SWI-Prolog. The parsing of *Metamath* sequences into nested terms is not entirely trivial because of potential ambiguity (but actually unambigous for `set.mm` [1]) and of size. For `set.mm` the total number of DCG rules would be too large to be efficiently handled naively with SWI-Prolog's DCG processing.

Our approach is to give the parsing of each statement two tries, one with a smaller DCG for evidently relevant symbols that allows very quick parsing, and, if this fails, a second larger DCG determined in more generous way via multisets of symbols. Also some heuristic ordering of DCG rules is applied. The expression parsing succeeds on the whole of `set.mm` with about 42500 theorems in reasonable time: On a contemporary notebook `set.mm` is read-in into the frame format in about 20 s. Formula parsing, proof uncompressing, conversion to versatile Prolog representations, and writing out the conversion results as Prolog facts then takes 100 s. Once these Prolog facts are compiled to SWI-Prolog's *quick load format*, they load in 0.5 s. Experiments verify that our parsed formulas are identical to the first-order formulas obtained with another tool, *mmhammer* [2], in a different workflow, modulo some systematic symbol renaming and the (semantically irrelevant) order of universal quantifiers upon different symbols.

Finally, we rewrite a few certain symbols that have a specially supported meaning in *CD Tools*. As shown in Table 1, this concerns implication, negation, the truth value constants and universal quantification. Thus, finally we obtain `P=>(Q=>P)` as formula of axiom `ax-1` and `(P=>(Q=>R))=>((P=>Q)=>(P=>R))` as formula of axiom `ax-2`.[2]

CD Tools	*Parsed Metamath*	*Metamath Source*
`P=>Q`	`wi(P,Q)`	`wi $a wff ( ph -> ps ) $.`
`n(P)`	`wn(P)`	`wn $a wff -. ph $.`
`truth`	`wtru`	`wtru $a wff T. $.`
`falsehood`	`wfal`	`wfal $a wff F. $.`
`forall(X,P)`	`wal(X,P)`	`wal $a wff A. x ph $.`

Table 1. Correspondence of symbols with predefined meaning in *CD Tools*.

Converting formulas in term form the other way around to *Metamath*'s sequence format is, based on information in the frame data structure, an easy task, which is also supported by our system.

[2] Prolog permits to declare function symbols as infix, which we used for `=>`.

3 Condensed Detachment, Proof Terms, The Most General Theorem (MGT)

Metamath's proof system is an extension of Carew A. Meredith's *condensed detachment (CD)*, which provides an elegant, convenient and powerful systematic way to represent "proof objects" as terms. We call these *proof terms* to distinguish them from *formulas* or *formula terms*. In this section we present aspects of *Metamath*'s proof system that bridge to *CD Tools*, our implemented Prolog-based tool to process and search for CD proofs in general. For more information on CD in general we refer to [7, 9].

We consider terms of two classes, *formulas*, specified in Section 3.1 below and *proof terms* with a basic form, DG-terms, specified in Section 3.2 and a generalized form, DGH-terms, specified in Section 4.2 below.

3.1 Formulas

From the viewpoint of first-order logic *Metamath* uses just a single unary predicate `|-` for *provable*, called `P` in Sect. 1. *Metamath formulas* (*wffs* in *Metamath* jargon) are technically first-order *terms* that appear as arguments of the *provable* predicate. In the following, unless specially indicated, we use *formula* to refer to formulas in this sense, that is, technically, first-order terms.

Formulas, are built from variables and function symbols. Two specific such function symbols, shown below, have a special meaning built in into the proof system. Other symbols are introduced on the user level with axioms characterizing their properties.

Variables are partitioned into *formula variables* and *individual variables*. Both are technically handled in exactly the same ways, but on the user level it is ensures that each argument position of a function symbol is associated with exactly one of these two variable sorts.

The built-in function symbols are the binary symbol $\Rightarrow$, written in infix notation, for *implication* between two formulas, and the binary function symbol $\forall$, for *universal quantification* with an individual variable and a formula as arguments.

3.2 DG-Terms: Basic Proof Terms

DG-terms are terms that represent a proof. They are built from constants called *axiom labels* that have an associated axiom, a formula, and two function symbols, the binary function symbol `D` for *condensed detachment* and the unary function symbol `G` for *condensed generalization* [5]. (The name *DG-terms* relates to *D-terms* used in [9] for proof terms with `D` as the only function symbol.) We call a mapping that maps each axiom label (in some implicitly given set of axiom labels) to a formula an *axiom assignment*. In Section 4.2 we generalize DG-terms to DGH-terms.

3.3 The *Proves* Relation

DG-terms and formulas are linked through the *proves* relation, which expresses that a DG-term provides a proof of a formula. We assume a fixed axiom assignment.

Definition 1. The *proves* relation between DG-terms and formulas is defined inductively as the smallest relation such that for all DG-terms A, B, formulas P, Q, formula substitutions σ and variables x it holds that

1. c *proves* $P\sigma$ if c is an axiom label with associated formula P.
2. $\mathsf{D}(A, B)$ *proves* $Q\sigma$ if A *proves* $P \Rightarrow Q$ and B *proves* P.
3. $\mathsf{G}(A)$ *proves* $\forall(x, P)\sigma$ if A *proves* P.

3.4 The Most General Theorem (MGT) of a DG-Term

We assume a fixed axiom assignment. If a DG-term proves a formula P, then it also proves all instances $P\sigma$ of P. If a DG-term A proves some formula at all, then there is a most general formula P that is proven by A, called *the most general theorem (MGT)* of A with respect to the axiom assignment. That is, whenever A proves Q then Q is an instance of the most general theorem P, i.e., there is a substitution σ such that $Q = P\sigma$. The most general theorem of a DG-term with respect to an axiom assignment is unique up to systematic renaming of variables.

3.5 Computing the MGT – An Abstract Prolog View

Definition 1 can fairly straightforwardly be turned into a simple Prolog program that computes the MGT of a given DG-term, shown in Fig. 5.

```
mgt(c1, Axiom1).
...
mgt(ck, Axiomk).
mgt(d(A,B), Q) :- mgt(A, (P=>Q)), mgt(B, P).
mgt(g(A), forall(X,P)) :- mgt(A, P).
```

Fig. 4. Definition of Prolog predicate `mgt`/2 to compute the MGT of a given DG-term.

We assume that the underlying Prolog system is configured to perform unification with the occurs check.[3] The first k clauses of the program are facts, one

[3] Originally motivated by performance reasons, Prolog systems by default omit the so-called occurs check, to the effect that they succeed in unifying terms that are actually not unifiable. Modern Prolog systems can be configured to activate the occurs check in general and provide library predicates to perform the occurs check in specific situations.

for each axiom label c_i, with the assigned axiom, a formula, as second argument. Each time such a fact is accessed, Prolog returns the formula as a copy with fresh variables. The clause for D then follows. The logical variable P has two occurrences in its body. In an invocation of the clause, Prolog quietly applies to all arguments of the clause the most general unifier that equates the values of P obtained in both recursive calls to mgt. The last clause is for G. Each time it is invoked, Prolog generates a fresh variable X.

If the DG-term is given as input argument, the predicate mgt returns the MGT as value of the formula argument. Here is an example, where the formula associated with 'ax-1' is P=>(Q=>P).

```
?- mgt(d('ax-1', 'ax-1'), F).
F = (P=>(Q=>(R=>Q))).
```

If the given DG-term has no MGT for the given axiom assignment, then the invocation of the mgt predicate fails, since Prolog's unification fails at some point. Here is an example of an invocation that fails. The axiom assigned to 'ax-2' is (P=>(Q=>R))=>((P=>Q)=>(P=>R)).

```
?- mgt(d(d('ax-2', 'ax-2'), 'ax-1'), F).
false.
```

From the view of type theory, computing the MGT of a DG-term (without occurrences of G) is computing the *principal type* an expression, where D represents application and the axiom labels are constants with the associated formula as principal type.

So far, we used our program mgt with the DG-term as input argument and the formula as output argument. But, moreover, the program can also be used as a verifier with both arguments as inputs. If the formula argument is ground, then it succeeds if and only if it is an instance of the MGT. Here is an example.

```
?- mgt(d('ax-1', 'ax-1'), (p=>((q=>r)=>(p=>(q=>r))))).
true.
```

Using the program as a prover with a ground formula as input and the DG-term as output is in general not possible, as Prolog's unrestricted depth-first regime then might quickly get lost in an infinite branch.

4 The Horn MGT and Horn Lemma Labels

We smoothly generalize the framework developed in Sect. 3 such that it covers *Metamath*'s way to express theorems and to reference theorems as lemmas in proofs.

We generalize our notion of *formula* to *Horn clause*, an implication with a body, a conjunction of formulas and a head, a single formula. (We implicitly assume that the Horn clause has a head, i.e., is a definite clause.) As an example of a Horn clause in this sense consider $((p \Rightarrow q) \wedge p) \rightarrow q$. From a first-order perspective, such a Horn clause corresponds to the definite first-order clause

obtained by wrapping all atoms (which are terms in the first-order view) in the *provable* predicate.

4.1 The Horn MGT of a DG-Term with Variables

We now consider a generalization of MGT computation for DG-terms $A[V_1, \ldots, V_n]$ in which distinct *variables* $v_1 \ldots, v_n$ occur. the *Horn MGT* of $A[v_1, \ldots, v_n]$ with respect to a given axiom assignment is the most general Horn clause $(P_1 \wedge \ldots \wedge P_n) \rightarrow Q$ with the property that for all substitutions σ it holds that if $A_1, \ldots, A_n$ are DG-terms such that

$$A_i \textit{ proves } P_i\sigma, \text{ for all } i \in \{1, \ldots, n\},$$

then

$$A[A_1, \ldots, A_n] \textit{ proves } Q\sigma,$$

where $A[A_1, \ldots, A_n]$ denotes $A[v_1, \ldots, v_n]$ with variables v_i replaced by the respective DG-terms A_i.

Computation of the Horn MGT can be implemented with a simple modification of the `mgt` program (Fig. 5) by adding the following clause as first clause.

```
mgt(V, P) :- var(V), !, V = P.
```

Fig. 5. Additional first clause to generalize the Prolog program of Fig. 5 for computation of the Horn MGT.

If instead of an DG-term a variable `V` is encountered, Prolog's cut (`!`) commits the clause, which means that the other clauses are no longer considered for this invocation of the predicate, and the variable `V` is unified with the formula `P` associated with `V`'s position in the DG-term. It effects, speaking informally, that the variable `V` now records the requirements on the formulas proven by a proof term inserted at that position. Our simple implementation does not work if a variable `V` has multiple occurrences in the input DG-term (once the first occurrence of `V` is bound to some formula, the second occurrence is no longer a variable).[4]

The output Horn clause is represented by the formula output argument as head and the instantiated variables in the DG-term as body. Here is an example.

```
?- mgt(d('ax-1', d(V, 'ax-1')), F).
V = ((P=>(Q=>P))=>R),
F = (S=>R).
```

From the output bindings of `V` and `F` we read off that the Horn MGT of $\mathsf{D}(\mathsf{ax\text{-}1}, \mathsf{D}(v, \mathsf{ax\text{-}1}))$ is the Horn clause

$$((p \Rightarrow (q \Rightarrow p)) \Rightarrow r) \rightarrow (s \Rightarrow r).$$

[4] The actual implementation in *CD Tools* does not suffer from that restriction.

We now replace v with ax-1 in the DG-term and compute the MGT.

```
?- mgt(d('ax-1', d('ax-1', 'ax-1')), F).
F = (P=>(Q=>(R=>(S=>R)))).
```

This relates to the above Horn MGT as follows. The MGT of ax-1 is the associated axiom, $p' \Rightarrow (q' \Rightarrow p')$, if written with fresh variables. Hence ax-1 also proves the instance $(p' \Rightarrow (q' \Rightarrow p'))\gamma$ where $\gamma = \{p' \mapsto (p \Rightarrow (q \Rightarrow p))\}$. This instance is identical to $((p \Rightarrow (q \Rightarrow p)) \Rightarrow r)\sigma$, where $\sigma = \{r \mapsto (q' \Rightarrow (p \Rightarrow (q \Rightarrow p)))$, an instance of the body atom of the Horn MGT. By the characteristics of Horn MGTs it follows that if in $\mathsf{D}(\text{ax-1}, \mathsf{D}(v, \text{ax-1}))$ the variable v is replaced by ax-1, then the resulting DG-term $\mathsf{D}(\text{ax-1}, \mathsf{D}(\text{ax-1}, \text{ax-1}))$ proves the head of the Horn MGT under substitution σ, that is, $(s \Rightarrow r)\sigma$. This is $s \Rightarrow (q' \Rightarrow (p \Rightarrow (q \Rightarrow p)))$, the MGT of $\mathsf{D}(\text{ax-1}, \mathsf{D}(\text{ax-1}, \text{ax-1}))$, a systematic renaming of the MGT as computed above.

4.2 DGH-Terms: Proof Terms with Horn Lemma Labels

We now generalize axiom labels and axiom assignments (Sect. 3.2) to *Horn lemma labels* and *Horn lemma assignments*. A Horn lemma label is a function symbol with an associated arity ≥ 0. A Horn lemma assignment maps each Horn lemma label with arity n to a Horn clause with body length n.

We call proof terms in which the Horn lemma labels may occur as n-ary function symbols DGH-terms. For DGH-terms the definition of *proves* is generalized as follows.

Definition 2. We generalize the definition of *proves* (Def. 1) for DGH-terms. The three cases of definition 1 now apply to DGH-terms instead of just DG-terms and a fourth case is added, for DGH-terms $A_1, \ldots, A_n$, formulas $P_1, \ldots, P_n, Q$ and formula substitutions σ, θ.

4. $f(A_1, \ldots, A_n)$ *proves* $Q\theta\sigma$ if f is a Horn lemma label with associated clause $(P_1 \wedge \ldots \wedge P_n) \rightarrow Q$ and for all $i \in \{1, \ldots, n\}$ it holds that A_i *proves* $P_i\theta$.

Note that this case 4 of the definition of *proves* actually could represent all the other cases: Case 1 can be represented by 0-ary Horn lemma labels f, case 2 by an f with Horn clause $((p \Rightarrow q) \wedge p) \rightarrow q$ and case 3 by an f with Horn clause $p \rightarrow \forall(x, p)$.

The notion of *Horn MGT* of DG-terms with variables discussed in Section 4.1 straightforwardly generalizes to DGH-terms. The Horn MGT of a DGH-term with variables is a Horn clause, as for a DG-term with variables.

4.3 Expanding Horn Lemma Labels

Only very few of the axioms specified in `set.mm` correspond to Horn clauses with a non-empty body. Notably ax-mp and ax-gen, representing the D and G inference rules, the basis of *Metamath* as specified by Megill [5]. We represent these by the proof term functions D and G, where for D the argument order is

subproof of *major* premise followed by subproof of *minor* premise, the reverse order of `ax-mp`.[5]

The formulas of the about 42,500 theorems in `set.mm` are Horn clauses, in 59% of the cases with a non-empty body. Thus `set.mm` provides a Horn lemma assignment for 42,500 Horn lemma labels. Moreover, each theorem in `set.mm` has a verified proof that might refer to axioms and to previously proven theorems. These proofs translate directly to DGH-terms with references to a previously proven theorem expressed by its Horn lemma label as function symbol. The DGH-term that proves a theorem whose formula is a Horn clause with body length n has n variables. The theorem formula stated in the *Metamath* database is the Horn MGT of this DGH-term with variables or, for about 10% of the theorems in `set.mm`, a strict instance of this Horn MGT.

The Horn lemma labels in a DGH-term term can be eliminated by rewriting terms $f(A_1, \ldots, A_n)$ with f a Horn lemma label to the proof of the respective Horn lemma, with variables instantiated by DGH-terms $A_1, \ldots, A_n$. Exhaustive rewriting leads to a pure DG-term with just unit axioms. For theorems intended to express applications rather than just intermediate steps these expanded proof trees are quite large. For `peano3`, an example of such a theorem with a relatively small proof, the tree has 7.42×10^{22} inner nodes. The DAG representation, however, has only $3,555$ inner nodes.

Starting from the pure DG-term, the representation with Horn lemmas can be considered as a tree compression that is stronger than the DAG representation. An related way of structure-based proof compression is achieved through the involvement of combinators, as indicated by example in Sect. 6. Advanced forms of compression and the relationship to human-created lemmas is shown as an application of this *CD Tools*/Metamath interface [10].

5 Proof Representations

We give some examples how *Metamath* proofs can be represented in the Prolog environment with our interface.

5.1 A Basic Example

Figure 6 shows some excerpts from `set.mm`.

[5] There are four further axioms in `set.mm` that have a Horn clause with non-empty body as formula: `df-cleq`, `df-clel`, `ax-prv1` and `ax-tgoldbachgt`. Special handling for these in our Prolog representation is deferred to future work.

```
ax-1 $a |- ( ph -> ( ps -> ph ) ) $. ax-2 $a |- ( ( ph -> ( ps -> ch ) ) -> (
( ph -> ps ) -> ( ph -> ch ) ) ) $.

a1i.1 $e |- ph $.
a1i $p |- ( ps -> ph ) $=
  wph wps wph wi a1i.1 wph wps ax-1 ax-mp $.

a2i.1 $e |- ( ph -> ( ps -> ch ) ) $.
a2i $p |- ( ( ph -> ps ) -> ( ph -> ch ) ) $=
  wph wps wch wi wi wph wps wi wph wch wi wi a2i.1 wph wps
  wch ax-2 ax-mp $.

mpd.1 $e |- ( ph -> ps ) $.
mpd.2 $e |- ( ph -> ( ps -> ch ) ) $.
mpd $p |- ( ph -> ch ) $=
  wph wps wi wph wch wi mpd.1 wph wps wch mpd.2 a2i ax-mp $.

mpi.1 $e |- ps $.
mpi.2 $e |- ( ph -> ( ps -> ch ) ) $.
mpi $p |- ( ph -> ch ) $=
  wph wps wch wps wph mpi.1 a1i mpi.2 mpd $.
```

Fig. 6. Excerpts from `set.mm`.

Axioms `ax-1` and `ax-2` (also shown in Fig. 1) are two of three axioms that formalize the propositional calculus, known as *Simp* and *Frege*, respectively [7]. Subsequently, four Horn lemmas, proven theorems, are stated: *inference introducing an antecedent* (`a1i`), *inference distributing an antecedent* (`a2i`), *modus ponens deduction* (`mpd`) and *a nested modus ponens inference* (`mpi`). For each theorem the `$e` statements provide the body of the Horn clause and the `$p` statement the head. The `$p` statements also include the proof of the theorem, shown here in *Metamath*'s *normal* form, which is not compressed and in reverse Polish notation.

Figure 7 shows the formulas of the axiom and theorem statements from Fig. 6 translated to Prolog. Horn clauses $(F_1 \wedge \ldots \wedge F_n) \rightarrow F$ are written there as $F_1, \ldots, F_n$ `-:` F.

```
ax-1: P=>(Q=>P)
ax-2: (P=>(Q=>R))=>((P=>Q)=>(P=>R))
a1i:  P -: Q=>P
a2i:  P=>(Q=>R) -: (P=>Q)=>(P=>R)
mpd:  P=>Q, P=>(Q=>R) -: P=>R
mpi:  P, Q=>(P=>R) -: Q=>R
```

Fig. 7. Formulas of the axiom and theorem statements from Fig. 6 translated to Prolog.

Figures 8 and 9 show proofs of the theorems from Fig. 6. They are displayed by the `metamath.exe` processor in the so-called *Lemmon-style* [4] as numbered steps with two columns: the left colum showing the proof structure with references to previous steps and the right column showing the proven formula. Figure 8 show a full proof, which includes "syntactic" steps that perform expression construction. Figure 9 shows several proofs in *essential* form, that is, eliding these syntactic steps, which is the default view in `metamath.exe`.

```
MM> show proof a1i /lemmon /all
1 wph            $f wff ph
2 wps            $f wff ps
3 wph            $f wff ph
4 2,3 wi         $a wff ( ps -> ph )
5 a1i.1          $e |- ph
6 wph            $f wff ph
7 wps            $f wff ps
8 6,7 ax-1       $a |- ( ph -> ( ps -> ph ) )
9 1,4,5,8 ax-mp  $a |- ( ps -> ph )
```

Fig. 8. Proof of theorem `a1i` from `set.mm` in Lemmon-style as full proof, including "syntactic" steps that perform expression construction.

Figure 10 shows how the *Metamath* proofs displayed in Fig. 9 translate into proof macro definitions as rules in Prolog syntax. The left hand side is the defined Horn lemma label as template with variable arguments. The right hand side is defining proof term, a DGH-term, with variables. Similar to the *essential* proof presentation in `metamath.exe` the "syntactic" proof steps are elided.

Proof macro definitions as in Fig. 10 may be read as rewrite rules for proof terms. By rewriting the rules of this figure exhaustively, we obtain the rules shown in Fig. 11, where the defining right sides of all rules are fully expanded to DG-terms with variables.

Another way to read a set of proof macro definitions as in Fig. 10 as a tree grammar that specifies in compressed form – with non-terminals that may contain variables – a set of DG-terms or patterns of DG-terms (i.e., DG-terms with variables).

Figures 10 and 11 suggest two different ways of determining the Horn MGT from the proof of a given *Metamath* theorem: (1) In a "shallow" way from the DGH-term as given in the *Metamath* database on the basis of the Horn lemmas and axioms directly mentioned there. (2) In a "deep" way from the DG-term obtained after exhaustively expanding the Horn axiom labels. The stated theorem as Horn clause is always an instance of the shallow Horn MGT, which in turn is always an instance of the deep Horn MGT. In both cases the instance relationship may be strict. For the examples considered in this section so far, all three Horn clauses are identical, up to systematic renaming of variables.

```
MM> show proof a1i /lemmon /renumber
1 a1i.1          $e |- ph
2 ax-1           $a |- ( ph -> ( ps -> ph ) )
3 1,2 ax-mp      $a |- ( ps -> ph )

MM> show proof a2i /lemmon /renumber
1 a2i.1          $e |- ( ph -> ( ps -> ch ) )
2 ax-2           $a |- ( ( ph -> ( ps -> ch ) ) ->
                         ( ( ph -> ps ) -> ( ph -> ch ) ) )
3 1,2 ax-mp      $a |- ( ( ph -> ps ) -> ( ph -> ch ) )

MM> show proof mpd /lemmon /renumber
1 mpd.1          $e |- ( ph -> ps )
2 mpd.2          $e |- ( ph -> ( ps -> ch ) )
3 2 a2i          $p |- ( ( ph -> ps ) -> ( ph -> ch ) )
4 1,3 ax-mp      $a |- ( ph -> ch )

MM> show proof mpi /lemmon /renumber
1 mpi.1          $e |- ps
2 1 a1i          $p |- ( ph -> ps )
3 mpi.2          $e |- ( ph -> ( ps -> ch ) )
4 2,3 mpd        $p |- ( ph -> ch )
```

Fig. 9. Proofs of theorem `mpi` and the theorems it depends on, i.e., `mpd`, `a2i`, and `a1i`, all from `set.mm`. Displayed in Lemmon-style and in essential form, that is, with eliding "syntactic" steps that perform expression construction.

Theorem `pm2.86d` is an example where the theorem statement is a strict instance of the shallow and the deep Horn MGT, which are both identical up to renaming of variables. In Prolog notation the theorem statement is

```
P=>((Q=>R)=>(Q=>S)) -: P=>(Q=>(R=>S))
```

while the MGT is

```
P=>((Q=>R)=>(S=>T)) -: P=>(S=>(R=>T)).
```

```
a1i(X) -> d('ax-1', X)
a2i(X) -> d('ax-2', X)
mpd(X, Y) -> d(a2i(Y), X)
mpi(X, Y) -> mpd(a1i(X), Y)
```

Fig. 10. *Metamath* proofs shown in Fig. 9 converted to proof macro definitions in Prolog syntax.

```
a1i(X) -> d('ax-1', X)
a2i(X) -> d('ax-2', X)
mpd(X, Y) -> d(d('ax-2', Y), X)
mpi(X, Y) -> d(d('ax-2', Y), d('ax-1', X))
```

Fig. 11. Proof macro definitions from Fig. 10 after exhaustively rewriting right sides to DG-terms.

5.2 Quantification: G in Proof Terms

Figure 12 shows syntax definitions, the definition of an axiom and the definitions of three theorems involving universal quantification from `set.mm`.

```
$c A. $.
$c setvar $.
$v x $.
vx $f setvar x $.
wal $a wff A. x ph $.

ax-4 $a |- ( A. x ( ph -> ps ) -> ( A. x ph -> A. x ps ) ) $.

alim $p |- ( A. x ( ph -> ps ) -> ( A. x ph -> A. x ps ) ) $=
  wph wps vx ax-4 $.

mpg.1 $e |- ( A. x ph -> ps ) $.
mpg.2 $e |- ph $.
mpg $p |- ps $=
  wph vx wal wps wph vx mpg.2 ax-gen mpg.1 ax-mp $.

alimi.1 $e |- ( ph -> ps ) $.
alimi $p |- ( A. x ph -> A. x ps ) $=
  wph wps wi wph vx wal wps vx wal wi vx wph wps vx alim alimi.1 mpg $.
```

Fig. 12. Excerpts from `set.mm`: syntax definitions, an axiom and three theorems involving universal quantification.

First syntax for universal quantification is defined with `A.`, mimicking ∀. Then `setvar` is declared as a constant representing the *individual variable type*, which contrasts with the *wff* type, and `x` is declared as an individual variable. The *axiom of quantified implication*, `ax-4` is restated as theorem `alim`, according to the comments in `set.mm` for labeling consistency. The two further defined theorems are `mpg`, *modus ponens combined with generalization*, and `alimi`, *inference quantifying both antecedent and consequent.*

Figure 13 shows the formulas of the axiom and theorem statements from Fig. 12 translated to Prolog.

```
ax-4:  forall(X, (P=>Q))=>(forall(X, P)=>forall(X, Q))
alim:  forall(X, (P=>Q))=>(forall(X, P)=>forall(X, Q))
mpg:   forall(X, P)=>Q, P -: Q
alimi: P=>Q -: forall(X, P)=>forall(X, Q)
```

Fig. 13. Formulas of the axiom and theorem statements from Fig. 12 translated to Prolog.

Figure 14 shows proofs of the theorems in Fig. 12 in essential form.

```
MM> show proof alim /lemmon /renumber
1 ax-4           $a |- ( A. x ( ph -> ps ) -> ( A. x ph -> A. x ps ) )

MM> show proof mpg /lemmon /renumber
1 mpg.2          $e |- ph
2 1 ax-gen       $a |- A. x ph
3 mpg.1          $e |- ( A. x ph -> ps )
4 2,3 ax-mp      $a |- ps

MM> show proof alimi /lemmon /renumber
1 alim           $p |- ( A. x ( ph -> ps ) -> ( A. x ph -> A. x ps ) )
2 alimi.1        $e |- ( ph -> ps )
3 1,2 mpg        $p |- ( A. x ph -> A. x ps )
```

Fig. 14. Proofs of theorem `alimi` and the theorems it depends on, i.e., `mpg` and `alim`, all from `set.mm`. Displayed in Lemmon-style and in essential form, that is, with eliding "syntactic" steps that perfom expression construction.

Figure 15 shows how the *Metamath* proofs displayed in Fig. 14 translate into proof macro definitions as rules in Prolog syntax.

```
alim -> 'ax-4'
mpg(X, Y) -> d(X, g(Y))
alimi(X) -> mpg(alim, X)
alimi(X) -> d('ax-4', g(X))
```

Fig. 15. *Metamath* proofs shown in Fig. 14 converted to proof macro definitions in Prolog syntax.

6 Combinator Representation

We consider combinators in the sense of Schönfinkel [6] and Curry [3]. Identifying D with application, we can define combinators **B** and **C** as

$$\mathbf{B} = \lambda x.\lambda y.\lambda z.\mathsf{D}(x, \mathsf{D}(y, z)).$$
$$\mathbf{C} = \lambda x.\lambda y.\lambda z.\mathsf{D}(\mathsf{D}(x, z), y).$$

If we permit combinators in proof terms, we can reformulate proof terms as shown for the two examples in Fig. 16.

```
mpd(X, Y) -> d(d(d(c, 'ax-2'), X), Y)
mpi(X, Y) -> d(d(d(d(b, d(c, 'ax-2')), 'ax-1'), X), Y)
```

Fig. 16. The DG-terms for `mpd` and `mpi` reformulated with combinators such that their variables appear as rightmost symbols in the same order in which they appear as arguments of the lemma label. Here `b` and `c` are Prolog symbols representing **B** and **C**.

The proofs of `mpd` and `mpi` can be considered by η-conversion as variable-free. The lemma labels `mpd` and `mpi` then just refer to DG-terms with combinators. If we note application by juxtaposition, this can be expressed as

$$\mathsf{mpd} = \mathbf{C}\mathsf{ax\text{-}2}.$$
$$\mathsf{mpi} = \mathbf{B}(\mathbf{C}\mathsf{ax\text{-}2})\mathsf{ax\text{-}1}.$$

The combinator representation effects that multiple occurrences of a lemma like `mpd` or `mpi` in a proof are shared in its DAG representation, even if lemma labels are not explicitly used. Subterms such as **C**ax-2 (i.e., `d(c, 'ax-2')` in the Prolog representation) or **B**(**C**ax-2)ax-1 (i.e., `d(d(b, d(c, 'ax-2')), 'ax-1')` in the Prolog representation) are in the DAG shareable subgraphs [8].

7 Current Limitations

We have not yet addressed conversion of DGH-terms to the *Metamath* proof format, which seems to require adding syntactic steps that are elided in *Metamath*'s *normal* proof presentation.

A few axioms and related items, some of them of central importance, seem to need special handling. This concerns `df-cleq`, `weq`, `wceq`, `wcel`, `df-clel`, `wel`,`ax-prv1` and `ax-tgoldbachgt`.

There is so far no support for checking or enforcing the so-called *disjoint variable restrictions*, but information about them is available in the Prolog representation.

References

1. Carneiro, M.: Grammar ambiguity in set.mm (2013), http://us.metamath.org/downloads/grammar-ambiguity.txt
2. Carneiro, M., Brown, C.E., Urban, J.: Automated theorem proving for Metamath. In: Naumowicz, A., Thiemann, R. (eds.) ITP 2023. LIPIcs, vol. 268, pp. 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ITP.2023.9
3. Curry, H., Feys, R.: Combinatory Logic, vol. I. North-Holland (1958)
4. Megill, N., Wheeler, D.A.: Metamath: A Computer Language for Mathematical Proofs. lulu.com, second edn. (2019), online https://us.metamath.org/downloads/metamath.pdf
5. Megill, N.D.: A finitely axiomatized formalization of predicate calculus with equality. Notre Dame J. of Formal Logic **36**(3), 435–453 (1995). https://doi.org/10.1305/ndjfl/1040149359
6. Schönfinkel, M.: Über die Bausteine der mathematischen Logik. Math. Ann. **92**(3–4), 305–316 (1924). https://doi.org/10.1007/BF01448013
7. Ulrich, D.: A legacy recalled and a tradition continued. J. Autom. Reasoning **27**(2), 97–122 (2001). https://doi.org/10.1023/A:1010683508225
8. Wernhard, C.: Generating compressed combinatory proof structures — an approach to automated first-order theorem proving. In: Konev, B., Schon, C., Steen, A. (eds.) PAAR 2022. CEUR Workshop Proc., vol. 3201. CEUR-WS.org (2022), https://arxiv.org/abs/2209.12592
9. Wernhard, C., Bibel, W.: Investigations into proof structures. J. Autom. Reasoning (2024), to appear, preprint https://arxiv.org/abs/2304.12827
10. Wernhard, C., Zombori, Z.: Exploring Metamath proof structures (extended abstract). In: Douglas, M.R., Hales, T.C., Kaliszyk, C., Schulz, S., Urban, J. (eds.) AITP 2024 (Informal Book of Abstracts) (2024)