The nanoCoP 2.0 Connection Provers for Classical, Intuitionistic and Modal Logics

Jens Otten

Department of Informatics, University of Oslo, Norway
jeotten@ifi.uio.no

Abstract. This paper introduces the *full* versions of the *non-clausal* connection provers nanoCoP for first-order classical logic, nanoCoP-i for first-order intuitionistic logic and nanoCoP-M for several first-order multimodal logics. The enhancements added to the core provers include several techniques to improve performance and usability, such as a strategy scheduling and the output of a detailed non-clausal connection proof for all covered logics. Experimental evaluations for all provers show the effectiveness of the integrated optimizations.

1 Introduction

The *non-clausal* connection calculus for classical logic [18] generalizes the *clausal* connection calculus [3, 4, 24] to arbitrary first-order formulae. By directly dealing with non-clausal formulae, a translation into a (disjunctive or conjunctive) clausal form can be avoided. Instead, the structure of the original input formula is preserved throughout the proof search. The non-clausal calculus combines the advantages of more natural (non-clausal) sequent and tableau calculi with the more systematic and goal-oriented proof search of connection calculi. Recently, the non-clausal connection calculus has been adapted and extended to first-order *intuitionistic* logic and several first-order *modal* logics [22]. This has been achieved by adding prefixes and a specialized prefix unification algorithm that captures the Kripke semantics of these non-classical logics.

Automated theorem provers that are based on these non-clausal calculi have been introduced as well: the nanoCoP (= *na*tural *no*n-clausal *Co*nnection *P*rover) series of provers for classical logic [20, 21], first-order intuitionistic logic and several first-order modal logics [22]. While already the basic implementations of the non-clausal core calculi show a decent performance, these basic provers were missing important features in terms of performance and usability, e.g., output of readable connection proofs and further proof search optimizations, such as strategy scheduling, a technique that consecutively tries a set of different strategies when searching for a proof.

After a brief introduction of the non-clausal connection calculi (Section 2), the paper presents the most recent versions of the non-clausal connection provers nanoCoP, nanoCoP-i and nanoCoP-M together with the (minimalistic) source code of the Prolog core prover (Section 3). The main enhancements are the integration of several *lean* proof search optimizations, a strategy scheduling, the output of readable connection proofs, the extension of nanoCoP-M to multimodal logics, and a better support to run the provers on different Prolog platforms. The paper also presents a comprehensive practical evaluation of all provers on the standard problem libraries (Section 4).

2 Non-clausal Connection Calculi

A *(first-order) formula* (denoted by F, G, H) is built up from atomic formulae, the connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and the standard first-order quantifiers $\forall$ and $\exists$. A *(first-order) modal formula* might also include the modal operators $\Box$ and $\Diamond$. An *atomic formula* (denoted by A) is built up from predicate symbols (P, Q), function symbols (f, g) and term variables (x, y). A *literal* L has the form A or $\neg A$.

In the *clausal* connection calculus a matrix is a set of clauses, where a clause is a set of literals. The *non-clausal* connection calculus works on *non-clausal* matrices, in which a matrix M is a set of clauses and a clause C is a set of literals L and (sub)matrices. It can be seen as a representation of a formula in negation normal form.

For a formula F and polarity $pol \in \{0,1\}$, the *classical non-clausal matrix* $M(F^{pol})$ of F^{pol} is defined inductively according to Table 1 ($: p$ and the last two lines are to be ignored). x^* is a new term variable, t^* is the Skolem term $f^*(x_1,\ldots,x_n)$ in which f^* is a new function symbol and $x_1,\ldots,x_n$ are all free variables in $(\forall xG)^0 : p$ or $(\exists xG)^1 : p$. The *(classical) non-clausal matrix* $M(F)$ of F is the classical non-clausal matrix $M(F^0)$.

In the *graphical representation* of a non-clausal matrix, its clauses are arranged horizontally, its literals and matrices are arranged vertically. A *connection* is a set $\{A_1{}^0, A_2{}^1\}$ of literals with the same predicate symbol but different polarities. A *term substitution* σ_T assigns terms to variables. A connection is σ_T*-complementary* iff $\sigma_T(A_1) = \sigma_T(A_2)$.

The axiom and the four rules of the *non-clausal connection calculus* are given in Figure 1 (again, $: p_1$ and $: p_2$ are to be ignored). Compared to the formal *clausal* connection calculus [23], the extension rule is restricted to certain extension clauses and a *decomposition rule* is added that splits subgoal clauses; see [18, 20] for details.

A clause C in a matrix M is an *extension clause (e-clause) of M with respect to* a set of literals *Path* iff either (a) C contains a literal of *path*, or (b) C is α-related to (i.e. occurs besides) all literals of *path* occurring in M, and if C has a parent clause, it contains a literal of *path*. In the clause β*-clause*$_{L_2}(C_2)$, the literal L_2 and all clauses that are α-related to (occur besides) L_2 are deleted from C_2, as these clauses do not need to be considered in the new subgoal clause in the premise of the extension rule. A *copy of the clause C* in the matrix M is made by renaming all *free variables* in C. $M[C_1 \backslash C_2]$ denotes the matrix M, in which the clause C_1 is replaced by the clause C_2.

Table 1. The definition of the (prefixed) non-clausal matrix for classical and modal logic

type	$F^{pol} : p$	$M(F^{pol} : p)$
atomic	$A^0 : p$	$\{\{A^0 : p\}\}$
α	$(G \wedge H)^1 : p$	$\{\{M(G^1 : p)\}\}, \{\{M(H^1 : p)\}\}$
	$(G \vee H)^0 : p$	$\{\{M(G^0 : p)\}\}, \{\{M(H^0 : p)\}\}$
	$(G \Rightarrow H)^0 : p$	$\{\{M(G^1 : p)\}\}, \{\{M(H^0 : p)\}\}$
β	$(G \wedge H)^0 : p$	$\{\{M(G^0 : p), M(H^0 : p)\}\}$
	$(G \vee H)^1 : p$	$\{\{M(G^1 : p), M(H^1 : p)\}\}$
	$(G \Rightarrow H)^1 : p$	$\{\{M(G^0 : p), M(H^1 : p)\}\}$
ν	$(\Box G)^1 : p$	$M(G^1 : pV^*)$
	$(\Diamond G)^0 : p$	$M(G^0 : pV^*)$

type	$F^{pol} : p$	$M(F^{pol} : p)$
atomic	$A^1 : p$	$\{\{A^1 : p\}\}$
α	$(\neg G)^0 : p$	$M(G^1 : p)$
	$(\neg G)^1 : p$	$M(G^0 : p)$
γ	$(\forall xG)^1 : p$	$M(G[x \backslash x^*]^1 : p)$
	$(\exists xG)^0 : p$	$M(G[x \backslash x^*]^0 : p)$
δ	$(\forall xG)^0 : p$	$M(G[x \backslash t^*]^0 : p)$
	$(\exists xG)^1 : p$	$M(G[x \backslash t^*]^1 : p)$
π	$(\Box G)^0 : p$	$M(G^0 : pa^*)$
	$(\Diamond G)^1 : p$	$M(G^1 : pa^*)$

Axiom (A) $\dfrac{}{\{\},M,Path}$ *Start (S)* $\dfrac{C_2,M,\{\}}{\varepsilon,M,\varepsilon}$ and C_2 is copy of $C_1 \in M$

Reduction (R) $\dfrac{C,M,Path \cup \{L_2:\underline{p_2}\}}{C \cup \{L_1:\underline{p_1}\},M,Path \cup \{L_2:\underline{p_2}\}}$ and $\{L_1:\underline{p_1},L_2:\underline{p_2}\}$ is σ-complementary

Extension (E) $\dfrac{C_3,M[C_1 \backslash C_2],Path \cup \{L_1:\underline{p_1}\} \quad C,M,Path}{C \cup \{L_1:\underline{p_1}\},M,Path}$

and $C_3 := \beta\text{-}clause_{L_2}(C_2)$, C_2 is copy of C_1, C_1 is e-clause of M wrt. $Path \cup \{L_1:\underline{p_1}\}$, C_2 contains $L_2:\underline{p_2}$, $\{L_1:\underline{p_1},L_2:\underline{p_2}\}$ is σ-complementary

Decomposition (D) $\dfrac{C \cup C_1,M,Path}{C \cup \{M_1\},M,Path}$ and $C_1 \in M_1$

Fig. 1. The non-clausal connection calculus for classical, intuitionistic and modal logic

The calculus works on tuples "$C,M,Path$", where M is a non-clausal matrix, C is a (subgoal) clause or ε and (the active) *path* is a set of literals or ε. The *rigid* σ_T is calculated by using *term unification* whenever a connection is identified. A *non-clausal connection proof* of M is a proof of $\varepsilon,M,\varepsilon$ in the non-clausal connection calculus.

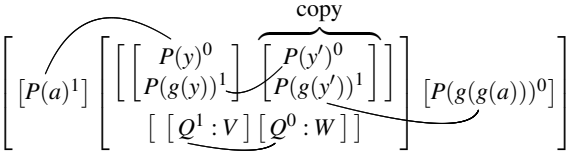
For *intuitionistic and modal logic*, the non-clausal matrix and the calculus are extended by prefixes, representing world paths in the Kripke semantics; see [22, 30, 31]. A *prefix* p is a string consisting of variables (V,W) and constants (a) and assigned to each literal. The *modal non-clausal matrix* $M(F^{pol}{:}p)$ *of a prefixed formula* $F^{pol}{:}p$ is defined according to Table 1. V^* is a new prefix variable, a^* is a prefix constant of the form $f^*(x_1,\ldots,x_n)$ in which f^* is a new function symbol and $x_1,\ldots,x_n$ are all free term and prefix variables in $(\Box G)^0 : p$ or $(\Diamond G)^1 : p$. The *modal non-clausal matrix* $M(F)$ of F is the modal non-clausal matrix $M(F^0{:}\varepsilon)$; see [22] for the *intuitionistic* case.

A *prefix substitution* σ_P assigns strings to prefix variables and is calculated by a *prefix unification* that depends on the specific non-classical logic. In intuitionistic and modal logic, a connection $\{L_1{:}p_1,L_2{:}p_2\}$ is σ*-complementary* iff both, its literals *and* prefixes can be unified under a combined substitution $\sigma = (\sigma_T,\sigma_P)$, i.e. additionally $\sigma_P(p_1) = \sigma_P(p_2)$ must hold. A *non-clausal connection proof* of M is a proof of $\varepsilon,M,\varepsilon$ in the calculus of Figure 1 (with the underlined text included) with an *admissible* σ [22].

Example 1. The formula $P(a) \wedge (\forall y(P(y) \Rightarrow P(g(y))) \vee \neg(\Box Q \Rightarrow \Diamond Q)) \Rightarrow P(g(g(a)))$ has the following (modal) non-clausal matrix (empty prefix strings are not shown):

$$\{\{P(a)^1\},\{\{\{P(y)^0,P(g(y))^1\}\},\{\{Q^1:V\},\{Q^0:W\}\}\},\{P(g(g(a)))^0\}\}\ .$$

It has the following graphical representation and (graphical) connection proof with the substitutions $\sigma_T(y) = a$, $\sigma_T(y') = g(a)$ and $\sigma_P(V) = W$: literals of each connection are connected with a line. A *clausal* proof would need eleven instead of four connections.



3 The Implementations

nanoCoP, nanoCoP-i and nanoCoP-M are theorem provers for first-order *classical* logic with equality, first-order *intuitionistic* logic with equality and several first-order *modal* logics, respectively.[1] They are very compact Prolog implementations of the basic non-clausal connection calculi extended by a few basic but effective optimizations.

3.1 Non-clausal Matrix

In the first step, the input formula F is translated into a non-clausal matrix M (see Table 1). Every (sub-)clause $(I,V,FV):C$ and submatrix $J:M$ are marked with unique indices I and J, sets V of (free) term and prefix variables that are newly introduced in C and sets FV including pairs $x:pre(x)$ of free term variables and their prefixes, necessary to check if σ_P is admissible. In Prolog, literals with polarity 1 are marked with "-". In the second step, for every literal `Lit` in M the fact `lit(Lit,ClaB,ClaC,Grnd)` is asserted into Prolog's database where `ClaC` $\in M$ is the clause in which `Lit` occurs, `ClaB` is β-$clause_{Lit}$(`ClaC`), `Grnd` is `g` iff the smallest clause in which `Lit` occurs is ground.

Example 2. The (modal) formula from Example 1 is expressed in nanoCoP syntax as

```
( p(a) , ( all Y: (p(Y) => p(g(Y)) ) ; ~ (# q => * q) ) => p(g(g(a))) )
```

and is translated into the following (modal) non-clausal matrix

```
[ (2^K)^[]^[]: [ -p(a): -[] ],
  (4^K)^[]^[]: [  5^K: [ (6^K)^[Y]^[Y:[]]: [ p(Y):[], -p(g(Y)): -[] ] ],
                 12^K: [ (13^K)^[V]^[]: [ -q: -[V] ], (16^K)^[W]^[]: [ q:[W] ] ] ],
 (18^K)^[]^[]: [ p(g(g(a))):[] ] ]
```

in which `V` and `W` are the prefix variables, the variable `K` is used to enumerate clauses.

3.2 nanoCoP for Classial Logic

The (minimalistic) source code of the nanoCoP core prover is shown in Figure 2. The underlined code is necessary only for the non-classical provers and is to be ignored for the (classical) nanoCoP prover. The predicate `prove(M,U,S,X)` implements the start rule (lines 1–5). `M` is the matrix generated in the preprocessing step, `U` is the maximum size of the active path used for iterative deepening (lines 6–7), `S` specifies a strategy (see Section 3.5), and `X` contains the returned (compact) non-clausal connection proof.

The predicate `prove(Cla,Mat,Path,T,U,Q,S,X)` implements axiom (line 8), decomposition rule (lines 9–11), reduction rule (lines 12–15, 20), and extension rule (lines 12–13, 16–20) of the calculus in Figure 1. It succeeds iff there is a proof for the tuple "`Cla`, `Mat`, `Path`" with $|$`Path`$| <$ `U`. The predicate `pe` calculates an appropriate extension clause (lines 21–24). σ is stored implicitly by Prolog. Prolog's `member` and `append` predicates are abbreviated by `m` and `a`, respectively (line 25). The predicate `posC(C,F)` (invoked in line 3 and 4) calculates a *positive (start) clause* `F` of the clause `C`. It is implemented in seven lines of (non-minimalistic) code and is the only predicate of the core prover that is not included in the code in Figure 2. The nanoCoP website includes a more readable version of the full source code.

[1] Provers available under the GNU General Public License at `http://leancop.de/nanocop/`, `http://leancop.de/nanocop-i/` , and `http://leancop.de/nanocop-m/` .

```
(1)  prove(M,U,S,[(I^0)^V:X]) :-
(2)   ( m(scut,S) -> ( a([(I^0)^V^W:F|_],[!|_],M) ; m((I^0)^V^W:C,M),
(3)   posC(C,F) ) -> true ; ( a(Z,[!|_],M) -> m((I^0)^V^W:F,Z) ;
(4)   m((I^0)^V^W:C,M),posC(C,F) ) ), prove(F,M,[],[I^0],U,[],P,B,S,X),
(5)   a(B,W,D), domain_cond(D), prefix_unify(P).

(6)  prove(M,U,S,X) :- retract(p) -> ( m(comp(U),S) -> prove(M,1,[],X);
(7)   V is U+1, prove(M,V,S,X) ) ; m(comp(_),S) -> prove(M,1,[],X).

(8)  prove([],_,_,_,_,_,[],[],_,[]).

(9)  prove([J^K:M|C],H,P,T,U,Q,A,B,S,X) :- !, m(I^V^W:F,M),
(10)  prove(F,H,P,[I,J^K|T],U,Q,D,E,S,Y), prove(C,H,P,T,U,Q,N,O,S,Z),
(11)  a(N,D,A), a(W,E,R), a(O,R,B), X=[J^K:I^V:Y|Z].

(12) prove([L:J|C],H,P,T,U,Q,P1,V1,S,X) :-
(13)  X=[L:J,I^V:[N:O|Y]|Z], \+ (m(A,[L:J|C]),m(B,P),A==B), (-N=L;-L=N)
(14)  -> ( m(R,Q), L:J==R, D=[], Y=[], I=1, V=[], O=J, P4=[], V4=[] ;
(15)  m(R:O,P), R=N, D=[], Y=[], I=r, V=[], \+ \+ prefix_unify([J=O]),
(16)  P4=[J=O], V4=[] ; lit(N:O,E,F,G), ( G=g -> true ; length(P,K),K<U
(17)  -> true ; \+ p -> assert(p), fail ), \+ \+ prefix_unify([J=O]),
(18)  pe(E,F,H,T,I^V^W:D,M), prove(D,M,[L:J|P],[I|T],U,Q,P2,V2,S,Y),
(19)  P4=[J=O|P2], a(V2,W,V4) ), ( m(cut,S) -> ! ; true ),
(20)  prove(C,H,P,T,U,[L:J|Q],P3,V3,S,Z), a(P4,P3,P1), a(V3,V4,V1).

(21) pe((I^K)^V:E,N:C,H,T,D,M) :- a(A,[(I^L)^W:F|B],H), length(T,K),
(22)  ( E=[J^K:[G]|_], m(J^L,T), V=W, C=[_:[R|_]|_], a(U,[J^L:Y|X],F),
(23)  pe(G,R,Y,T,D,Z), a(U,[J^L:Z|X],S), a(A,[(I^L)^W:S|B],M) ;
(24)  (\+m(I^L,T);V\==W) -> D=(I^K)^V:E, a(A,[N:C|B],M) ).

(25) m(A,B) :- member(A,B). a(A,B,C) :- append(A,B,C).
```

Fig. 2. Source code of the nanoCoP, nanoCoP-i and nanoCoP-M core provers

3.3 nanoCoP-i for Intuitionistic Logic

For intuitionistic logic, prefixes are added to all literals in the non-clausal matrix (details in [22]) and to the non-clausal connection calculus. For nanoCoP-i, the underlined text in Figure 2 is added to the classical nanoCoP prover; no other changes are done.

A list `P1` of prefix equations and a list `V1` of term variables (with their prefixes) are collected during the proof search and are added as arguments to the main predicate `prove(Cla,Mat,Path,T,U,Q,P1,V1,S,X)`. Two predicates need to be added to the code: `prefix_unify(P)` implements the prefix unification and `domain_cond(V)` checks whether σ is an admissible substitution.

3.4 nanoCoP-M for Multimodal Logics

For modal logic, prefixes are added to all literals in the non-clausal matrix (according to Table 1) and to the non-clausal connection calculus. The nanoCoP-M core prover shown in Figure 2 has the same source code as the intuitionistic nanoCoP-i prover.

Again, `prefix_unify(P)` implements the prefix unification with respect to a specific modal logic and `domain_cond(V)` checks whether σ is an admissible substitution with respect to a specific domain condition. nanoCoP-M supports the modal logics D, T, S4, and S5 with varying, cumulative and constant domain condition; terms are considered rigid and local, the logical consequence relation is *local* [22, 31].

For the modal logics D and T the *accessibility condition* $|\sigma_P(V)|=1$ and $|\sigma_P(V)|\leq 1$, respectively, has to hold for all prefix variables V. There is no such restriction for the modal logic S4 and only the last prefix character is considered for the modal logic S5.

nanoCoP-M also supports heterogeneous multimodal logics. For *multimodal* logic, an index can be added to the modal operators $\Box$ and $\Diamond$, i.e. modal operators from the set $\{\Box_i, \Diamond_i \,|\, i \in I\!N\}$ are allowed. Modal operators with different indices can be assigned to different modal logics. See the nanoCoP-M website for more details.

3.5 Proof Search Optimizations

Following the *lean* methodology, a few basic but effective techniques are carefully selected and integrated into the nanoCoP, nanoCoP-i and nanoCoP-M provers.

Regularity and Lemmata. *Regularity* (line 13) and *lemmata* (line 14) are effective techniques for pruning the search space in *clausal* connection calculi [10] and were already included in the basic versions of the nanoCoP provers [20, 22].

Restricted Backtracking. *Restricted backtracking* is an effective (but incomplete) technique to prune the search space in the (non-confluent) connection calculus [17]. Besides restricted backtracking for the extension and reduction rules ("`cut`") (line 19), restricted backtracking for the start rule ("`scut`") (lines 2–3) is now integrated as well, which cuts off backtracking over alternative start clauses in the connection calculus.

Conjecture Start Clauses. *Conjecture start clauses* ("`conj`") restricts the start rule for formulae of the form $(A_1 \wedge \ldots \wedge A_n) \Rightarrow C$ to clauses of the conjecture C (line 2 and 3), instead of the default positive clauses (line 3 and 4). This technique is in particular effective for formulae with many axioms A_i. This approach is incomplete for formulae with inconsistent/unsatisfiable axioms $A_1, \ldots, A_n$ and invalid conjecture C.

Reordering Clauses. *Reordering clauses* ("`reo(I)`") is a technique to modify (indirectly) the proof search order, which is in particular effective in combination with restricted backtracking. For non-clausal calculi it is important to produce diverse clause orders even for small sets of clauses, e.g., if a (sub)matrix contains only two or three clauses. It is done in a preprocessing step using a pseudo-randomized shuffle algorithm.

Strategy Scheduling. *Strategy scheduling* is a very effective technique that uses a sequence of different strategies to prove a formula. A strategy is specified in the argument `S` of the `prove` predicate. It is a list that contains a (possibly empty) subset of the options {`scut`,`cut`,`conj`, `reo(`*I*`)`,`comp(`*J*`)`}, which effect the proof search as follows:

- `scut`: switches on restricted backtracking for start clauses,
- `cut`: switches on restricted backtracking for reduction/extension/lemma rule,
- `conj`: uses conjecture clauses as start clauses instead of positive clauses,
- `reo(`I`)` for $I \in \mathbb{N}$: reorders the clauses I times before the proof search starts,
- `comp(`J`)` for $J \in \mathbb{N}$: restarts the proof search using a complete search strategy, i.e. without `scut`, `cut`, and `conj`, if the path limit `U` exceeds J (lines 6–7).

A fixed strategy scheduling (sequence) is implemented using a shell script that invokes the Prolog prover. Comprehensive tests were performed in order to select a set of 20 strategies for nanoCoP and 12 strategies for nanoCoP-i and nanoCoP-M, respectively. The first three stategies used by nanoCoP, nanoCoP-i and nanoCoP-M are `[cut,comp(7)]` / `[reo(22),conj,cut]` / `[scut]`, `[cut,comp(6)]` / `[scut]` / `[scut,cut]`, and `[cut,comp(6)]` / `[cut]` / `[reo(20),conj,cut]`, respectively. The empty (and complete) strategy `[]` is the last one used by all three nanoCoP provers.

3.6 Proof Output

All three nanoCoP provers can output a detailed non-clausal connection proof. The nanoCoP core provers return a very compact (and hardly readable) non-clausal connection proof that has been further optimized in terms of size and included proof information. It is returned in the last argument `X` of the `prove` predicate in Figure 2.

Example 3. For the (modal) formula from Example 1 and its non-clausal (modal) matrix given in Example 2, the nanoCoP-M core prover returns the following compact (modal) non-clausal connection proof

```
[ (18^0)^[]: [ p(g(g(a))):  [],
  (4^1)^[]: [ -p(g(g(a))): -[], 5^1: (6^1)^[g(a)]: [ p(g(a)):  [],
                                    (6^4)^[a]: [ -p(g(a)): -[], p(a):  [],
                                                 (2^5)^[]: [ -p(a): -[]] ] ],
             12^1: (13^1)^[[V]]: [ -q: -[[V]],
                     (16^4)^[V]: [ q:   [V] ] ] ] ]
```

in which the literals of the connections have been underlined. In the terms of the form `(I^K)^L:C`, `I` and `K` are the index and the instance (number) of the clause `C`, respectively, and `L` is a list that contains the substituted term and prefix variables.

Based on this returned compact proof, a detailed and more readable non-clausal connection proof is reconstructed in a separate module. As non-clausal connection proofs are closely related to proofs in Gentzen's LK/LJ sequent calculi [8] and Schütte's GS calculus [5], they can (rather) easily be translated into LK/LJ/GS proofs.

4 Experimental Evaluation

The optimizations described in Section 3 were integrated into the nanoCoP 2.0 provers and are evaluated on different benchmark libraries. All evaluations were conducted on a 2.3 GHz Xeon system with 32 GB of RAM running Linux 2.6.32. If not stated otherwise, ECLiPSe Prolog 5.10 was used for all provers implemented in Prolog.[2]

[2] ECLiPSe Prolog 5.x is available at `https://eclipseclp.org/Distribution/Builds/`. Newer versions of ECLiPSe Prolog are missing important features (e.g. the possibility to switch on a global occurs check) and have a significantly lower performance.

Table 2. Results on the first-order problems of the TPTP library

	leanTAP 2.3	leanCoP 2.2	E 2.4	nanoCoP 1.0	nanoCoP 2.0 SWI	nanoCoP 2.0	nanoCoP +leanCoP
proved	555	2541	4377	2055	2132	**2500**	2709
0 to 1sec.	520	1643	3152	1543	1325	**1573**	1590
1 to 10sec.	20	369	780	277	317	**264**	294
10 to 100sec.	15	529	445	235	490	**663**	825
refuted	0	67	510	133	132	**133**	133
total	555	2608	4887	2188	2264	**2633**	2842

nanoCoP. The classical nanoCoP prover was evaluated on all 8044 first-order (so-called FOF) problems in the TPTP library v6.4.0 [29]. Table 2 shows the results of the evaluation for a CPU time limit of 100 seconds. Besides nanoCoP 2.0, it includes the following provers: the lean tableau prover leanTAP 2.3 [1], the superposition prover E 2.4 [27] (using the options "`--proof-object -s --satauto`"), leanCoP 2.2 [17], and nanoCoP 1.0 [20]. It also includes the results of nanoCoP running on SWI Prolog 7.6.4 and the combined results of nanoCoP 2.0 and leanCoP 2.2.

nanoCoP 2.0 proves 22% more problems than nanoCoP 1.0 and 350% more problems than leanTAP, the other lean prover that is based on a non-clausal (tableau) calculus. E proves 75% more problems than nanoCoP 2.0. nanoCoP 2.0 performs significantly better on ECLiPSe Prolog than on SWI Prolog. The numbers in the last column indicate that nanoCoP 2.0 proves 168 problems that are not proven by leanCoP 2.2.

Optimizations. Table 3 shows the effectiveness of the different optimization techniques implemented in nanoCoP 2.0 on all 8044 FOF problems in the TPTP library v6.4.0 for a CPU time limit of 10 seconds. The following versions of nanoCoP are evaluated: a basic version without regularity and lemmata ("basic"), the standard version using regularity and lemmata (i.e. strategy `[]`), a version with conjecture start clauses (`[conj]`), two versions with restricted backtracking (`[scut]` and `[cut]`, respectively), nanoCoP 1.0 (which uses the single strategy `[cut,comp(6)]`), a "reo" version with reordering of clauses (using the strategy `[reo(22),conj,cut]`), and the full nanoCoP 2.0 prover using all of the described optimizations including strategy scheduling.

As different optimizations can be combined within the strategy scheduling, not only the total number of proved problems is given, but also the number of new problems proved compared to the "basic" or the standard nanoCoP version (using strategy `[]`).

Table 3. Evaluation of different optimization techniques

	"basic"	`[]`	`[conj]`	`[scut]`	`[cut]`	1.0	"reo"	2.0
proved	1465	1516	1682	1598	1691	1820	1855	**2079**
new proved	–	64	253	248	421	409	260	**314**
compared to	–	"basic"	`[]`	`[]`	`[]`	`[]`	1.0	1.0

Table 4. Results on the first-order problems of the ILTP library

	ileanTAP 1.17	ileanCoP 1.2	Slakje 2.14	nanoCoP-i 1.0	nanoCoP-i 2.0	nanoCoP-i +ileanCoP
proved	314	782	1019	764	**839**	848
0 to 1sec.	303	612	95	681	**676**	676
1 to 10sec.	7	51	367	44	**47**	51
10 to 100sec.	4	119	557	39	**116**	121
refuted	4	78	363	89	89	91

nanoCoP-i. Table 4 shows the results of the evaluation on all 2550 first-order problems in the ILTP library v1.1.2 [26] for a CPU time limit of 100 seconds. Included are the provers ileanTAP 1.17, ileanCoP 1.2, Slakje 2.14, nanoCoP-i 1.0, nanoCoP-i 2.0 and the combined results of nanoCoP-i 2.0 and ileanCoP 1.2. ileanTAP [14] implements a prefixed free-variable tableau calculus and is written in Prolog; ileanCoP [15, 16] is a compact Prolog prover that implements the prefixed *clausal* connection calculus; Slakje [6] uses a prover for classical logic to search for a classical proof and the GAPT system [7] to subsequently reconstruct an intuitionistic proof from the classical one. These are currently the fastest theorem provers for intuitionistic first-order logic (JProver, ft and ileanSeP prove significantly less problems than ileanCoP [16, 22]).

nanoCoP-i 2.0 proves about 10% more problem than nanoCoP-i 1.0. It also proves more problem than ileanCoP, currently the fastest connection/tableau prover for first-order intuitionistic logic. Slakje proves the largest number of problems, but the proof reconstruction with GAPT shows a significant overhead. ileanCoP as well as nanoCoP-i prove significant more problems than Slakje within a time limit of 10 seconds.

nanoCoP-M. Table 5 shows the results of the evaluation on all 580 unimodal problems of the QMLTP library v1.1 [25]. Results are shown for the modal logics D, T, S4 and S5, and for the varying, cumulative, and constant domain variants. It includes the following provers: MleanTAP 1.3, MleanCoP 1.3, nanoCoP-M 1.0, and nanoCoP-M 2.0. MleanTAP [2] implements a prefixed tableau calculus; MleanCoP [19] implements a prefixed *clausal* connection calculus; both provers are written in Prolog. Up to the authors knowledge, these are currently the only provers for modal first-order logic (the sequent prover MleanSeP proves about the same number of problems as MleanTAP [19]).

nanoCoP-M 2.0 proves on average 16%, 4%, 11% and 6% more problems than nanoCoP-M 1.0 for the modal logics D, T, S4 and S5, respectively. It refutes about the same number of problems as nanoCoP-M 1.0. nanoCoP-M 2.0 also proves more problems than MleanCoP, which was so far the most successful prover on the QMLTP library [28]. The higher-order prover Leo-III [28] uses an embedding of modal logics into simple type theory in order to deal with a wide range of different (higher-order) modal logics. Leo-III does not support the QMLTP syntax, but previous evaluations show that it proves slightly fewer problems of the QMLTP library than MleanCoP [28]. nanoCoP 2.0 solves 17 of the 20 multimodal problems in the QMLTP library, all of them within one second.

Table 5. Results on the unimodal problems (varying/cumul./constant) of the QMLTP library

Logic	MleanTAP 1.3 proved	— MleanCoP 1.3 — proved	refuted	nanoCoP-M1.0 proved	— nanoCoP-M 2.0 — proved	refuted
D	100/120/135	184/206/223	274/248/222	167/187/204	**193/213/230**	265/245/229
T	138/160/175	223/251/271	159/132/114	222/244/263	**231/253/273**	153/133/119
S4	169/205/220	286/349/363	127/96/83	271/321/336	**297/355/370**	124/98/85
S5	219/272/272	358/435/435	94/41/41	343/414/414	**365/440/440**	92/44/44

5 Conclusion

In this paper the nanoCoP 2.0 provers for classical, intuitionistic and modal logics have been presented. They are very compact and modular Prolog implementations of the non-clausal connection calculi for classical and non-classical logics. The integration of a few effective optimization techniques improves performance significantly. Compared to the previous versions, the classical nanoCoP 2.0 system solves about 20% more problems from the TPTP library, the intuitionistic nanoCoP-i 2.0 and the modal nanoCoP-M 2.0 systems prove about 10% more problems from the ILTP and QMLTP libraries. Despite the overhead caused by the more complex non-clausal data structure, nanoCoP-i and nanoCoP-M prove more problems than the corresponding clausal provers ileanCoP and MleanCoP, and they are now among the fastest provers for these non-classical logics.

All nanoCoP 2.0 provers can provide detailed non-clausal connection proofs. Preliminary results show that on the non-clausal problems in the TPTP library, the non-clausal proofs of nanoCoP have on average only half the number of connections than the clausal proofs produced by leanCoP. The non-clausal proofs are also more "natural" as the structure of the original formula is preserved throughout the whole proof search. This makes them in particular interesting for applications where a human readable output or interaction is required. For example, the normative reasoner NAI uses MleanCoP at its backend in order to reason over legal texts formalized in a multimodal first-order logic [11, 12]. nanoCoP-M 2.0, which now also supports heterogeneous multimodal logics, could be used in order to return a more natural human-readable proof.

Future work includes the integration of better *refuting techniques* into the nanoCoP provers, which were so far not in the focus of the development. It also includes the extension to other modal logics, such as the *modal logic K*, for which a connection-based proof approach is more difficult as subformulae that are not involved in any connection might be relevant for a successful proof [31]. More straightforward is the development of connection calculi and provers for first-order *intuitionistic modal* logic. The presented calculi and provers are optimized for full *first-order* logic. Combining these with calculi for propositional logic might be promising as these calculi are entirely different from the first-order ones. Another future work is the integration of *learning techniques* into nanoCoP as already done in many (re-)implementations of leanCoP [9, 13, 32].

Acknowledgements. The author would like to thank W. Bibel for his helpful feedback.

References

1. Beckert, B., Posegga, J.: leanTAP: Lean tableau-based deduction. Journal of Automated Reasoning **15**(3), 339–358 (1995)
2. Benzmüller, C., Otten, J., Raths, T.: Implementing and evaluating provers for first-order modal logics. In: De Raedt, L., et al. (eds.) 20th European Conference on Artificial Intelligence (ECAI 2012). pp. 163–168. IOS Press, Amsterdam (2012)
3. Bibel, W.: Matings in matrices. Commun. ACM **26**(11), 844–852 (1983)
4. Bibel, W.: Automated Theorem Proving. Artificial intelligence, F. Vieweg und Sohn, Wiesbaden, 2nd edn. (1987)
5. Bibel, W., Otten, J.: From Schütte's formal systems to modern automated deduction. In: Kahle, R., Rathjen, M. (eds.) The Legacy of Kurt Schütte. pp. 217–251. Springer, Cham (2020)
6. Ebner, G.: Herbrand constructivization for automated intuitionistic theorem proving. In: Cerrito, S., Popescu, A. (eds.) TABLEAUX 2019. LNAI, vol. 11714, pp. 355–373. Springer, Cham (2019)
7. Ebner, G., Hetzl, S., Reis, G., Riener, M., Wolfsteiner, S., Zivota, S.: System description: GAPT 2.0. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNAI, vol. 9706, pp. 293–301. Springer, Cham (2016)
8. Gentzen, G.: Untersuchungen über das Logische Schließen. Mathematische Zeitschrift **39**, 176–210, 405–431 (1935)
9. Kaliszyk, C., Urban, J.: FeMaLeCoP: Fairly efficient machine learning connection prover. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR-20. LNAI, vol. 9450, pp. 88–96. Springer, Heidelberg (2015)
10. Letz, R., Stenz, G.: Model elimination and connection tableau procedures. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 2015–2112. Elsevier Science, Amsterdam (2001)
11. Libal, T., Pascucci, M.: Automated reasoning in normative detachment structures with ideal conditions. In: Seventeenth International Conference on Artificial Intelligence and Law. pp. 63–72. ICAIL '19, Association for Computing Machinery, New York (2019)
12. Libal, T., Steen, A.: The NAI suite – drafting and reasoning over legal texts. In: Araszkiewicz, M., Rodríguez-Doncel, V. (eds.) 32nd International Conference on Legal Knowledge and Information Systems (JURIX 2019). Frontiers in Artificial Intelligence and Applications, vol. 322, pp. 243–246. IOS Press, Amsterdam (2019)
13. Olšák, M., Kaliszyk, C., Urban, J.: Property invariant embedding for automated reasoning. In: Giacomo, G.D., et al. (eds.) ECAI 2020. Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 1395–1402. IOS Press, Amsterdam (2020)
14. Otten, J.: ileanTAP: An intuitionistic theorem prover. In: Galmiche, D. (ed.) TABLEAUX 1997. LNAI, vol. 1227, pp. 307–312. Springer, Heidelberg (1997)
15. Otten, J.: Clausal connection-based theorem proving in intuitionistic first-order logic. In: Beckert, B. (ed.) TABLEAUX 2005. LNAI, vol. 3702, pp. 245–261. Springer, Heidelberg (2005)
16. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNAI, vol. 5195, pp. 283–291. Springer, Heidelberg (2008)
17. Otten, J.: Restricting backtracking in connection calculi. AI Commun. **23**(2–3), 159–182 (2010)
18. Otten, J.: A non-clausal connection calculus. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNAI, vol. 6793, pp. 226–241. Springer, Heidelberg (2011)

19. Otten, J.: MleanCoP: A connection prover for first-order modal logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNAI, vol. 8562, pp. 269–276. Springer, Heidelberg (2014)
20. Otten, J.: nanoCoP: A non-clausal connection prover. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNAI, vol. 9706, pp. 302–312. Springer, Heidelberg (2016)
21. Otten, J.: nanoCoP: Natural non-clausal theorem proving. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, Sister Conference Best Paper Track. pp. 4924–4928. IJCAI (2017)
22. Otten, J.: Non-clausal connection calculi for non-classical logics. In: Schmidt, R., Nalon, C. (eds.) TABLEAUX 2017. LNAI, vol. 10501, pp. 209–227. Springer, Cham (2017)
23. Otten, J., Bibel, W.: leanCoP: lean connection-based theorem proving. Journal of Symbolic Computation **36**(1–2), 139–161 (2003)
24. Otten, J., Bibel, W.: Advances in connection-based automated theorem proving. In: Hinchey, M., Bowen, J.P., Olderog, E.R. (eds.) Provably Correct Systems. pp. 211–241. NASA Monographs in Systems and Software Engineering, Springer, Cham (2017)
25. Raths, T., Otten, J.: The QMLTP problem library for first-order modal logics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNAI, vol. 7364, pp. 454–461. Springer, Heidelberg (2012)
26. Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic. Journal of Automated Reasoning **38**, 261–271 (2007)
27. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) CADE 27. LNCS, vol. 11716, pp. 495–507. Springer, Cham (2019)
28. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Automated Reasoning. LNAI, vol. 10900, pp. 108–116. Springer, Cham (2018)
29. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning **59**(4), 483–502 (2017)
30. Waaler, A.: Connections in nonclassical logics. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1487–1578. Elsevier Science, Amsterdam (2001)
31. Wallen, L.A.: Automated Deduction in Nonclassical Logics. MIT Press, Cambridge (1990)
32. Zombori, Z., Urban, J., Brown, C.E.: Prolog technology reinforcement learning prover. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNAI, vol. 12167, pp. 489–507. Springer, Cham (2020)