Non-clausal Connection-based Theorem Proving in Intuitionistic First-Order Logic

Jens Otten

Department of Informatics, University of Oslo
PO Box 1080 Blindern, 0316 Oslo, Norway
jeotten@leancop.de

Abstract

This paper introduces a non-clausal connection calculus for intuitionistic first-order logic. It is an extension of the non-clausal connection calculus for classical logic by prefixes and an additional prefix unification, which encode the Kripke semantics of intuitionistic logic. nanoCoP-i is a first implementation of this intuitionistic non-clausal connection calculus. Details of the compact Prolog code are presented, which extends the non-clausal connection prover nanoCoP for classical logic. Experimental evaluations on the ILTP and the TPTP problem libraries show a solid performance of the nanoCoP-i prover. In comparison to the ileanCoP prover, the resulting non-clausal proofs are not only shorter, but can also be more easily translated into, e.g., sequent proofs.

1 Introduction

Intuitionistic (or *constructive*) *logic* is one of the most popular non-classical logics. It is often used when constructing provably correct software, for example within the NuPRL [6] proof development system. Other interactive proof assistants, such as Coq [3], use a constructive logic as well. Hence, *automated reasoning* in intuitionistic logic is an important task and many applications would benefit from more powerful reasoning tools.

During the last decade the development of automated theorem proving (ATP) systems for *classical* first-order logic has made significant progress. Extending these ATP systems to intuitionistic logic is usually not (easily) possible, as many of the underlying calculi and techniques can not be adapted to intuitionistic logic. Most leading ATP systems for classical logic require the translation of the input formulae into a *clausal form*, i.e. into disjunctive or conjunctive normal form. For intuitionistic logic there is no validity-preserving translation into such a (simple) clausal form.

Nevertheless, ileanCoP, one of the fastest ATP systems for intuitionistic first-order logic, uses a clausal connection calculus and additional prefixes to capture the semantics of intuitionistic logic [16]. While the use of a clausal form technically simplifies the proof search, the standard translation as well as the definitional translation [22] into clausal form introduce a significant overhead into the proof search [17]. Furthermore, a translation into clausal form modifies the structure of the original formula and the translation of the clausal form proof back into one of a more readable proof of the original formula is not straightforward [24, 25]. On the other hand, fully automated theorem provers that use non-clausal calculi, such as standard tableau or sequent calculi, have a significant lower performance [16].

The paper is structured as follows. Section 2 introduces the non-clausal connection calculus for intuitionistic first-order logic, which works on prefixed non-clausal matrices. The non-clausal connection calculus and the prefix unification that is additionally required are presented. An implementation of the intuitionistic non-clausal connection calculus is described in Section 3. It provides details about the non-clausal prefixed matrices, the source code of the non-clausal proof search and the prefix unification algorithm. Section 4 presents an evaluation of the implementation on the ILTP and the TPTP problem libraries. The paper concludes with a short summary and an outlook on future work in Section 5.

2 The Intuitionistic Connection Calculus

The prefixed non-clausal matrix is the main concept used in the intuitionistic connection calculus. The intuitionistic connection calculus uses prefixes and requires an additional prefix unification algorithm.

2.1 Preliminaries

The standard notation for first-order formulae is used. Terms (denoted by t) are built up from functions (denoted by f), constants and (term) variables (denoted by x). An atomic formula (denoted by A) is built up from predicate symbols and terms. 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 *literal* L has the form A or $\neg A$. Its *complement* $\overline{L}$ is A if L is of the form $\neg A$; otherwise $\overline{L}$ is $\neg L$.

A *connection* is a set $\{A, \neg A\}$ of literals with the same predicate symbol but different polarity. A *term substitution* σ_Q assigns terms to variables.

Intuitionistic logic [7] and classical logic share the same *syntax*, i.e. formulae in both logics use the same connectives and quantifiers, but their *semantics* is different. For example, the formula

$$\textit{man(Socrates)} \vee \neg\textit{man(Socrates)} \tag{1}$$

is valid in classical logic, but not in intuitionistic logic. This property holds for all formulae of the form $A \vee \neg A$ for any atomic formula A. In classical logic this formula is valid as either A is true or $\neg A$ is true whether A is true or not true. The semantics of intuitionistic logic requires a proof for A or for $\neg A$. As this property neither holds for A nor for $\neg A$, the formula is not valid in intuitionistic logic. Formally, the semantics of intuitionistic logic is specified by a Kripke semantics [30].

2.2 Prefixed Non-clausal Matrices

The intuitionistic non-clausal connection calculus is based on Wallen's *matrix characterization* [29, 30] and uses prefixes to encode the Kripke semantics for intuitionistic logic.

Definition 2.1 (Prefix)**.** *A* prefix *(denoted by p) is a string, i.e., a sequence of characters over an alphabet $\Phi \cup \Psi$, in which Φ is a set of* prefix variables *($V_1, ...$) and Ψ is a set of* prefix constants *($a_1, ...$).*

Semantically, a prefix encodes a sequence of worlds in a Kripke model. Proof-theoretically, prefix constants and variables represent applications of the rules $\neg$*-right*, $\Rightarrow$*-right*, $\forall$*-right*, and $\neg$*-left*, $\Rightarrow$*-left*, $\forall$*-left* in the sequent calculus [8], respectively. The prefix p of a subformula G, denoted $G : p$, specifies the sequence of rules that have to be applied (bottom-up) to obtain G in the sequent. In order to preserve the atomic formulae that form an axiom in the intuitionistic sequent calculus, their prefixes need to unify under an intuitionistic substitution σ_J. Hence, in the matrix characterization for intuitionistic logic, it is required that the prefixes of the literals in every connection unify under σ_J [30].

Definition 2.2 (Intuitionistic substitution, σ-complementary)**.** *An* intuitionistic substitution *$\sigma_J : \Phi \to (\Phi \cup \Psi)^*$ maps elements of Φ to strings over $\Phi \cup \Psi$. For a combined substitution $\sigma := (\sigma_Q, \sigma_J)$, a* connection *$\{L_1 : p_1, L_2 : p_2\}$ is* σ-complementary *iff $\sigma_Q(L_1) = \sigma_Q(\overline{L_2})$ and $\sigma_J(p_1) = \sigma_J(p_2)$.*

An additional *domain condition* on σ ensures that σ_Q and σ_J are mutually consistent [30]. The *irreflexivity test* of the *reduction ordering* [30] is realized by the *occurs check* during the term and prefix unification. To this end, the *skolemization* technique, originally used to eliminate *eigenvariables* in classical logic, is extended and also used for prefix constants in intuitionistic logic [15]. For the extended skolemization, the same skolem function symbol is used for instances of the same subformula, a technique that is similar to the *liberalized δ^+-rule* in classical tableau calculi [10].

Table 1: The definition of the prefixed non-clausal matrix for intuitionistic logic

type	$F^{pol}:p$	$M(F^{pol}:p)$
atomic	$A^0:p$	$\{\{A^0:pa^*\}\}$
	$A^1:p$	$\{\{A^1:pV^*\}\}$
α	$(\neg G)^0:p$	$M(G^1:pa^*)$
	$(\neg G)^1:p$	$M(G^0:pV^*)$
	$(G\wedge H)^1:p$	$\{\{M(G^1:p)\}\},\{\{M(H^1:p)\}\}$
	$(G\vee H)^0:p$	$\{\{M(G^0:p)\}\},\{\{M(H^0:p)\}\}$
	$(G\Rightarrow H)^0:p$	$\{\{M(G^1{:}pa^*)\}\},\{\{M(H^0:pa^*)\}\}$
β	$(G\wedge H)^0:p$	$\{\{M(G^0:p),M(H^0:p)\}\}$
	$(G\vee H)^1:p$	$\{\{M(G^1:p),M(H^1:p)\}\}$
	$(G\Rightarrow H)^1:p$	$\{\{M(G^0:pV^*),M(H^1:pV^*)\}\}$
γ	$(\forall xG)^1:p$	$M(G[x\backslash x^*]^1:pV^*)$
	$(\exists xG)^0:p$	$M(G[x\backslash x^*]^0:p)$
δ	$(\forall xG)^0:p$	$M(G[x\backslash t^*]^0:pa^*)$
	$(\exists xG)^1:p$	$M(G[x\backslash t^*]^1:p)$

An intuitionistic non-clausal matrix is a set of prefixed clauses, which consist of prefixed literals and prefixed (sub)matrices. The *polarity* 0 or 1 is used to represent negation in a matrix, i.e. literals of the form A and $\neg A$ are represented by A^0 and A^1, respectively,

Definition 2.3 (Intuitionistic non-clausal matrix)**.** *Let F be a formula, pol be a polarity, and p be a prefix. The* prefixed (non-clausal) matrix $M(F^{pol}{:}p)$ *of a* prefixed formula $F^{pol}{:}p$ *is a set of prefixed clauses, in which a prefixed clause is a set of prefixed literals and prefixed (non-clausal) matrices, and defined inductively according to Table 1. $G[x\backslash t]$ denotes the formula G in which all free occurrences of x are replaced by t. 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 term and prefix variables in $(\forall xG)^0:p$ or $(\exists xG)^1:p$. 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 $A^0:p$, $(\neg G)^0:p$, $(G{\Rightarrow}H)^0:p$, or $(\forall xG)^0:p$. The* intuitionistic (non-clausal) matrix $M(F)$ *of F is the prefixed matrix $M(F^0:\varepsilon)$.*

In the *graphical representation* of a matrix, its clauses are arranged horizontally, while the literals and (sub-)matrices of each clause are arranged vertically.

For example, the formula

$$\begin{aligned} &(\,\mathit{man(Socrates)} \Rightarrow \mathit{man(Socrates)}\,)\\ \wedge\, (\,(\,&\mathit{man(Plato)} \wedge \forall x(\mathit{man(x)}{\Rightarrow}\mathit{mortal(x)})\,)\Rightarrow \mathit{mortal(Plato)}\,) \end{aligned} \tag{2}$$

has the (simplified, i.e. redundant brackets are removed) intuitionistic non-clausal matrix

$$\begin{aligned} &\{\{\ \{\{\mathit{man(Socrates)}^0:a_1V_1\},\{\mathit{man(Socrates)}^1:a_1a_2\}\},\ \{\{\mathit{man(Plato)}^1:a_3V_2\},\\ &\{\mathit{man(x)}^0:a_3V_3V_4a_4(V_3,x,V_4),\mathit{mortal(x)}^1:a_3V_3V_4V_5\},\{\mathit{mortal(Plato)}^0:a_3a_5\}\}\ \}\}\ , \end{aligned} \tag{3}$$

and the following graphical representation

$$\left[\left[\begin{array}{c} \left[\ \left[\ \mathit{man(Socrates)}^0{:}a_1V_1\ \right]\ \left[\ \mathit{man(Socrates)}^1{:}a_1a_2\ \right]\ \right] \\ \left[\ \left[\ \mathit{man(Plato)}^1{:}a_3V_2\ \right] \left[\begin{array}{c} \mathit{man(x)}^0{:}a_3V_3V_4a_4(V_3,x,V_4) \\ \mathit{mortal(x)}^1{:}a_3V_3V_4V_5 \end{array}\right] \left[\ \mathit{mortal(Plato)}^0{:}a_3a_5\ \right]\ \right] \end{array}\right]\right].$$

2.3 An Intuitionistic Non-clausal Connection Calculus

The non-clausal connection calculus for intuitionistic logic extends the non-clausal connection calculus for classical logic [18] and generalizes the clausal connection calculus for intuitionistic logic [15]. According to the matrix characterization of logical validity [4, 5, 30] a formula F is valid, if and only if all paths through its matrix $M(F)$ (with added clause copies) contain a σ-complementary connection. The (non-clausal) connection calculus uses a *connection-driven* search strategy. In each reduction and extension step of the calculus a σ-complementary connection is identified and only paths that do not contain this connection are investigated afterwards. If every path contains a σ-complementary connection, the proof search succeeds and the given formula is valid. In contrast to sequent calculi, connection calculi permit a more *goal-oriented* proof search. This leads to a significantly smaller search space and, thus, to a more efficient proof search. A *non-clausal connection proof* can be illustrated within the graphical matrix representation.

For example, the proof of formula (2) and its matrix (3) consists of three connections, which are represented by a line in the following graphical matrix representation. The three connections are σ-complementary with $\sigma=(\sigma_Q,\sigma_J)$ and $\sigma_Q(x)=\textit{Plato}$, $\sigma_J(V_1)=a_2$, $\sigma_J(V_2)=a_4(\varepsilon,\textit{Plato},\varepsilon)$, $\sigma_J(V_3)=\varepsilon$, $\sigma_J(V_4)=\varepsilon$, $\sigma_J(V_5)=a_5$ (where ε is the empty string).

$$\left[\left[\begin{array}{c} \left[\;\left[\;\textit{man(Socrates)}^0{:}a_1V_1\;\right]\;\left[\;\textit{man(Socrates)}^1{:}a_1a_2\;\right]\;\right] \\ \left[\;\left[\;\textit{man(Plato)}^1{:}a_3V_2\;\right]\left[\begin{array}{c}\textit{man(x)}^0{:}a_3V_3V_4a_4(V_3,x,V_4)\\ \textit{mortal(x)}^1{:}a_3V_3V_4V_5\end{array}\right]\left[\;\textit{mortal(Plato)}^0{:}a_3a_5\;\right]\;\right]\end{array}\right]\right] .$$

The intuitionistic matrix of formula (1) is

$$\{\{\textit{man(Socrates)}^0 : a_1\},\{\textit{man(Socrates)}^1 : a_2\}\} \; . \tag{4}$$

There is only one connection $\{\textit{man(Socrates)}^0 : a_1, \textit{man(Socrates)}^1 : a_2\}$ which is σ-complementary if the two prefixes a_1 and $_2$ can be unified. There is no substitution σ_J with $\sigma_J(a_1)=\sigma_J(a_2)$ and, hence, no connection proof of this matrix. Therefore, the formula *man(Socrates)* $\vee\neg$*man(Socrates)* is *not* valid in intuitionistic logic.

A few additional concepts are required as follows in order to specify which clauses can be used within the generalized non-clausal extension rule.

Definition 2.4 (α-related, parent clause, clause copy)**.** *A clause C* contains *a literal L (or clause C'') iff $L\in C$ or C' contains L ($C=C''$ or C' contains C'') for a matrix $M'\in C$ with $C'\in M$. A clause C is α-related to a literal L iff it occurs besides L in the graphical matrix representation; more precisely, C is* α-related *to a literal L iff $\{C',C''\}\subseteq M'$ for some matrix M' such that C' contains L and C'' contains C. C' is a* parent clause *of C iff $M'\in C'$ and $C\in M'$ for some matrix M'. In the* copy of a clause *C all* free variables *in C are replaced by fresh variables. $M[C_1\backslash C_2]$ denotes the matrix M, in which the clause C_1 is replaced by the clause C_2.*

For example, in the matrix (3) the clauses $\{\textit{man(Plato)}^1 : a_3V_2\}$ and $\{\textit{mortal(Plato)}^0 : a_3a_5\}$ are α-related. $\{\ \{\{\textit{man(Socrates)}^0 : a_1V_1\},\{\textit{man(Socrates)}^1 : a_1a_2\}\},\ \{\{\textit{man(Plato)}^1 : a_3V_2\},\{\textit{man(x)}^0 : a_3V_3V_4a_4(V_3,x,V_4),\textit{mortal(x)}^1 : a_3V_3V_4V_5\},\{\textit{mortal(Plato)}^0 : a_3a_5\}\}\ \}$ is the parent clause of all other clauses in the matrix, e.g., of the clause $\{\textit{man(Socrates)}^0 : a_1V_1\}$.

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)	$\dfrac{C,M,Path{\cup}\{L_2:p_2\}}{C{\cup}\{L_1:p_1\},M,Path{\cup}\{L_2:p_2\}}$	and $\{L_1:p_1,L_2:p_2\}$ is σ-complementary
Extension (E)	$\dfrac{C_3,M[C_1\backslash C_2],Path{\cup}\{L_1{:}\,p_1\}\quad C,M,Path}{C{\cup}\{L_1{:}\,p_1\},M,Path}$	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. $Path\cup\{L_1{:}p_1\}$, C_2 contains $L_2{:}p_2$, $\{L_1{:}p_1,L_2{:}p_2\}$ is σ-complementary
Decomposition (D)	$\dfrac{C\cup C_1,M,Path}{C{\cup}\{M_1\},M,Path}$	and $C_1{\in}M_1$

Figure 1: The intuitionistic non-clausal connection calculus

Definition 2.5 (Extension clause, β-clause)**.** *C is an* extension clause (e-clause) of the matrix 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 all literals of $Path$ occurring in M and if C has a parent clause, it contains a literal of $Path$. In the* β-clause of C_2 with respect to L_2*, denoted by* β-clause$_{L_2}(C_2)$*, L_2 and all clauses that are α-related to L_2 are deleted from C_2 (in the new subgoal C_3).*

The non-clausal connection calculus for intuitionistic logic has the same axiom, start rule, and reduction rule as the *clausal* connection calculus [15]. The extension rule is slightly modified and a decomposition rule is added. It is an extension of the non-clausal connection calculus for classical logic [16], in which a prefix is added to each literal and an additional intuitionistic substitution is used to identify σ-complementary connections.

Definition 2.6 (Intuitionistic non-clausal connection calculus)**.** *The axiom and the rules of the* intuitionistic (non-clausal) connection calculus *are given in Fig. 1. It works on tuples "$C,M,Path$", where M is a prefixed non-clausal matrix, C is a prefixed (subgoal) clause or ε and (the active) $Path$ is a set of prefixed literals or ε. $\sigma=(\sigma_Q,\sigma_J)$ is a combined term and prefix substitution. An* intuitionistic (non-clausal) connection proof *of a prefixed matrix M is an intuitionistic connection proof of $\varepsilon,M,\varepsilon$.*

The non-clausal connection calculus for intuitionistic logic is *correct* and *complete*, i.e. a formula F is valid in intuitionistic logic if and only if there is an intuitionistic non-clausal connection proof of its intuitionistic non-clausal matrix $M(F)$. It follows from the correctness and completeness of the non-clausal connection calculus [16] and the clausal connection calculus for intuitionistic logic [15].

The analytic, i.e., bottom-up *proof search* in the non-clausal calculus is carried out in the same way as in the clausal calculus. Additional *backtracking* might be required when choosing the clause C_1 in the decomposition rule; no backtracking is required when choosing the matrix M_1. The *rigid* term and intuitionistic substitutions σ_Q and σ_J are calculated whenever a reduction or extension rule is applied. The term substitution is calculated by one of the well-known algorithms for term unification. The intuitionistic substitution is calculated by a prefix unification algorithm (see Section 2.4). *Optimization techniques*, such as positive start clauses, regularity, lemmata and restricted backtracking, can be employed in a similar way as in the clausal connection calculus for intuitionistic logic [16].

2.4 Prefix Unification

The intuitionistic substitution σ_J is calculated by a *prefix unification algorithm* [15]. For a given set of prefix equations $\{p_1 = q_1, \ldots, p_n = q_n\}$, an appropriate substitution σ_J is a unifier such that $\sigma_J(p_i) = \sigma_J(q_i)$ for all $1 \leq i \leq n$. General algorithms for string unification exist, but the following unification algorithm is more efficient, as it takes the *prefix property* of the prefixes $p_1, p_2, \ldots$ of a formula into account: for two prefixes $p_i = u_1 X w_1$ and $p_j = u_2 X w_2$ with $X \in \Phi \cup \Psi$ the property $u_1 = u_2$ holds. This reflects the fact that prefixes correspond to sequences of connectives and quantifiers within the same formula.

Definition 2.7. *The prefix unification for the prefixes equation $\{p = q\}$ is carried out by applying the* rewriting rules *R1 to R10 in Figure 2. It is $V, \bar{V}, V' \in \Phi$ with $V \neq \bar{V}$, V' is a new prefix variable, $a, b \in \Psi$, $X \in \Phi \cup \Psi$, and $u, w, z \in (\Phi \cup \Psi)^*$. For rule 10 the restriction $(*)$ $u = \varepsilon$ or $w \neq \varepsilon$ or $X \in \Psi$ applies. $\sigma_J(V) = u$ is written $\{V \backslash u\}$. The unification starts with the tuple $(\{p = \varepsilon | q\}, \{\})$. The application of a rewriting rule $E \to E', \tau$ replaces the tuple (E, σ_J) by the tuple $(E', \tau(\sigma_J))$. E and E' are prefix equations, σ_J and τ are substitutions. The unification terminates when the tuple $(\{\}, \sigma_J)$ is derived. In this case, σ_J represents a* most general unifier. *Rules can be applied non-deterministically and lead to a* minimal *set of most general unifiers.*

R1.	$\{\varepsilon{=}\varepsilon\|\varepsilon\}$	$\to \{\}, \{\}$	R6.	$\{Vu{=}\varepsilon\|aw\}$	$\to \{u{=}\varepsilon\|aw\}, \{V\backslash\varepsilon\}$
R2.	$\{\varepsilon{=}\varepsilon\|Xu\}$	$\to \{Xu{=}\varepsilon\|\varepsilon\}, \{\}$	R7.	$\{Vu{=}z\|abw\}$	$\to \{u{=}\varepsilon\|bw\}, \{V\backslash za\}$
R3.	$\{Xu{=}\varepsilon\|Xw\}$	$\to \{u{=}\varepsilon\|w\}, \{\}$	R8.	$\{Vau{=}\varepsilon\|\bar{V}w\}$	$\to \{\bar{V}w{=}V\|au\}, \{\}$
R4.	$\{au{=}\varepsilon\|Vw\}$	$\to \{Vw{=}\varepsilon\|au\}, \{\}$	R9.	$\{Vau{=}Xz\|\bar{V}w\}$	$\to \{\bar{V}w{=}V'\|au\}, \{V\backslash XzV'\}$
R5.	$\{Vu{=}z\|\varepsilon\}$	$\to \{u{=}\varepsilon\|\varepsilon\}, \{V\backslash z\}$	R10.	$\{Vu{=}z\|Xw\}$	$\to \{Vu{=}zX\|w\}, \{\}$ $(*)$

Figure 2: The prefix unification algorithm for intuitionistic logic

In the worst-case, the number of unifiers grows exponentially with the length of the prefixes p and q. To solve a *set* of prefix equations $\bar{E} = \{p_1 = p_1, \ldots, q_n = t_q\}$, the equations in $\bar{E}$ are solved one after the other and each calculated unifier is applied to the remaining prefix equations in $\bar{E}$.

For example, for the prefix equation $\{a_1 V_2 V_3 = a_1 a_3\}$, there are the two possible derivations:

1. $\{a_1 V_2 V_3 = \varepsilon | a_1 a_3\}, \{\} \xrightarrow{\text{R3.}} \{V_2 V_3 = \varepsilon | a_3\}, \{\} \xrightarrow{\text{R6.}} \{V_3 = \varepsilon | a_3\}, \{V_2 \backslash \varepsilon\} \xrightarrow{\text{R10.}} \{V_3 = a_3 | \varepsilon\}, \{V_2 \backslash \varepsilon\} \xrightarrow{\text{R5.}} \{\varepsilon = \varepsilon | \varepsilon\}, \{V_2 \backslash \varepsilon, V_3 \backslash a_3\}$ and
2. $\{a_1 V_2 V_3 = \varepsilon | a_1 a_3\}, \{\} \xrightarrow{\text{R3.}} \{V_2 V_3 = \varepsilon | a_3\}, \{\} \xrightarrow{\text{R10.}} \{V_2 V_3 = a_3 | \varepsilon\}, \{\} \xrightarrow{\text{R5.}} \{V_3 = \varepsilon | \varepsilon\}, \{V_2 \backslash a_3\} \xrightarrow{\text{R5.}} \{\varepsilon = \varepsilon | \varepsilon\}, \{V_2 \backslash a_3, V_3 \backslash \varepsilon\}$.

They yield the most general unifiers $\{V_2 \backslash \varepsilon, V_3 \backslash a_3\}$ and $\{V_2 \backslash a_3, V_3 \backslash \varepsilon\}$.

3 An Intuitionistic Connection Prover

The following implementation of the intuitionistic non-clausal connection calculus of Fig. 1 follows the *lean methodology* [2], which is already used for the clausal connection provers leanCoP [21] and ileanCoP [16]. It uses very compact Prolog code to implement the basic calculus and adds a few essential optimization techniques in order to prune the search space. The resulting *na*tural *no*nclausal *co*nnection *p*rover for *i*ntuitionistic logic nanoCoP-i is available under the GNU General Public License and can be downloaded at `http://www.leancop.de/nanocopi/`.

3.1 Prefixed Non-clausal Matrices

In a first step the input formula F is translated into its intuitionistic non-clausal matrix $M(F)$ according to Table 1; redundant brackets of the form "$\{\{\ldots\}\}$" are removed [18]. Additionally, every (sub-)clause $(I,V,FV)\!:\!C$ and (sub-)matrix $J\!:\!M$ are marked with unique *indices* I and J; clauses C are also marked with a set V of (free) term and prefix variables and a set FV of prefixed (free) term variables of the form $x\!:\!p$ that are newly introduced in C but not in any subclause of C. Atomic formulae are represented by Prolog atoms, term and prefix variables by Prolog variables and the polarity 1 by "–". Sets, e.g. clauses and matrices, are represented by Prolog lists (representing multisets); prefixes are also represented by Prolog lists and marked with the polarity of the corresponding literal.

For example, the matrix 3 from Sec. 2.2 is represented by the Prolog term

```
[(2^K)^[]^[]:
     [4^K:[(5^K)^[]^[]:[-(man(socrates)): -([3^[]])],
           (7^K)^[]^[]:[man(socrates):[3^[]]] ],
      10^K:[(11^K)^[]^[]:[-(man(plato)): -([8^[]])],
            (13^K)^[V,X,W]^[X:[8^[],W]]:[man(X):[8^[],W,V],
                                         -(mortal(X)):-([8^[],W,V])],
            (19^K)^[]^[]:[mortal(plato):[8^[]]] ] ] ]
```

in which the Prolog variable `K` is instantiated later on in order to enumerate clause copies; as an optimization, prefix characters introduced by atomic formulae are only considered during the unification. In the second step the matrix $M(F)$ is written into Prolog's database. For every literal `Lit` in M the fact

```
lit(Lit,ClaB,ClaC,Grnd)
```

is asserted into the database where `ClaC` $\in M$ is the (largest) clause in which `Lit` occurs and `ClaB` is the β-clause of `ClaC` with respect to `Lit`. `Grnd` is set to `g` if the smallest clause in which `Lit` occurs is ground, i.e. does not contain any variables; otherwise `Grnd` is set to `n`. Storing literals of M in the database in this way is called *lean Prolog technology* [17] and integrates the advantages of the Prolog technology approach [27] into the lean theorem proving framework. No other modifications or simplifications of the original formula (structure) are done during these two preprocessing steps.

3.2 Intuitionistic Non-clausal Proof Search

The nanoCoP-i source code is shown in Fig. 3. It is an extension of the non-clausal connection prover nanoCoP [20] for classical first-order logic. The underlined text was added to the source code of nanoCoP: (1) prefixes are added to all literals, (2) the sets `PreS` and `VarS` are added, which contain prefix equations and free (prefixed) term variables, respectively, and (3) a prefix unification is added.

First, nanoCoP-i performs a classical proof search, in which the prefixes of each connection are stored in `PreS`. If the search succeeds, the domain condition is checked and the prefixes in `PreS` are unified (line 4), using the predicates `domain_cond` and `prefix_unify` (see Section 3.3), respectively.

The predicate `prove(Mat,PathLim,Set,Proof)` implements the start rule (lines 1–4) and iterative deepening on the length of the active path (lines 5–9). `Mat` is the prefixed matrix generated in the preprocessing step, `PathLim` is the maximum size of the active path used for iterative deepening, `Set` is a list of options used to control the proof search, and `Proof` contains the returned intuitionistic (non-clausal) connection proof. Start clauses are restricted to positive clauses (line 2): after `member` selects a start clause, `positiveC(Cla,Cla1)` returns the clause `Cla1` in which all clauses that are not positive in `Cla` are deleted. A clause is positive if all of its elements (matrices and literals) are positive; a matrix is positive if it contains at least one positive clause; a literal is positive if its polarity is 0.

The predicate `prove(Cla,Mat,Path,PathI,PathLim,Lem,PreS,VarS,Set,Proof)` implements the axiom (line 10), the decomposition rule (lines 11–16), the reduction rule (lines 17–20, 24–26, 37–38), and the extension rule (lines 17–20, 28–49) of the non-clausal connection calculus in Fig. 1.

```
% start rule
prove(Mat,PathLim,Set,[(I^0)^V:Cla1|Proof]) :-
    member((I^0)^V^VS:Cla,Mat), positiveC(Cla,Cla1), Cla1\=!,
    prove(Cla1,Mat,[],[I^0],PathLim,[],PreS,VarS,Set,Proof),
    append(VarS,VS,VarS1), domain_cond(VarS1), prefix_unify(PreS).
prove(Mat,PathLim,Set,Proof) :-
    retract(pathlim) ->
    ( member(comp(PathLim),Set) -> prove(Mat,1,[],Proof) ;
      PathLim1 is PathLim+1, prove(Mat,PathLim1,Set,Proof) ) ;
    member(comp(_),Set) -> prove(Mat,1,[],Proof).

% axiom
prove([],_,_,_,_,_,[],[],_,[]).

% decomposition rule
prove([J^K:Mat1|Cla],MI,Path,PI,PathLim,Lem,PreS,VarS,Set,Proof) :- !,
    member(I^_^FV:Cla1,Mat1),
    prove(Cla1,MI,Path,[I,J^K|PI],PathLim,Lem,PreS1,VarS1,Set,Proof1),
    prove(Cla,MI,Path,PI,PathLim,Lem,PreS2,VarS2,Set,Proof2),
    append(PreS2,PreS1,PreS), append(FV,VarS1,VarS3),
    append(VarS2,VarS3,VarS), append(Proof1,Proof2,Proof).

% reduction and extension rules
prove([Lit:Pre|Cla],MI,Path,PI,PathLim,Lem,PreS,VarS,Set,Proof) :-
    Proof=[[I^V:[NegLit|ClaB1]|Proof1]|Proof2],
    \+ (member(LitC,[Lit:Pre|Cla]), member(LitP,Path), LitC==LitP),
    (-NegLit=Lit;-Lit=NegLit) ->
       ( member(LitL,Lem), Lit:Pre==LitL, PreS3=[], VarS3=[],
         ClaB1=[], Proof1=[]
         ;
         member(NegL:PreN,Path), unify_with_occurs_check(NegL,NegLit),
         \+ \+ prefix_unify([Pre=PreN]), PreS3=[Pre=PreN], VarS3=[],
         ClaB1=[], Proof1=[]
         ;
         lit(NegLit:PreN,ClaB,Cla1,Grnd1),
         ( Grnd1=g -> true ; length(Path,K), K<PathLim -> true ;
           \+ pathlim -> assert(pathlim), fail ),
         \+ \+ prefix_unify([Pre=PreN]),
         prove_ec(ClaB,Cla1,MI,PI,I^V^FV:ClaB1,MI1),
         prove(ClaB1,MI1,[Lit:Pre|Path],[I|PI],PathLim,Lem,PreS1,VarS1,
           Set,Proof1), PreS3=[Pre=PreN|PreS1], append(VarS1,FV,VarS3)
       ),
       ( member(cut,Set) -> ! ; true ),
       prove(Cla,MI,Path,PI,PathLim,[Lit:Pre|Lem],PreS2,VarS2,Set,Proof2),
       append(PreS3,PreS2,PreS), append(VarS2,VarS3,VarS).

% extension clause (e-clause)
prove_ec((I^K)^V:ClaB,IV:Cla,MI,PI,ClaB1,MI1) :-
    append(MIA,[(I^K1)^V1:Cla1|MIB],MI), length(PI,K),
    ( ClaB=[J^K:[ClaB2]|_], member(J^K1,PI),
      unify_with_occurs_check(V,V1), Cla=[_:[Cla2|_]|_],
      append(ClaD,[J^K1:MI2|ClaE],Cla1),
      prove_ec(ClaB2,Cla2,MI2,PI,ClaB1,MI3),
      append(ClaD,[J^K1:MI3|ClaE],Cla3),
      append(MIA,[(I^K1)^V1:Cla3|MIB],MI1)
      ;
      (\+member(I^K1,PI);V\==V1;V\=[]^[]) ->
      ClaB1=(I^K)^V:ClaB, append(MIA,[IV:Cla|MIB],MI1) ).
```

Figure 3: Source code of the nanoCoP-i prover

`Cla`, `Mat`, and `Path` represent the subgoal clause C, the prefixed matrix M and the (active) $Path$. The *indexed path* `PathI` contains the indices of all clauses and matrices that contain literals of `Path`; it is used for calculating extension clauses. The list `Lem` is used for the lemmata rule and contains all literals that have been "solved" already [17]. `PreS` and `VarS` are lists of prefix equations and free (prefixed) term variables, respectively. `Set` is a list of options and may contain the elements "`cut`" and "`comp(`I`)`" for $I \in I\!N$, which are used to control the restricted backtracking technique [17]. This `prove` predicate succeeds iff there is an intuitionistic connection proof for the tuple (`Cla`, `Mat`, `Path`) with $|$`Path`$| <$ `PathLim`. In this case `Proof` returns a compact intuitionistic connection proof. The prefixed input matrix `Mat` has to be stored in Prolog's database (as explained above). The substitution σ is stored implicitly by Prolog. and also applied to the variables returned in `Proof`. The predicate `prove_ec(ClaB,Cla1,Mat,ClaB1,Mat1)` is used to calculate extension clauses (lines 39–49).

nanoCoP-i uses additional optimization techniques that are already used in the classical connection provers leanCoP [17] and nanoCoP [20]: *regularity* (line 19), *lemmata* (line 21), and restricted backtracking (line 36). *Restricted backtracking* is a very effective (but incomplete) technique for pruning the search space in connection calculi [17]. It is switched on if the list `Set` contains the element "`cut`". If it also contains "`comp(`I`)`" for $I \in I\!N$, then the proof search restarts again without restricted backtracking if the path limit `PathLim` exceeds I.

3.3 Prefix Unification

The source code of the prefix unification is shown in Figure 4. Each clause *R1* to *R10* corresponds to one of the rewrite rules defined in Figure 2. The predicate `tunify(S,[],T)` succeeds if the two prefixes `S` and `T` can be unified. The second argument contains the left part of the right prefix. The prefix variables are instantiated with a most general unifier; alternative unifiers are calculated via backtracking. As the skolemization technique is applied to *prefix* constants as well, a term unification with `unify_with_occurs_check` is required whenever a prefix constant is unified with another prefix constant or variable. The predicate `prefix_unify(G)` solves a *set* of prefix equations (lines *a*–*d*).

```
(a)    prefix_unify([]).
(b)    prefix_unify([S=T|G]) :- (-S2=S -> T2=T ; -S2=T, T2=S),
(c)                             flatten([S2,_],S1), flatten(T2,T1),
(d)                             tunify(S1,[],T1), prefix_unify(G).

(R1)   tunify([],[],[]).
(R2)   tunify([],[],[X|T])        :- tunify([X|T],[],[]).
(R3)   tunify([X1|S],[],[X2|T])   :- (var(X1) -> (var(X2), X1==X2);
                                      (\+var(X2), unify_with_occurs_check(X1,X2))),
                                      !, tunify(S,[],T).
(R4)   tunify([C|S],[],[V|T])     :- \+var(C), !, var(V), tunify([V|T],[],[C|S]).
(R5)   tunify([V|S],Z,[])         :- unify_with_occurs_check(V,Z), tunify(S,[],[]).
(R6)   tunify([V|S],[],[C1|T])    :- \+var(C1), V=[], tunify(S,[],[C1|T]).
(R7)   tunify([V|S],Z,[C1,C2|T]) :- \+var(C1), \+var(C2), append(Z,[C1],V1),
                                     unify_with_occurs_check(V,V1),
                                     tunify(S,[],[C2|T]).
(R8)   tunify([V,X|S],[],[V1|T])      :- var(V1), tunify([V1|T],[V],[X|S]).
(R9)   tunify([V,X|S],[Z1|Z],[V1|T]) :- var(V1), append([Z1|Z],[Vnew],V2),
                                          unify_with_occurs_check(V,V2),
                                          tunify([V1|T],[Vnew],[X|S]).
(R10)  tunify([V|S],Z,[X|T])      :- (S=[]; T[]; \+var(X)) ->
                                     append(Z,[X],Z1), tunify([V|S],Z1,T).
```

Figure 4: Source code of the prefix unification

4 Experimental Evaluation

The following evaluations were conducted on a 3.4 GHz Xeon system with 4 GB of RAM running Linux 3.13.0 and ECLiPSe Prolog 5.10. The CPU time limit was set to 10 seconds.

4.1 ILTP Library

The ILTP problem library [23] contains 2550 first-order formulae in various problem domains. Table 2 shows the number of proved problems of the ILTP library v1.1.2 for the intuitionistic theorem provers JProver, ileanTAP, ft, ileanCoP, and nanoCoP-i.

JProver [25] is based on a (simple) prefixed non-clausal connection calculus for intuitionistic first-order logic [12]; it is implemented in OCaml. ileanTAP [14] implements a prefixed free-variable tableau calculus for intuitionistic first-order logic; it is implemented in Prolog. ft [26] is a C implementation of an analytic tableau calculus for intuitionistic first-order logic and uses many additional optimization techniques . ileanCoP [15, 16] implements a prefixed *clausal* connection calculus for intuitionistic first-order logic and is implemented in Prolog. In order to make the results comparable to nanoCoP-i, the (Prolog) *core prover* of ileanCoP was used with the standard translation ("[nodef]") and the definitional translation ("[def]") into the (*prefixed*) clausal form. nanoCoP-i was tested with and without restricted backtracking technique, i.e. `Set=[]` and `Set=[cut,comp(6)]`, respectively.

Table 2: Results on ILTP library v1.1.2

	JProver	ileanTAP	ft (C)	— ileanCoP 1.2 —		—- nanoCoP-i 1.0 —-	
	11-2005	1.17	1.23	[nodef]	[def]	[]	[cut,comp(6)]
proved	250	303	328	529	560	**605**	**700**
0 to 1sec.	239	295	323	487	509	**546**	**588**
1 to 10sec.	11	8	5	42	51	**59**	**112**

nanoCoP-i proves more problems than both clausal translations of ileanCoP. The "full" version of ileanCoP1.2 (using the TPTP syntax translation of nanoCoP-i and a shell script to implement the strategy scheduling) proves 717 problems. The proofs found by nanoCoP-i are in general shorter than those found by ileanCoP. The *proof size* is the number of connections (and applications of the lemma rule). Compared to ileanCoP [nodef], 95% of the nanoCoP-i proofs are on average 30% shorter; 4% of the proofs are larger, 1% have the same size. Compared to ileanCoP [def], 97% of the nanoCoP-i proofs are on average 33% shorter; 2% of the proofs are larger, 1% have the same size.

4.2 TPTP Library

Table 2 shows the results on all 3644 first-order (FOF) problems of the TPTP library v3.3.0 [28].

Table 3: Results on TPTP library v3.3.0

	JProver	ileanTAP	ft (C)	— ileanCoP 1.2 —		—- nanoCoP-i 1.0 —-	
	11-2005	1.17	1.23	[nodef]	[def]	[]	[cut,comp(6)]
proved	177	251	260	636	652	**731**	**872**
0 to 1sec.	171	248	258	566	572	**636**	**717**
1 to 10sec.	6	3	2	70	80	**95**	**155**

Again, nanoCoP-i proves more problem than both clausal translations of ileanCoP. The "full" version of ileanCoP1.2 proves 932 problems. Compared to ileanCoP [nodef], 92% of the nanoCoP-i proofs are on average 28% shorter; 7% of the proofs are larger, 1% have the same size. Compared to ileanCoP [def], 94% of the nanoCoP-i proofs are on average 34% shorter; 5% of the proofs are larger, 1% have the same size.

5 Conclusion

This paper introduces a non-clausal connection calculus for intuitionistic first-order logic and nanoCoP-i, a compact implementation of this calculus. Using prefixed non-clausal matrices, the proof search works directly on the original structure of the input formula; no translation steps to any clausal or other normal form are required. This combines the advantages of more *natural* non-clausal tableau or sequent provers with the goal-oriented *efficiency* of connection provers.

Even though the non-clausal inferences introduce a slight overhead, nanoCoP-i outperforms both clausal form translations of the ileanCoP core prover on the ILTP and the TPTP problem library. It is expected that the integration of strategy scheduling into nanoCoP-i will also outperform the "full" ileanCoP prover. More than 90% of the returned non-clausal proofs are on average about 30% shorter than their clausal counterparts.

Both, the standard translation as well as a definitional translation [22] into clausal form not only increase the size of the formula, but also modify the structure of the original formula This makes it difficult to translate the resulting proof back into a proof of the original formula, an effect that has already been observed for classical logic [24].

By considering the intuitionistic substitution, the returned connection proof can be translated into an intuitionistic sequent proof [8], making nanoCoP-i an ideal tool to be used within interactive proof systems, such as Coq [3] , Isabelle [13], HOL [9] or NuPRL [6].

In contrast to the calculus used in nanoCoP-i, the non-clausal connection calculus used in JProver [25] does not add clause copies dynamically during the proof search. Instead they are added iteratively, which introduces a huge redundancy into the proof search. Hence, the performance of nanoCoP-i is significantly higher than the performance of JProver.

Future work include the adaption of the non-clausal connection calculus and the nanoCoP-i prover to other non-classical logics, such as modal or description logics, for which so far only clausal connection calculi exist [19]. Integrating search techniques into nanoCoP-i in order to obtain a decision procedure is another important task. Furthermore, optimization techniques that are used for classical logic, such as strategy scheduling [17], learning [11] and variable splitting [1], could be integrated into an intuitionistic non-clausal connection calculus as well.

References

[1] Antonsen, R., Waaler, A.: Liberalized variable splitting. Journal of Automated Reasoning 38, 3–30 (2007)

[2] Beckert, B., Posegga, J.: leanTAP: lean, tableau-based deduction. Journal of Automated Reasoning 15(3), 339–358 (1995)

[3] Bertot, Yves, Castéran, Pierre: Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions. Springer Heidelberg (2010)

[4] Bibel, W.: Matings in matrices. Communications of the ACM 26, 844–852 (1983)

[5] Bibel, W.: Automated Theorem Proving. 2nd edition. Vieweg, Wiesbaden (1987)

[6] Constable, R. et al.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, Upper Saddle River, N.J (1986)

[7] D. van Dalen. Intuitionistic Logic. In L. Goble (ed.): The Blackwell Guide to Philosophical Logic. Blackwell, Oxford (2001)

[8] Gentzen, G.: Untersuchungen über das logische Schließen. Mathematische Zeitschrift 39, 176–210, 405–431 (1935)

[9] Gordon, M. J. C. and Melham, T. F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press, New York (1993)

[10] Hähnle, R.: Tableaux and Related Methods. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 100–178. Elsevier, Amsterdam (2001)

[11] Kaliszyk, C., Urban, J.: FEMaLeCoP: Fairly Efficient Machine Learning Connection Prover. In: Davis, M. et al. (eds.) LPAR 2015. LNAI, vol. 9450, pp. 88–96. Springer, Heidelberg (2015)

[12] Kreitz, C., Otten, J.: Connection-based theorem proving in classical and non-classical logics. Journal of Universal Computer Science 5, 88–112 (1999)

[13] Nipkow, T., Wenzel, M. Paulson, L.: Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer, Heidelberg (2002)

[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. et al. (eds.) IJCAR 2008, LNAI, vol. 5195, pp. 283–291. Springer, Heidelberg (2008)

[17] Otten, J.: Restricting backtracking in connection calculi. AI Communications 23, 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. et al. (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. Springer, Heidelberg (2016)

[21] Otten, J., Bibel, W.: leanCoP: lean connection-based theorem proving. Journal of Symbolic Computation 36, 139–161 (2003)

[22] Plaisted, D., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation 2, 293–304 (1986)

[23] Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic. Journal of Automated Reasoning 38, 261–271 (2007)

[24] Reis, G.: Importing SMT and connection proofs as expansion trees. In: Kaliszyk, C., Paskevich, A. (eds.) 4th Workshop on Proof eXchange for Theorem Proving (PxTP15), EPTCS 186, pp. 3–10 (2015)

[25] Schmitt, S. et al.: JProver: Integrating Connection-based Theorem Proving into Interactive Proof Assistants. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001, LNAI, vol. 2083, pp. 421–426. Springer, Heidelberg (2001)

[26] Sahlin, D., Franzen, T., Haridi, S.: An Intuitionistic Predicate Logic Theorem Prover. Journal of Logic and Computation 2(5), 619–656 (1992)

[27] Stickel, M.: A Prolog technology theorem prover: implementation by an extended Prolog compiler. Journal of Automated Reasoning 4, 353–380 (1988)

[28] Sutcliffe, G.: The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. Journal of Automated Reasoning 43(4), 337–362 (2009)

[29] Waaler, A.: Connections in Nonclassical Logics. In A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, pp 1487–1578. Elsevier, Amsterdam (2001)

[30] Wallen, L. A.: Automated Deduction in Nonclassical Logics. MIT Press, Cambridge (1990)