Intelligenza Artificiale 9 (2015) 109–125
DOI 10.3233/IA-150082
IOS Press

Nested sequent calculi and theorem proving for normal conditional logics: The theorem prover NESCOND

Nicola Olivetti[a] and Gian Luca Pozzato[b,1,*]

Abstract. In this paper we focus on proof methods and theorem proving for normal conditional logics, by describing nested sequent calculi as well as a theorem prover for them. We first present some nested sequent calculi, recently introduced, for the basic conditional logic CK and some of its significant extensions with axioms ID, MP and CEM. We also describe a calculus for the flat fragment of the conditional logic CK+CSO+ID, which corresponds to Kraus, Lehmann and Magidor's cumulative logic **C**. The calculi are internal, cut-free and analytic. Next, we describe NESCOND, a Prolog implementation of these calculi in the style of leanT^{A}P. We finally present an experimental comparison between our theorem prover NESCOND and other known theorem provers for conditional logics. Our tests show that the performances of NESCOND are promising and in all cases better, with the only exception of systems including CEM, than those ones of the other provers. The program NESCOND, as well as all the Prolog source files, are available at http://www.di.unito.it/∼pozzato/nescond/.

Keywords: Conditional logics, nonmonotonic reasoning, theorem proving, nested sequent calculi, proof theory

1. Introduction

Conditional logics are extensions of classical logic by a *conditional* operator $\Rightarrow$. They can be seen as a generalization of (multi)modal logics, where the modality $\Rightarrow$ is indexed by a formula of the same language. Conditional logics have a long history [27, 28]. They have been introduced in order to formalize a kind of hypothetical reasoning, where a conditional formula $A \Rightarrow B$ is used to formalize a sentence like "if A were the case then B" that cannot be captured by classical logic with material implication. One original motivation was to formalize *counterfactual sentences*, i.e. conditionals of the form "if A were the case then B would be the case", where A is false [16].

Over the years, conditional logics have been studied in various fields of artificial intelligence and knowledge representation. Just to mention a few, they have been used:

– to reason about prototypical properties and defeasible inheritance [11]: the understanding of a conditional $A \Rightarrow B$ is "the As have typically the property B" or "normally, the As are Bs". In the conditional logic introduced in [11] one can consistently express, for instance, that birds typically fly, whereas penguins are birds that do not fly, as follows:

$$\forall x(Bird(x) \Rightarrow Fly(x))$$
$$\forall x(Penguin(x) \rightarrow Bird(x))$$
$$\forall x(Penguin(x) \rightarrow \neg Fly(x))$$

[1]The author is partially supported by the Project "ExceptionOWL: Nonmonotonic Extensions of Description Logics and OWL for defeasible inheritance with exceptions", progetti di ricerca di Ateneo anno 2014, Call 01 "Excellent Young PI", Torino_call2014_L1_111, Università di Torino and Compagnia di San Paolo.

*Corresponding author: Gian Luca Pozzato, Dipartimento di Informatica, Universitá degli Studi di Torino, c.so Svizzera 185, 10149 Torino, Italy. Tel. +39 011 670 6848; Fax: +39 011 75 16 03; E-mail: gianluca.pozzato@unito.it.

It is easy to observe that the knowledge base obtained from the one above by replacing the conditional operator $\Rightarrow$ with the classical implication $\rightarrow$, that is to say by replacing $\forall x(Bird(x) \Rightarrow Fly(x))$ with $\forall x(Bird(x) \rightarrow Fly(x))$, is consistent only if there are no penguins;

- to provide an axiomatic foundation of nonmonotonic reasoning [8, 26]: in this context, a conditional $A \Rightarrow B$ is read as "in normal circumstances, if A then B". More in detail, in [26] the authors propose an axiomatization of the properties of a nonmonotonic consequence relation: their system comprises nonmonotonic assertions of the form $A \mid\sim B$, interpreted as "B is a plausible conclusion of A". It turns out that all forms of inference studied in [26] are particular cases of well-known conditional axioms [10] (in this respect the language of [26] is just a fragment of conditional logics);
- to model database update [18, 22] and belief change and revision [17] in [22] the author presents a conditional logic, more precisely a variant of the logic **VCU** introduced in [27], to formalize knowledge-update as defined by Katsuno and Mendelzon in [25]; in [18] the authors show a tight correspondence between AGM revision systems [14] and a specific conditional logic, called BCR. The connection between revision/update and conditional logics can be intuitively explained in terms of the so-called Ramsey Test (RT): $A \Rightarrow B$ "holds" in a knowledge base K if and only if B "holds" in the knowledge base K revised/updated with A;
- to reason about access control policies [15]: in this context, the statement A **says** B, intuitively meaning that a user/program A *asserts* B to hold in the system, can be naturally expressed by a conditional $A \Rightarrow B$. In [15] it is shown that constructive conditional logics can also be used to reason about transfer of authority from one principal to another, in particular, the fact that a principal A *speaks for* another principal B, can be defined as $B \Rightarrow A$;
- to model hypothetical queries in deductive databases and logic programming [13]: the logic CK+ID is the basis of the logic programming language CondLP, in which one can express hypothetical goals of the form[1] $Load\_Gun \Rightarrow (Shoot \Rightarrow Dead)$, and the idea is that the hypothetical goal succeeds if $Dead$ succeeds in the state "revised" first by $Load\_Gun$ and then by $Shoot$;
- to formalize causal inference and reasoning about action execution in diagnosis [16, 29] and planning [33, 20]: in this context, a conditional formula $A \Rightarrow B$ is interpreted as "A causes B";
- to formalize epistemic change in a multi-agent setting and in some kind of epistemic "games" [3, 7]: here, each conditional operator expresses the "conditional beliefs" of an agent.

All conditional logics enjoy a possible world semantics, with the intuition that a conditional formula $A \Rightarrow B$ is true in a world w if B is true in the set of worlds that are most similar to/closest to/as normal as w given the formula A. Since there are different ways of formalizing "the set of worlds similar/closest/..." to a given world, there are expectedly rather different semantics for conditional logics, from the most general selection function semantics to the stronger sphere semantics.

In this work, we focus our attention on the more general *selection function semantics*: models are equipped by an accessibility relation (called selection function) for each formula of the language, with the restriction that logically equivalent formulas have the same accessibility relation. Here a conditional formula $A \Rightarrow B$ is true in a world w if B is true in all the worlds selected by the accessibility relation/selection function for w and A. We consider the basic normal conditional logic CK and its extensions with ID, MP and CEM, as well as the cumulative logic **C** introduced in [26] which corresponds to the *flat* fragment (i.e., without nested conditionals) of the conditional logic CK+CSO+ID. The logic CK is the basic system of conditional logics based on the selection function semantics: it plays the same role of the system K in modal logics. Extensions of CK are obtained by assuming suitable properties on the selection function. The axiom

$$\text{MP} \qquad (A \Rightarrow B) \rightarrow (A \rightarrow B)$$

captures a *conditional modus ponens*, stating a relation between the material implication and the conditional one. Systems allowing MP are those whose models are such that if a formula A holds in a world w, then w is always closest to w itself given A. The axiom

$$\text{ID} \qquad A \Rightarrow A$$

captures the idea that the result of supposing A is always successful. In systems allowing ID, it is imposed that, given a formula A, A holds in all the worlds more similar to w given A. The axiom

$$\text{CEM} \qquad (A \Rightarrow B) \vee (A \Rightarrow \neg B)$$

[1] [1]The example is inspired by the classic Yale's Shooting problem.

is the *conditional excluded middle*. Models of systems allowing CEM are those whose selection function selects at most *one* world given a formula A. The axiom

$$\text{CSO } ((A \Rightarrow B) \wedge (B \Rightarrow A)) \rightarrow ((A \Rightarrow C) \rightarrow (B \Rightarrow C))$$

captures a kind of conditional independence of the antecedent of a conditional formula. In models of systems allowing CSO, if both $A \Rightarrow B$ and $B \Rightarrow A$ hold in a world w, the worlds closest to w are the same given both A and B. As mentioned before, the fragment of the logic CK+CSO+ID not allowing nested conditionals corresponds to the logic **C** introduced in [26] by Kraus, Lehmann and Magidor (KLM) in order to formalize the basic properties of nonmonotonic reasoning. Their work, that has been early recognized as a landmark, led to a classification of nonmonotonic consequence relations, determining a hierarchy of stronger and stronger systems. The so called *KLM properties* have been widely accepted as the "conservative core" of default reasoning. The role of KLM logics is similar to the role of AGM postulates in Belief Revision [14]: they give a set of postulates for default reasoning that any concrete reasoning mechanism should satisfy. As an example, in the conditional logic CK+CSO+ID, if we know that normally Italian people love soccer, and that Italian people loving soccer usually watch soccer matches on television, we can infer that typical Italian people watch soccer matches on television, that is to say the following formula is valid:

$$(Italian \Rightarrow SoccerLover) \wedge$$
$$(Italian \wedge SoccerLover \Rightarrow WatchSoccerOnTV)$$
$$\rightarrow (Italian \Rightarrow WatchSoccerOnTV).$$

Despite their relevance, from the point of view of proof-theory and automated deduction, conditional logics have not achieved a state of the art comparable with, say, the one of modal logics, where there are well-established calculi, whose proof-theoretical and computational properties are well-understood, and efficient theorem provers have been implemented.

In this work we first describe *nested sequent calculi* $\mathcal{NS}$ for propositional conditional logics, recently introduced in [1, 2]. Nested sequent calculi [9, 12, 21, 24] are a natural generalization of ordinary sequent calculi where sequents are allowed to occur within sequents. A nested sequent always corresponds to a formula of the language, so that we can think of the rules as operating "inside a formula", combining subformulas rather than just combining outer occurrences of formulas as in ordinary sequent calculi. Since the calculi stay within the language, they can be classified as "internal", as opposed to "external" calculi which make use of additional ingredients (e.g. labels and relations among them) to encode the semantics into the syntax.

The calculi $\mathcal{NS}$ are rather natural and all rules have a fixed number of premises. Moreover, they can be used to obtain a decision procedure for the respective logics by imposing some restrictions preventing redundant applications of rules. In all cases, we get a PSPACE upper bound, a bound that for CK and its extensions with ID and MP is optimal (but not for CK+CEM that is known to be coNP). For flat CK+CSO+ID = cumulative logic **C** we also get a PSPACE bound, we are not aware of a better upper bound for this logic.

Furthermore, we introduce an implementation of $\mathcal{NS}$ calculi in Prolog. The program, called NESCOND, is inspired by the methodology introduced by the system $\mathsf{lean}T^AP$ [5], even if it does not fit its style in a strict sense. The basic idea is that each axiom or rule of the nested sequent calculi is implemented by a Prolog clause of the program. The resulting code is therefore simple and compact: the implementation of NESCOND for CK consists of only 6 predicates, 24 clauses and 34 lines of code.

We finally provide extensive experimental results by comparing NESCOND with CondLean [30] and GOALD$\mathcal{U}$CK [31], to the best of our knowledge the only existing provers for conditional logics, as well as with CoLoSS [23], a generic-purpose theorem prover for coalgebraic modal logics which can handle also basic conditional logics. Concerning the logic CK+CSO+ID, we have also compared the performances of NESCOND with KLMLean [19], a theorem prover for the cumulative logic **C**.

Performances of NESCOND are promising, in all cases superior to those of the other provers, with the exception of systems supporting CEM. We can conclude that nested sequent calculi are not only a useful proof theoretical tool, but they can be the basis of efficient theorem proving for conditional logics.

This paper is an extended and revised version of the work in [32], where we have proposed a preliminary presentation of NESCOND, not including the description of the systems with CEM and for the cumulative logic **C**, and reporting only basic experimental results.

2. Normal conditional logics

We consider a propositional conditional language $\mathcal{L}$ over a set ATM of propositional variables. Formulas of $\mathcal{L}$ are built as usual: $\bot$, $\top$ and the propositional variables

in ATM are *atomic formulas*; if A and B are formulas, then $\neg A$ and $A \otimes B$ are *compound formulas*, where $\otimes \in \{\wedge, \vee, \rightarrow, \Rightarrow\}$. Atomic formulas and compound formulas are formulas of $\mathcal{L}$.

We adopt the *selection function semantics*. We consider a non-empty set of possible worlds $\mathcal{W}$. Intuitively, the selection function f selects, for a world w and a formula A, the set of worlds of $\mathcal{W}$ which are *closest* to w given the information A. A conditional formula $A \Rightarrow B$ holds in a world w if the formula B holds in *all the worlds selected by f for w and A*.

Definition 1. [Selection function semantics] A model is a triple $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$ where:

- $\mathcal{W}$ is a non-empty set of worlds;
- $f : \mathcal{W} \times 2^{\mathcal{W}} \longmapsto 2^{\mathcal{W}}$ is the *selection function*;
- $[\,]$ is the *evaluation function*, which assigns to an atomic formula $P \in ATM$ the set of worlds where P is true, and is extended to compound formulas as follows:
  - $[\top] = \mathcal{W}$;
  - $[\bot] = \emptyset$;
  - $[\neg A] = \mathcal{W} - [A]$;
  - $[A \wedge B] = [A] \cap [B]$;
  - $[A \vee B] = [A] \cup [B]$;
  - $[A \rightarrow B] = [B] \cup (\mathcal{W} - [A])$;
  - $[A \Rightarrow B] = \{w \in \mathcal{W} | f(w, [A]) \subseteq [B]\}$

A formula $F \in \mathcal{L}$ is valid in a model $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$, and we write $\mathcal{M} \models F$, if $[F] = \mathcal{W}$. A formula $F \in \mathcal{L}$ is valid, and we write $\models F$, if it is valid in every model, that is to say $\mathcal{M} \models F$ for every $\mathcal{M}$.

We have defined f taking $[A]$ rather than A (i.e. $f(w, [A])$ rather than $f(w, A)$) as an argument; this is equivalent to define f on formulas, i.e. $f(w, A)$ but imposing that if $[A] = [A']$ in the model, then $f(w, A) = f(w, A')$. This condition is called *normality*.

The semantics above characterizes the *basic conditional system*, called CK, where no specific properties of the selection function are assumed. An axiomatization of CK is given by ($\vdash$ denotes provability in the axiom system):

- any axiomatization of the classical propositional calculus (prop)
- If $\vdash A$ and $\vdash A \rightarrow B$, then $\vdash B$ (Modus Ponens)
- If $\vdash A \leftrightarrow B$ then $\vdash (A \Rightarrow C) \leftrightarrow (B \Rightarrow C)$ (RCEA)
- If $\vdash (A_1 \wedge \cdots \wedge A_n) \rightarrow B$ then $\vdash (C \Rightarrow A_1 \wedge \cdots \wedge C \Rightarrow A_n) \rightarrow (C \Rightarrow B)$ (RCK)

Moreover, we consider the standard extensions of the basic system CK shown in Fig. 1.

3. Nested sequent calculi $\mathcal{NS}$ for conditional logics

In this section we recall nested sequent calculi $\mathcal{NS}$ introduced in [1]. S is an abbreviation for CK{+X}, and X $\in$ {CEM, ID, MP, ID+MP, CEM+ID}.

The specificity of nested sequent calculi is to allow inferences that apply within sequents. A nested sequent Γ is defined inductively as follows:

- a formula of $\mathcal{L}$ is a nested sequent;
- if A is a formula and Γ is a nested sequent, then $[A : \Gamma]$ is a nested sequent;
- a finite multiset of nested sequents is a nested sequent.

A nested sequent can be displayed as

$$A_1, \ldots, A_m, [B_1 : \Gamma_1], \ldots, [B_n : \Gamma_n],$$

where $n, m \geq 0$, $A_1, \ldots, A_m$, $B_1, \ldots, B_n$ are formulas and $\Gamma_1, \ldots, \Gamma_n$ are nested sequents.

A nested sequent can be directly interpreted as a formula, just replace "," by $\vee$ and ":" by $\Rightarrow$. More explicitly, the interpretation of a nested sequent

$$A_1, \ldots, A_m, [B_1 : \Gamma_1], \ldots, [B_n : \Gamma_n]$$

is inductively defined by the formula

$$\mathcal{F}(\Gamma) = A_1 \vee \ldots \vee A_m \vee (B_1 \Rightarrow \mathcal{F}(\Gamma_1)) \vee \ldots \vee (B_n \Rightarrow \mathcal{F}(\Gamma_n)).$$

In order to introduce the rules of the calculus, we need the notion of context. Intuitively a context denotes a "hole", a *unique* empty position, within a sequent that can be filled by a nested sequent. We use the symbol $(\,)$ to denote the empty context. A context is defined inductively as follows:

- $\Gamma(\,) = \Lambda, (\,)$ is a context;
- if $\Sigma(\,)$ is a context, then $\Gamma(\,) = \Lambda, [A : \Sigma(\,)]$ is a context,

where Λ is a nested sequent and A is a formula. Finally, we define the result of filling "the hole" of a context by a sequent. Let $\Gamma(\,)$ be a context and Δ be a sequent, then the sequent obtained by filling the context by Δ, denoted by $\Gamma(\Delta)$ is defined as follows:

SYSTEM	AXIOM	MODEL CONDITION
ID	$A \Rightarrow A$	$f(w,[A]) \subseteq [A]$
CEM	$(A \Rightarrow B) \vee (A \Rightarrow \neg B)$	$\mid f(w,[A]) \mid \leq 1$
MP	$(A \Rightarrow B) \rightarrow (A \rightarrow B)$	$w \in [A]$ implies $w \in f(w,[A])$
CSO	$(A \Rightarrow B) \wedge (B \Rightarrow A) \rightarrow ((A \Rightarrow C) \rightarrow (B \Rightarrow C))$	$f(w,[A]) \subseteq [B]$ and $f(w,[B]) \subseteq [A]$ implies $f(w,[A]) = f(w,[B])$

Fig. 1. Some standard extensions of CK.

$$\frac{}{\Gamma(P,\neg P)}\,(AX)_{P \in ATM} \qquad \frac{}{\Gamma(\top)}\,(AX_\top) \qquad \frac{}{\Gamma(\neg\bot)}\,(AX_\bot) \qquad \frac{\Gamma(A)}{\Gamma(\neg\neg A)}\,(\neg)$$

$$\frac{\Gamma(A) \quad \Gamma(B)}{\Gamma(A \wedge B)}\,(\wedge^+) \qquad \frac{\Gamma(\neg A,\neg B)}{\Gamma(\neg(A \wedge B))}\,(\wedge^-) \qquad \frac{\Gamma(A,B)}{\Gamma(A \vee B)}\,(\vee^+) \qquad \frac{\Gamma(\neg A) \quad \Gamma(\neg B)}{\Gamma(\neg(A \vee B))}\,(\vee^-)$$

$$\frac{\Gamma(\neg A,B)}{\Gamma(A \rightarrow B)}\,(\rightarrow^+) \qquad \frac{\Gamma(A) \quad \Gamma(\neg B)}{\Gamma(\neg(A \rightarrow B))}\,(\rightarrow^-) \qquad \frac{\Gamma(\neg(A \Rightarrow B),[C:\Delta,\neg B]) \quad A,\neg C \quad C,\neg A}{\Gamma(\neg(A \Rightarrow B),[C:\Delta])}\,(\Rightarrow^-)$$

$$\frac{\Gamma([A:B])}{\Gamma(A \Rightarrow B)}\,(\Rightarrow^+) \qquad \frac{\Gamma(\neg(A \Rightarrow B),A) \quad \Gamma(\neg(A \Rightarrow B),\neg B)}{\Gamma(\neg(A \Rightarrow B))}\,(MP) \qquad \frac{\Gamma([A:\Delta,\Sigma],[B:\Sigma]) \quad A,\neg B \quad B,\neg A}{\Gamma([A:\Delta],[B:\Sigma])}\,(CEM)$$

$$\frac{\Gamma([A:\Delta,\neg A])}{\Gamma([A:\Delta])}\,(ID) \qquad \frac{\Gamma,\neg(A \Rightarrow B),[C:\Delta,\neg B] \quad \Gamma,\neg(A \Rightarrow B),[A:C] \quad \Gamma,\neg(A \Rightarrow B),[C:A]}{\Gamma,\neg(A \Rightarrow B),[C:\Delta]}\,(CSO)$$

Fig. 2. The nested sequent calculi $\mathcal{NS}$.

$$\dfrac{\dfrac{\dfrac{}{[A:A,\neg A]}\,(AX)}{[A:A]}\,(ID)}{A \Rightarrow A}\,(\Rightarrow^+)$$

Fig. 3. A derivation of the axiom ID in $\mathcal{N}$CK+ID. Thanks to the modularity of the $\mathcal{NS}$ calculi, the same derivation can also be obtained in $\mathcal{N}$CK+ID+MP , in $\mathcal{N}$CK+CEM+ID, and in $\mathcal{NC}_{KLM}$.

$$\dfrac{\dfrac{\dfrac{}{\neg(A \Rightarrow B),\neg A,B,A}\,(AX) \qquad \dfrac{}{\neg(A \Rightarrow B),\neg A,B,\neg B}\,(AX)}{\neg(A \Rightarrow B),\neg A,B}\,(MP)}{\neg(A \Rightarrow B),A \rightarrow B}\,(\rightarrow^+)}{(A \Rightarrow B) \rightarrow (A \rightarrow B)}\,(\rightarrow^+)$$

Fig. 4. A derivation of the axiom MP in $\mathcal{N}$CK+MP and $\mathcal{N}$CK+ID+MP.

- if $\Gamma(\,) = \Lambda, (\,)$, then $\Gamma(\Delta) = \Lambda, \Delta$;
- if $\Gamma(\,) = \Lambda, [A : \Sigma(\,)]$ then $\Gamma(\Delta) = \Lambda, [A : \Sigma(\Delta)]$.

Figure 2 shows nested sequent calculi $\mathcal{NS}$.

$$\dfrac{\dfrac{\dfrac{}{[A:B,\neg B],[A:\neg B]}\,(AX) \qquad \dfrac{}{A,\neg A}\,(AX) \qquad \dfrac{}{\neg A,A}\,(AX)}{[A:B],[A:\neg B]}\,(CEM)}{[A:B],A \Rightarrow \neg B}\,(\Rightarrow^+)}{A \Rightarrow B,A \Rightarrow \neg B}\,(\Rightarrow^+)}{(A \Rightarrow B) \vee (A \Rightarrow \neg B)}\,(\vee^+)$$

Fig. 5. A derivation of the axiom CEM in $\mathcal{N}$CK+CEM and $\mathcal{N}$CK+CEM+ID.

As usual, we say that a nested sequent Γ is *derivable* in $\mathcal{NS}$ if it admits a *derivation*. A derivation is a tree whose nodes are nested sequents. A branch is a sequence of nodes $\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$ such that each node Γ_i is obtained from its immediate successor Γ_{i-1} by applying *backward* a rule of $\mathcal{NS}$, having Γ_{i-1} as the conclusion and Γ_i as one of its premises. A branch is closed if one of its nodes is an instance of axioms (AX), $(AX_\top)$, $(AX_\bot)$, otherwise it is open. We say that a tree is closed if all its branches are closed. A nested sequent Γ has a derivation in $\mathcal{NS}$ if there is a closed tree having Γ as the root.

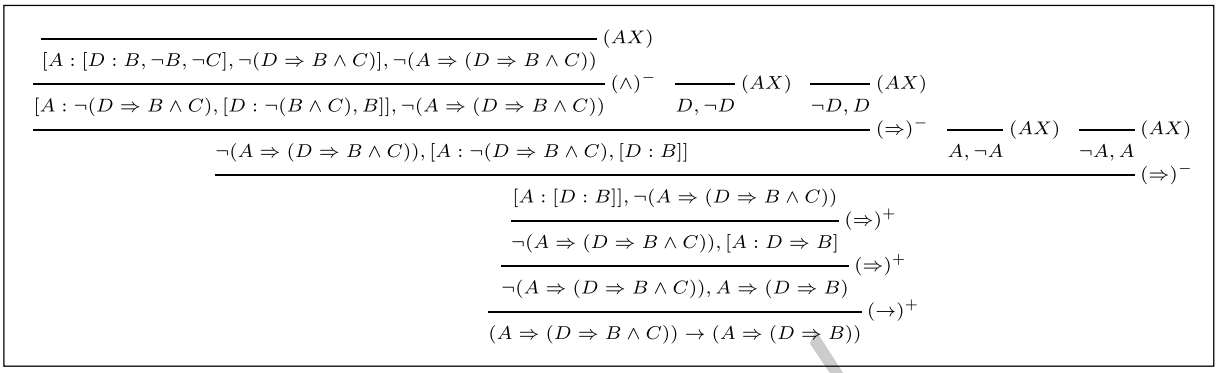

Fig. 6. A derivation of $(A \Rightarrow (D \Rightarrow B \wedge C)) \rightarrow (A \Rightarrow (D \Rightarrow B))$ in $\mathcal{N}$CK provided by the theorem prover NESCOND.

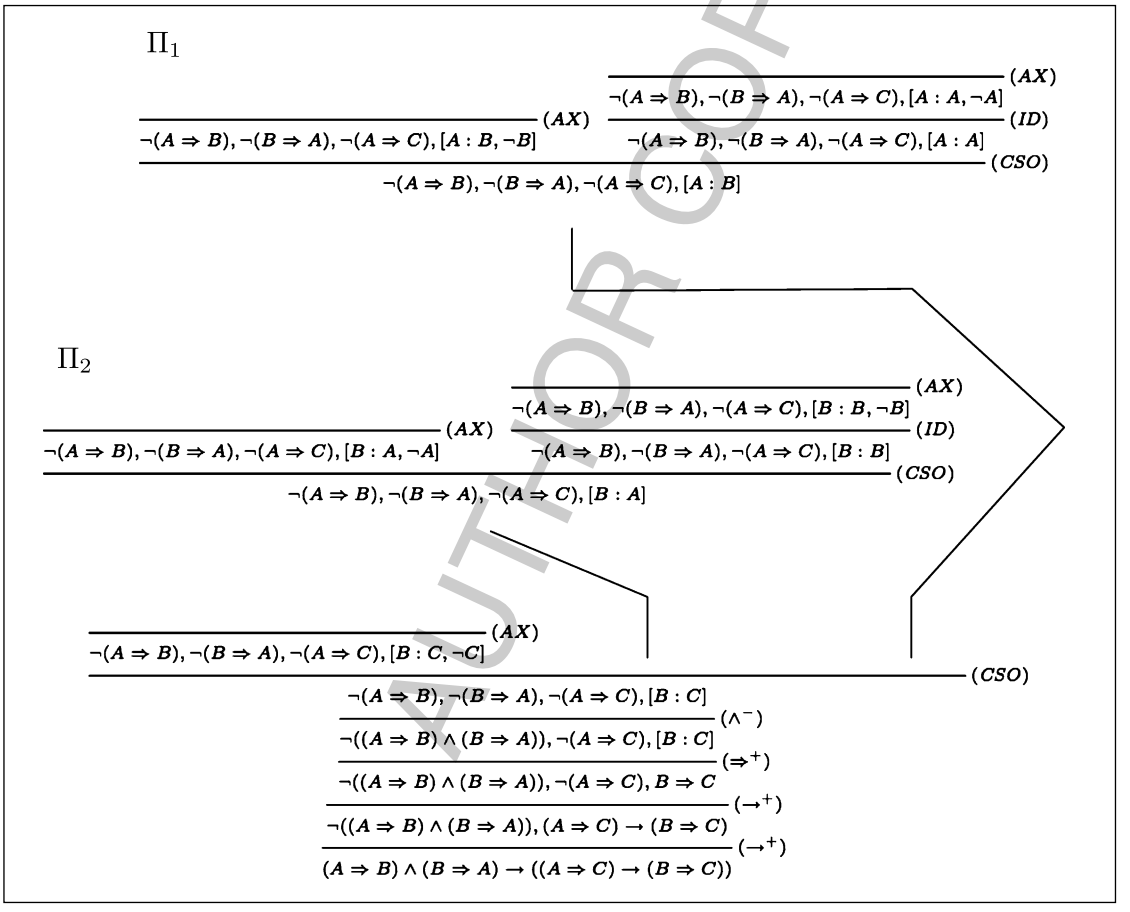

Fig. 7. A derivation of the axiom CSO. Notice that the sequent $\neg(A \Rightarrow B), \neg(B \Rightarrow A), \neg(A \Rightarrow C), [A : A]$ occurs twice as a premise of the application of the rule (CSO) in the derivation Π_1; therefore, in order to increase readability, we only show it once. This is the reason why the application of (CSO) in Π_1 has two premises (and not three). The same for $\neg(A \Rightarrow B), \neg(B \Rightarrow A), \neg(A \Rightarrow C), [B : B]$, occurring twice in Π_2.

In Figs. 3, 4 and 5 we show derivations in the calculi $\mathcal{NS}$ of the axioms (ID), (MP), and (CEM), respectively. Figure 6 shows a derivation of the CK valid formula $(A \Rightarrow (D \Rightarrow B \wedge C)) \rightarrow (A \Rightarrow (D \Rightarrow B))$ found by the theorem prover NESCOND described in Section 4.

Conditional logic	Calculus	$(\Rightarrow^+)$	$(\Rightarrow^-)$	(ID)	(MP)	(CEM)	(CSO)
CK	$\mathcal{N}$CK	☑	☑	☐	☐	☐	☐
CK+CEM	$\mathcal{N}$CK+CEM	☑	☑	☐	☐	☑	☐
CK+ID	$\mathcal{N}$CK+ID	☑	☑	☑	☐	☐	☐
CK+MP	$\mathcal{N}$CK+MP	☑	☑	☐	☑	☐	☐
CK+ID+MP	$\mathcal{N}$CK+ID+MP	☑	☑	☑	☑	☐	☐
CK+CEM+ID	$\mathcal{N}$CK+CEM+ID	☑	☑	☑	☐	☑	☐
KLM's cumulative **C**	$\mathcal{N}\mathbf{C}_{\mathrm{KLM}}$	☑	☐	☑	☐	☐	☑

Fig. 8. Nested sequent calculi $\mathcal{NS}$: conditional logics and adopted rules.

We have also provided a nested sequent calculus for the flat fragment, i.e. without nested conditionals, of CK+CSO+ID, corresponding to Kraus, Lehmann and Magidor's *cumulative* logic **C** [26]. The rules of the calculus, called $\mathcal{NC}_{KLM}$, are those ones of $\mathcal{N}$CK+ID (restricted to the flat fragment) where the rule $(\Rightarrow^-)$ is replaced by the rule (CSO). In Fig. 7 we show a derivation in $\mathcal{NC}_{KLM}$ of the (CSO) axiom.

Figure 8 summarizes the rules of each nested sequent calculus of $\mathcal{NS}$. Axioms (AX), $(AX_\top)$, and $(AX_\bot)$, as well as rules for propositional connectives, belong to all the systems.

Nested sequent calculi $\mathcal{NS}$ are sound and complete with respect to the semantics for the respective logics, a proof can be found in Section 3 in [2].

Theorem 1. *The nested sequent calculi $\mathcal{NS}$ are sound and complete for the respective logics, i.e. a formula F of $\mathcal{L}$ is valid in CK$\{$+X$\}$ if and only if it is derivable in* $\mathcal{N}$CK$\{$+X$\}$.

3.1. Termination and complexity of $\mathcal{NS}$

As usual, in order to obtain a decision procedure for the conditional logics under consideration, we have to control the application of the rules $(\Rightarrow^-)$, (CSO), (MP), (CEM), and (ID) that otherwise may be applied infinitely often in a backward proof search, since their principal formula is copied into the respective premise(s). In detail, we have the following results, whose proofs can be found in [2].

Proposition 1. *The calculi $\mathcal{NS}$ with the following restrictions on the application of the rules:*

- $(\Rightarrow^-)$ *can be applied only once to each formula* $\neg(A \Rightarrow B)$ *with a context* $[C : \Delta]$ *in each branch;*
- (ID) *can be applied only once to each context* $[A : \Delta]$ *in each branch;*
- (MP) *can be applied only once to each formula* $\neg(A \Rightarrow B)$ *in each branch.*
- *in the system CK+CSO+ID, the rule* (CSO) *can be applied only once to each formula* $\neg(A \Rightarrow B)$ *with a context* $[C : \Delta]$ *in each branch*

are sound, complete and terminating.

For systems with (CEM), we need a more complicated mechanism. Intuitively, the rule is applied only to sequents containing only atomic formulas and negated conditionals to which $(\Rightarrow^-)$ has been applied as much as possible. We need the following definitions.

Definition 2. Given a nested sequent

$$\Gamma = A_1, \ldots, A_n, [B_1 : \Gamma_1], \ldots, [B_m : \Gamma_m]$$

and a formula F, we define $F \in^* \Gamma$ if either $F = A_i$ for some $i \in \{1, 2, \ldots, n\}$ or $F \in^* \Gamma_j$ for some $j \in \{1, 2, \ldots, m\}$.

Definition 3. (CEM-reduced sequent) Given a nested sequent $\Gamma = A_1, \ldots, A_n, [B_1 : \Gamma_1], \ldots, [B_m : \Gamma_m]$ occurring in a derivation Π, we say that Γ is *CEM-reduced* if the following conditions hold:

- for each formula F such that $F \in^* \Gamma$, either F is a literal, i.e. $F = P$ or $F = \neg P$, where $P \in ATM$, or F is a negated conditional $\neg(C \Rightarrow D)$;
- for each negated conditional $\neg(C \Rightarrow D) \in^* \Gamma$ and for each (sub)context $[C' : \Delta]$ occurring in Γ, if $C, \neg C'$ and $C', \neg C$ are derivable, then the rule $(\Rightarrow^-)$ has been applied to $\neg(C \Rightarrow D)$ by using $[C' : \Delta]$ in Π.

Definition 4. Given two sequents $\Gamma = A_1, \ldots, A_n, [B_1 : \Gamma_1], \ldots, [B_m : \Gamma_m]$ and Δ, we define $\Gamma \subseteq^* \Delta$ if (i) $A_i \in \Delta$, for each $i = 1, 2, \ldots, n$ and (ii) there is $[B_i : \Delta_i] \in \Delta$ such that $\Gamma_i \subseteq^* \Delta_i$, for each $i = 1, 2, \ldots, m$.

For instance, given $\Gamma = C, C, [D : E, E, F], [D : E, F], [G : [H : K, K]]$ and $\Delta = C, [D : E, E, F, H], [G : [H : K, M]]$, it holds that $\Gamma \subseteq^* \Delta$.

In [2] it is shown that:

Proposition 2. *The calculi $\mathcal{NS}$ for systems with CEM with the following restrictions on the application of the rule* (CEM) *to* $\Gamma([A : \Delta], [B : \Sigma])$:

(i) $\Gamma([A : \Delta], [B : \Sigma])$ *is CEM-reduced*
(ii) *not* $\Delta \subseteq^* \Sigma$.

are sound, complete and terminating.

The terminating calculi can be used to provide a PSPACE decision procedure for their respective logics [2]. As mentioned in the Introduction, this bound is optimal for logics not including CEM.

4. Design of NESCOND

In this section we present a Prolog implementation of the nested sequent calculi $\mathcal{NS}$. The program, called NESCOND (NESted sequent calculi for CONDitional logics), is inspired by the "lean" methodology of leanTAP [5]. The basic idea is that the Prolog program comprises a set of clauses, each one of which implements a sequent rule or axiom of $\mathcal{NS}$. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism.

NESCOND represents a nested sequent with a Prolog list of the form:

```
[F_1, F_2, ..., F_m,
[[A_1,Gamma_1],AppliedCond_1], ...,
 [[A_n,Gamma_n],AppliedCond_n]] ]
```

where `F_i` and `A_i` are formulas of $\mathcal{L}$ whereas `Gamma_i` are Prolog lists representing nested sequents. Elements of the form

```
[[A,Gamma],AppliedCond]]
```

are pairs where `[A,Gamma]` represents a context $[A : \Gamma]$, while `AppliedCond` is a Prolog list `[A_1=>B_1,A_2=>B_2,...,A_k=>B_k]` used in order to implement the restriction on the application of the rule $(\Rightarrow^{-})$ to ensure termination, by keeping track of the negated conditionals to which the rule $(\Rightarrow^{-})$ has been already applied by using `[A,Gamma]` in the current branch.

Symbols $\top$ and $\bot$ are represented by constants `true` and `false`, respectively, and connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\Rightarrow$ are represented by `!`, `^`, `v`, `->`, and `=>`.

As an example, the Prolog list

```
[p, q, !(p => q), [[p, [q v !p,
[[p,[p => r]],[]], !r]],[p => q]],
[[q, [p, !p]],[]]]
```

represents the nested sequent

$$P, Q, \neg(P \Rightarrow Q), [P : Q \vee \neg P, [P : P \Rightarrow R], \neg R], [Q : P, \neg P].$$

Furthermore, the list `[p => q]` in the leftmost context is used to represent the fact that, in a backward proof search, the rule $(\Rightarrow^{-})$ has already been applied to $\neg(P \Rightarrow Q)$ by using $[P : Q \vee \neg P, [P : P \Rightarrow R], \neg R]$.

4.1. Auxiliary predicates

In order to manipulate formulas "inside" a sequent, NESCOND makes use of the three following auxiliary predicates:

- `deepMember(+Formulas,+NS)` succeeds if and only if there is a nested sequent occurring in `NS` containing formulas of `Formulas`, that is to say either (i) the nested sequent `NS` contains formulas in the list `Formulas` or (ii) there exists a `[[A,Delta],AppliedConditionals]` in `NS` such that `deepMember (Formulas, Delta)` succeeds.
- `deepSelect(+Formulas,+NS,-NewNS)` operates exactly as `deepMember`, however it removes the formulas of the list `Formulas` by replacing them with a placeholder `hole`; the output term `NewNS` matches the resulting sequent.
- `fillTheHole(+NewNS,+Formulas, -ResultingNS)` replaces `hole` in `NewNS` with the formulas in the list `Formulas`. `ResultingNS` is the output term matching the result.

4.2. NESCOND for CK

The calculi $\mathcal{NS}$ are implemented by the predicate

```
prove(+NS,-ProofTree).
```

This predicate succeeds if and only if the nested sequent represented by the list `NS` is derivable. When it succeeds, the output term `ProofTree` matches with a representation of the derivation found by the prover. For instance, in order to prove that the formula $(A \Rightarrow (B \wedge C)) \rightarrow (A \Rightarrow B)$ is valid in CK, one queries NESCOND with the goal: `prove([(a => b ^ c) -> (a => b)],ProofTree).` Each clause of `prove` implements an axiom or rule of $\mathcal{NS}$. To search for a derivation of a nested sequent Γ, NESCOND proceeds as follows. First of all, if Γ is an axiom, the goal will succeed immediately by using one of the following clauses for the axioms:

```
prove(NS,tree(ax)):-deepMember([F,!F],NS),!.
prove(NS,tree(axt)):-deepMember([top],NS),!.
prove(NS,tree(axb)):-deepMember([!bot],NS),!.
```

implementing (AX), $(AX_{\top})$ and $(AX_{\bot})$, respectively. It is worth noticing that NESCOND applies Lemma 3.6 in [2], asserting that, given any formula F and any context $\Gamma(\,)$, the sequent $\Gamma(F, \neg F)$ is derivable in $\mathcal{NS}$.

Therefore, the above Prolog clause can be applied to any formula F, generalizing axiom (AX) which is restricted to atomic formulas only.

If Γ is not an instance of the axioms, then the first applicable rule will be chosen, e.g. if a nested sequent in Γ contains a formula `A v B`, then the clause implementing the $(\vee^+)$ rule will be chosen, and NESCOND will be recursively invoked on the unique premise of $(\vee^+)$. NESCOND proceeds in a similar way for the other rules. The ordering of the clauses is such that the application of the branching rules is postponed as much as possible.

As an example, the clause implementing $(\Rightarrow^-)$ is as follows:

```
1. prove(NS,tree(condn,A,B,Sub1,Sub2,Sub3)):-
2.     deepSelect([!(A => B),
                [[C,Delta],AppliedConditionals]],
                                NS,NewNS),
3.     \+member(!(A => B),AppliedConditionals),
4.     prove([A,!C],Sub2),
5.     prove([C,!A],Sub3),!,
6.     fillTheHole(NewNS,[!(A => B),
                    [[C,[!B|Delta]],
          [!(A => B)|AppliedConditionals]]],
               DefNS),
7.     prove(DefNS,Sub1).
```

In line 2, the auxiliary predicate `deepSelect` is invoked in order to find both a negated conditional $\neg(A \Rightarrow B)$ and a context $[C : \Delta]$ in the sequent (even in a nested subsequent). In this case, such formulas are replaced by the placeholder `hole`. Line 3 implements the restriction on the application of $(\Rightarrow^-)$ in order to guarantee termination: the rule is applied only if $\neg(A \Rightarrow B)$ does not belong to the list `Applied-Conditionals` of the selected context. In lines 4, 5 and 7, NESCOND is recursively invoked on the three premises of the rule. In line 7, NESCOND is invoked on the premise in which the context $[C : \Delta]$ is replaced by $[C : \Delta, \neg B]$. To this aim, in line 6 the auxiliary predicate `fillTheHole(+NS,+Formulas,-ResultingNS)` is invoked to replace the `hole` introduced by `deepSelect`, with the negated conditional $\neg(A \Rightarrow B)$, which is copied into the premise, and the context $[C : \Delta, \neg B]$, whose list of `Applied-Conditionals` is updated by adding the formula $\neg(A \Rightarrow B)$ itself, in order to prevent further, useless applications in the current branch.

As mentioned, the last argument of the predicate `prove` is an output term corresponding to a functor `tree` storing information about the derivation found by the prover, that will be used by the graphical interface to display a closed tree of a valid sequent. For axioms, this functor just recalls the name of the axioms themselves, namely it is `tree(ax)` for (AX), `tree(axt)` for $(AX_\top)$ and `tree(axb)` for $(AX_\bot)$. In the example of $(\Rightarrow^-)$, the term is `tree(condn,A,B,Sub1,Sub2,Sub3)`, where:

- the constant `condn` is used to store the name of the applied rule, in this case $(\Rightarrow^-)$ (conditional+negative);
- `A` and `B` represent the principal formula to which the rule is applied, here representing the negated conditional $\neg(A \Rightarrow B)$;
- `Sub1`, `Sub2` and `Sub3` are the sub-`trees` of the three premises of the rule, returned by the recursive invocations of `prove`.

Similarly for the other rules. It is worth noticing that, since the rules of the calculi are invertibile (see Lemma 3.9 and the discussion following it in [2]), we can use cut `!` in line 5 to avoid the creation of a useless backtracking point.

The implementation of the calculi for extensions of CK with axioms ID and MP are similar to the one of CK. The main differences are described in the following sections.

4.3. NESCOND for extensions with ID

In these systems, contexts are triples `[Context, AppliedConditionals, AllowID]`. The third element `AllowID` is a flag used in order to implement the restriction on the application of the rule (ID), namely the rule is applied to a context only if `AllowID=true`, as follows:

```
prove(NS,tree(id,A,SubTree)):-
    deepSelect([[
        [A,Delta],AppliedConditionals,true]]],
                          NS,NewNS),!,
    fillTheHole(NewNS,[[[A,[!A|Delta]],
        AppliedConditionals,false]]],DefNS),
    prove(DefNS,SubTree).
```

When (ID) is applied to

```
[Context,AppliedConditionals,true],
```

then the predicate `prove` is invoked on the unique premise of the rule represented by the list `DefNS`, and the flag is set to `false` in order to avoid multiple applications in a backward proof search.

4.4. NESCOND for extensions with MP

The restriction on the application of the rule (MP) is implemented by equipping the predicate `prove` by a third argument, a Prolog list called `AppliedMP`, keeping track of the negated conditionals to which the rule has already been applied in the current branch. The clause of `prove` implementing (MP) is:

```
1. prove(NS,AppliedMP,tree(mp,A,B,Sub1,
     Sub2)):-
2.    deepSelect([!(A => B)],NS,NewNS),
3.     \+member(A => B,AppliedMP),!,
4.     fillTheHole(NewNS,[A,!(A => B)],NS1),
5.     fillTheHole(NewNS,[!B,!(A => B)],NS2),
6.     prove(NS1,[A => B|AppliedMP],Sub1),
7.     prove(NS2,[A => B|AppliedMP],Sub2).
```

The rule is applicable to a formula $\neg(A \Rightarrow B)$ only if `[A => B]` does not belong to `AppliedMP` (line 3). When (MP) is applied, then `A => B` is added to `AppliedMP` in the recursive calls of `prove` on the premises of the rule (lines 6 and 7).

4.5. NESCOND for extensions with CEM

The implementation of NESCOND for systems allowing the axiom CEM is quite different from the other ones. In particular, these systems implement the much more complicated mechanism needed to ensure termination described in Proposition 2.

Condition (i) in Proposition 2 is guaranteed by the fact that the clause implementing (CEM) is the last one in the Prolog program: this ensures that the other rules are applied as much as possible, then the rule (CEM) is applied only to CEM-reduced sequents. The clause implementing the rule (CEM) is as follows:

```
1. prove(NS,tree(cem,A,B,Sub1,Sub2,Sub3)):-
2.    deepSelect([[[A,Delta],ApplCond1],
           [[B,Sigma],ApplCond2]],NS,NewNS),
3.    notSequentIncluded(Delta,Sigma),
4.    prove([A,!B],Sub2),
5.    prove([B,!A],Sub3),
6.    append(Delta,Sigma,ResDelta),
7.    fillTheHole(NewNS,
                [[[A,ResDelta],ApplCond1],
              [[B,Sigma],ApplCond2]],DefNS),
8.    prove(DefNS,Sub1).
```

In line 3 the following predicate `notSequentIncluded` is invoked:

```
1. notSequentIncluded(Delta,Sigma):-
2.   partitionNS(Delta,Formulas,Contexts),!,
3.    ((member(F,Formulas),\+member(F,Sigma),!)
4.    ;
5.    (extractSubContexts(Sigma,List),
6.    member([[A,Gamma],_],Contexts),
7.    notSeqList(A,Gamma,List))).
```

This predicate ensures condition (ii) in Proposition 2, namely that not $\Delta \subseteq^* \Sigma$. In line 2 the list representing Δ is partitioned into formulas $F_1, \ldots, F_n$ and contexts $[A_1 : \Delta_1], \ldots, [A_m : \Delta_m]$. Two alternatives are taken into account by the disjunction in line 4:

- in line 3, the theorem prover checks whether there is a formula $F \in \Delta$ such that $F \notin \Sigma$. If this is the case, then $\Delta \not\subseteq^* \Sigma$ and the predicate succeeds;
- otherwise, the predicate tries to find a context $[A : \Gamma] \in \Delta$ such that, for all contexts $[B_i : \Sigma_i] \in \Sigma$, we have that $\Gamma \not\subseteq^* \Sigma_i$: this is again enough to conclude that $\Delta \not\subseteq^* \Sigma$, and the predicate `notSequentIncluded(Delta,Sigma)` succeeds. More in detail, an auxiliary predicate `extractSubContexts(Sigma,List)` first builds the list `List` of contexts $[B_i : \Sigma_i] \in \Sigma$ (line 5); then, in line 6 a context $[A : \Gamma] \in \Delta$ is selected by invoking the standard `member` predicate. Last, another auxiliary predicate `notSeqList` is invoked to check whether $[A : \Gamma]$ is not included in $[B_i : \Sigma_i] \in \Sigma$, by recursively invoking `notSequentIncluded` in order to check whether $\Gamma \not\subseteq^* \Sigma_i$.

If the predicate `notSequentIncluded (Delta, Sigma)` succeeds, i.e. $\Delta \not\subseteq^* \Sigma$, then ($CEM$) is applicable, and the predicate `prove` is recursively invoked on its three premises (lines 4, 5, and 8).

4.6. NESCOND for $\mathcal{NC}_{KLM}$

The implementation of the calculus for the flat fragment of CK+CSO+ID, corresponding to Kraus, Lehmann and Magidor's cumulative logic **C**differs from the one for CK+ID in three main respects:

1. the rule ($\Rightarrow^-$) is replaced by (CSO). This does not make use of the predicate `deepSelect` to "look inside" a sequent to find the principal formulas $\neg(A \Rightarrow B)$ and $[C : \Delta]$: since the calculus only deals with the *flat* fragment of the logic under consideration, the shown principal formulas are directly selected from the current sequent by easy membership tests (standard Prolog predicates `member` and `select`), without searching inside other contexts;

2. contexts are pairs

   ```
   [Context, AllowID]
   ```

   rather than triples

   ```
   [Context, AppliedConditionals,
                AllowID],
   ```

   since the control of the application of the (CSO) rule is more complicated and then delegated to a further argument of the `prove` predicate, see point 3 here below;
3. the predicate `prove` has three arguments:

   ```
   prove(+NS,+AppliedCSO,-ProofTree).
   ```

   where `AppliedCSO` is a list of pairs

   ```
   [!(A=>B),[C: Delta]]
   ```

   used in order to control the backward application of the (CSO) rule and representing the fact that (CSO) has been already applied to a context $[C : \Delta]$ in the current branch by using the negated conditional $\neg(A \Rightarrow B)$ as a principal formula.

The clause of `prove` implementing (CSO) is as follows:

```
1. prove(NS,AppliedCSO,
           tree(cso,A,B,Sub1,Sub2,Sub3)):-
2.     member(!(A => B),NS),
3.     select([[C,Delta],AllowID],NS,NewNS),
4.     \+member(!B,Delta),
5.     \+excludeCSO(!(A => B),[C,Delta],
          AppliedCSO),!,
6.    prove([[[C,[A]],true]|NewNS],
      [[!(A => B),[C,Delta]]|AppliedCSO],Sub1),
7.    prove([[[A,[C]],true]|NewNS],
        [[!(A => B),[C,Delta]]|AppliedCSO],
          Sub2),
8.    prove([[[C,[(!B)|Delta]],AllowID]|NewNS],
            [[!(A => B),[C,Delta]]|AppliedCSO],
              Sub3).
```

In line 2 the predicate `member` checks whether a negated conditional $\neg(A \Rightarrow B)$ belongs to the current nested sequent; if this is the case, in line 3 the `select` predicate is invoked to select and remove a context $[C : \Delta]$ for building the premise containing $[C : \Delta, \neg B]$ on which `prove` is recursively invoked in line 8. Similarly to the implementation of the rule $(\Rightarrow^{-})$ in the other systems, in lines 6 and 7 the `prove` predicate is recursively invoked on the other two premises $\Gamma, \neg(A \Rightarrow B), [C : A]$ and $\Gamma, \neg(A \Rightarrow B), [A : C]$, respectively.

In line 4 the predicate `\+member(!B,Delta)` is invoked in order to check whether the application of (CSO) is useful, by avoiding the situation in which the predicate `prove` is further invoked in line 8 on the same sequent, namely when $\neg B$ already belongs to Δ, therefore one of the premise of (CSO) is $\Gamma, \neg(A \Rightarrow B), [C : \Delta, \neg B, \neg B]$, the same as $\Gamma, \neg(A \Rightarrow B), [C : \Delta, \neg B]$ since contraction is admissible (see Lemma 3.11 in [2]). In line 5, the auxiliary predicate `excludeCSO` is invoked in order to check whether the rule (CSO) can be applied to $[C : \Delta]$ by using $\neg(A \Rightarrow B)$ as a principal formula. This predicate succeeds if (CSO) has been already applied in the current branch to a context $[C : \Delta_1]$, such that $\Delta_1 \subseteq \Delta$, by using the same conditional $\neg(A \Rightarrow B)$, in other words if there is a pair `[!(A => B), [C,Delta1]]` in the list `AppliedCSO)`. In the implementation of the rule $(\Rightarrow^{-})$ in CK+ID, this mechanism is implemented by means of the list `AppliedConditionals` belonging to each context: indeed, as mentioned before, when $(\Rightarrow^{-})$ is applied to a sequent containing $\neg(A \Rightarrow B), [C : \Delta]$, then the `prove` predicate is recursively invoked on a premise containing $\neg(A \Rightarrow B), [C : \Delta, \neg B]$, whose context is such that $\neg(A \Rightarrow B)$ is added to its list of `AppliedConditionals`. This is no longer applicable in systems containing the (CSO) rule, because all its three premises include a context, and two contexts are "new" ones, not coming from the context to which the rule is applied. Consider an application of (CSO) to a sequent $\Gamma, \neg(A \Rightarrow B), [C : \Delta]$, then the premises contain $[C : A]$ (line 6), $[A : C]$ (line 7) and $[C : \Delta, \neg B]$ (line 8), respectively. Notice that $[C : A]$ and $[A : C]$ are "new", they do not include $[C : \Delta]$. The list of negated conditionals already used in order to apply (CSO) to $[C : A]$ should be empty: however, a further (backward) application of (CSO) to $[C : A]$ introduces a "new" context $[C : A]$ in one of the premises, leading to a non-terminating proof search.

This is why we need a more complicated machinery, making use of a further auxiliary predicate `excludeCSO` as well as of a third argument `AppliedCSO` in the `prove` predicate. It is worth noticing that the pair $\langle \neg(A \Rightarrow B), [C : \Delta] \rangle$ is added to the list `AppliedCSO` in lines 6, 7 and 8, implementing the restriction on the application of (CSO) of Proposition 1.

NESCOND is available at the web address `http://www.di.unito.it/~pozzato/nescond/`. It also comprises a graphical user interface corresponding to a web application. In detail, the user can query the theorem prover by means of a php web page, typing in a suitable text box the sequent whose validity is under consideration. The php application asks the SWI Prolog engine (`http://www.swi-prolog.org`) to check whether such a sequent is valid by invoking

the predicate `prove` described above. If the sequent is valid, the application builds a pdf file containing a proof tree for the sequent. Some pictures of NESCOND are presented in Figs. 15 and 16.

5. Performance of NESCOND

The performances of NESCOND are promising. We have tested it by running SICStus Prolog 4.0.2 on an Apple MacBook Pro, 2.7 GHz Intel Core i7, 8GB RAM machine[2], by comparing its performances with all the other theorem provers for conditional logics known in the literature.

5.1. NESCOND vs CondLean vs GOALD$\mathcal{UC}K$

We have compared the performances of NESCOND with the ones of two other provers for conditional logics: CondLean 3.2, implementing labelled sequent calculi [30], and the goal-directed procedure GOALD$\mathcal{UC}K$ [31].

5.1.1. Randomly generated sequents

We have tested the three provers over a set of 6000 randomly generated sequents, obtaining the results shown in Fig. 9. Obviously, in order to obtain a relevant result, we have run the same set of 6000 sequents on all the three theorem provers.

We have first considered sequents whose formulas are built from 15 different atomic variables and have a high level of nesting (10), obtaining results in Fig. 9(a): NESCOND is not able to answer only in 0,05% of cases (1 sequent over 2000) within 10 seconds, whereas both GOALD$\mathcal{UC}K$ and CondLean are not able to conclude anything in more than 3% of cases (60 tests over 2000). If the time limit is extended to 2 minutes, we have obtained the results in Fig. 9(b): NESCOND answers in 100% of cases, whereas its two competitors have still at least the 1,30% of timeouts.

We have then considered sequents with a lower level of nesting (3) and whose formulas contain only 3 different atomic variables, obtaining the results in Fig. 9(c). The difference is surprisingly much more significant: with a time limit of 10 seconds, NESCOND is not able to answer only in 9,05% of cases, whereas both CondLean and GOALD$\mathcal{UC}K$ are not able to conclude in 16,55% and in 51,15% of cases, respectively. This phenomena can be explained by the fact that NESCOND is faster than the other provers to find 355 not valid sequents (against 17 of CondLean and 34 of GOALD$\mathcal{UC}K$) within the fixed time limit.

5.1.2. Valid sequents

We have tested NESCOND and its two competitors over sets of valid sequents. In absence of a set of acknowledged benchmarks for conditional logics, we have considered known significant structures of valid sequents in CK and all the considered extensions. Concerning CK, we have considered 88 valid formulas obtained by translating K valid formulas, obtained by replacing $\Box A$ with $\top \Rightarrow A$, and $\Diamond A$ with $\neg(\top \Rightarrow \neg A)$. These formulas have been provided by Heuerding and used to test the theorem prover ModLeanTAP [4] by Beckert and Goré.

The experimental results are shown in Fig. 10. The performances of NESCOND are encouraging also in this kind of tests. Considering CK, NESCOND is not able to give an answer in less than 10 seconds only in 5 cases over 88, against the 8 of CondLean and the 12 of GOALD$\mathcal{UC}K$; the number of timeouts drops to 4 if we extend the time limit to 1 minute, whereas this extension has no effect on the competitors (still 8 and 12 timeouts). We have similar results also for extensions of CK as shown in Fig. 11 here we have not included GOALD$\mathcal{UC}K$ since the most formulas adopted do not belong to the fragments admitting goal-directed proofs [31].

These results show that the performances of NESCOND are encouraging, probably better than the ones of the other existing provers for conditional logics, and this also holds for extensions of CK. For systems not allowing CEM, NESCOND gives an answer in 95% of the tests (all of them are valid formulas) in less than 1ms. The performances worsen in systems with CEM because of the overhead of the termination mechanism.

5.2. NESCOND vs CoLoSS

We have also compared the performances of NESCOND with the ones of CoLoSS, the Coalgebraic Logic Satisfiability Solver, a generic-purpose theorem prover for modal logics introduced in [23]. CoLoSS is an Haskell implementation of a uniform polynomial space algorithm to decide satisfiability for modal logics that are amenable to coalgebraic semantics, including

[2]It is worth noticing that the experimental results presented in this section have been obtained by running SICStus Prolog, whereas, as mentioned at the end of Section 4, the Prolog engine which is the core of the web application available at http://www.di.unito.it/~pozzato/nescond/ is implemented in SWI Prolog.

conditional logics. Also in this case, we distinguish tests over randomly generated formulas and tests over significant examples.

5.2.1. Randomly generated sequents

We have tested both NESCOND and CoLoSS over 200 randomly generated CK formulas, corresponding to sequents with only one formula, obtaining the results presented in Fig. 12. For CoLoSS, we have considered an "Optimized" version, using dynamic programming plus a restriction to connected pairs. CoLoSS kills tasks requiring more than 2 minutes. It is easy to observe that the performances of NESCOND are better than the ones of the competitor: it gives an answer for all generated formulas within 5 seconds, whereas CoLoSS in 5 seconds is able to conclude only in 66% of the examples. Furthermore, in some cases (about 5%), CoLoSS either crashes with a stack space overflow exception or reaches the 2 minutes time limit.

5.2.2. Valid sequents

We have tested both NESCOND and CoLoSS over the set of formulas available at CoLoSS web site (http://www.informatik.uni-bremen.de/cofi/CoLoSS/), used by the authors to witness the performances of their prover. In all the available examples, NESCOND answers in less than 1ms, whereas CoLoSS needs 776 ms on average to conclude its computation (in two examples, CoLoSS requires more than 2 seconds).

5.3. NESCOND for $\mathcal{NC}_{KLM}$ vs KLMLean for cumulative logic **C**

Last, we have compared the performances of the implementation of NESCOND for $\mathcal{NC}_{KLM}$ with those of KLMLean 2.0, a SICStus Prolog implementation of a tableau calculus for **C** introduced in [19] and available at http://www.di.unito.it/∼pozzato/klmlean%202.0/. Again, the performances of NESCOND seem to be significantly better than those ones of its opponent; as in the other cases, we distinguish tests over randomly generated formulas and tests over significant examples.

5.3.1. Randomly generated sequents

We have tested both NESCOND and KLMLean over 4000 randomly generated CK+ID+CSO sequents, corresponding to sets of formulas of **C**. We have considered different combinations of the following parameters: number of different propositional variables, number of conditional formulas (positive and negative), and fixed time limit. Experimental results are presented in Fig. 13. In tables (a) and (b) we have tested the two provers over formulas with 8 conditionals: they both need at least 5 seconds to obtain a number of timeouts under 90%; NESCOND is not able to answer in 59% of cases, whereas KLMLean needs more time in the 87% of cases. Tables (c) and (d) show the performances over formulas containing 25 conditionals: in this case, we can observe that the performances of NESCOND are significantly better: by extending the time limit to 30 seconds, NESCOND is not able to answer only in 19% of cases, whereas KLMLean has still 88% of timeouts.

5.3.2. Valid sequents

We have tested both NESCOND and KLMLean over a set of valid and significant formulas (Fig. 14). Also in this case, NESCOND if faster than KLMLean: after 5 seconds NESCOND is able to answer in more than 70% of cases, whereas KLMLean is able to answer only in 50% of cases.

(a)

number of tests: 2000	Prop. vars: 15, Depth: 10, Timeout: 10 s		
	yes	no	timeout
GoalDUCK	75,65%	21,35%	3,00%
CondLean	77,75%	19,10%	3,15%
NESCOND	77,75%	22,20%	0,05%

(b)

number of tests: 2000	Prop. vars: 15, Depth: 10, Timeout: 2 min		
	yes	no	timeout
GoalDUCK	74,50%	23,65%	1,85%
CondLean	74,90%	23,80%	1,30%
NESCOND	75,40%	24,60%	0,00%

(c)

number of tests: 2000	Prop. vars: 3, Depth: 3, Timeout: 10 s		
	yes	no	timeout
GoalDUCK	47,15%	1,7%	51,15%
CondLean	77,6%	0,85%	21,55%
NESCOND	73,2%	17,75%	9,05%

Fig. 9. NESCOND vs CondLean vs GOALDUCK over a set of 6000 random sequents.

Timeouts

	100ms	1s	10s	1m
CondLean	11	11	8	8
GoalDUCK	19	16	12	12
NESCOND	11	9	5	4

Fig. 10. NESCOND vs CondLean: timeouts over 88 valid formulas for CK.

Timeouts for extensions of CK

	1ms	1s	10s
CondLean	48,08%	36,54%	28,85%
NESCOND	38,46%	28,85%	26,92%

Timeouts for systems without CEM

	1ms	1s	10s
CondLean	19,23%	15,38%	13,46%
NESCOND	5,77%	0,00%	0,00%

Timeouts for systems CEM and CEM+ID

	1ms	1s	10s
CondLean	51,72%	37,93%	27,59%
NESCOND	58,62%	51,72%	48,28%

Fig. 11. NESCOND vs CondLean: timeouts over valid formulas for extensions of CK.

NESCOND vs CoLoSS

Theorem provers	1ms	100ms	5s	1m	2m
CoLoSS	36%	42,5%	66%	95,5%	95,5%
NESCOND	89,5%	91,5%	100%	100%	100%

Fig. 12. Percentage of answers by NESCOND and CoLoSS over 200 random formulas. Notice that in CoLoSS formulas which take more than 2 minutes are killed.

NESCOND vs KLMLean 2.0 over random formulas

(a)

number of tests: 1000	Prop. vars: 3, # conditionals: 8, Timeout: 2s		
	yes	no	timeout
KLMLean 2.0	0%	0%	100%
NESCOND	4,5%	0%	95,5%

(b)

number of tests: 1000	Prop. vars: 3, # conditionals: 8, Timeout: 5s		
	yes	no	timeout
KLMLean 2.0	9,06%	3,75%	87,19%
NESCOND	37,81%	2,50%	59,69%

(c)

number of tests: 1000	Prop. vars:10, #conditionals: 25, Timeout: 5s		
	yes	no	timeout
KLMLean 2.0	6,00%	5,50%	88,50%
NESCOND	29,50%	2,50%	68,00%

(a)

number of tests: 1000	Prop. vars:10, #conditionals:25, Timeout:30s		
	yes	no	timeout
KLMLean 2.0	8,00%	4,00%	88,00%
NESCOND	53,33%	27,33%	19,34 %

Fig. 13. Percentage of answers by NESCOND and KLMLean over 4000 random formulas.

NESCOND vs KLMLean 2.0 over valid formulas

	1ms	1s	5s	2m
KLMLean 2.0	30,77%	46,15%	50,00%	57,69%
NESCOND	69,23%	73,08%	73,08%	76,92%

Fig. 14. Percentage of answers by NESCOND and KLMLean over valid formulas.

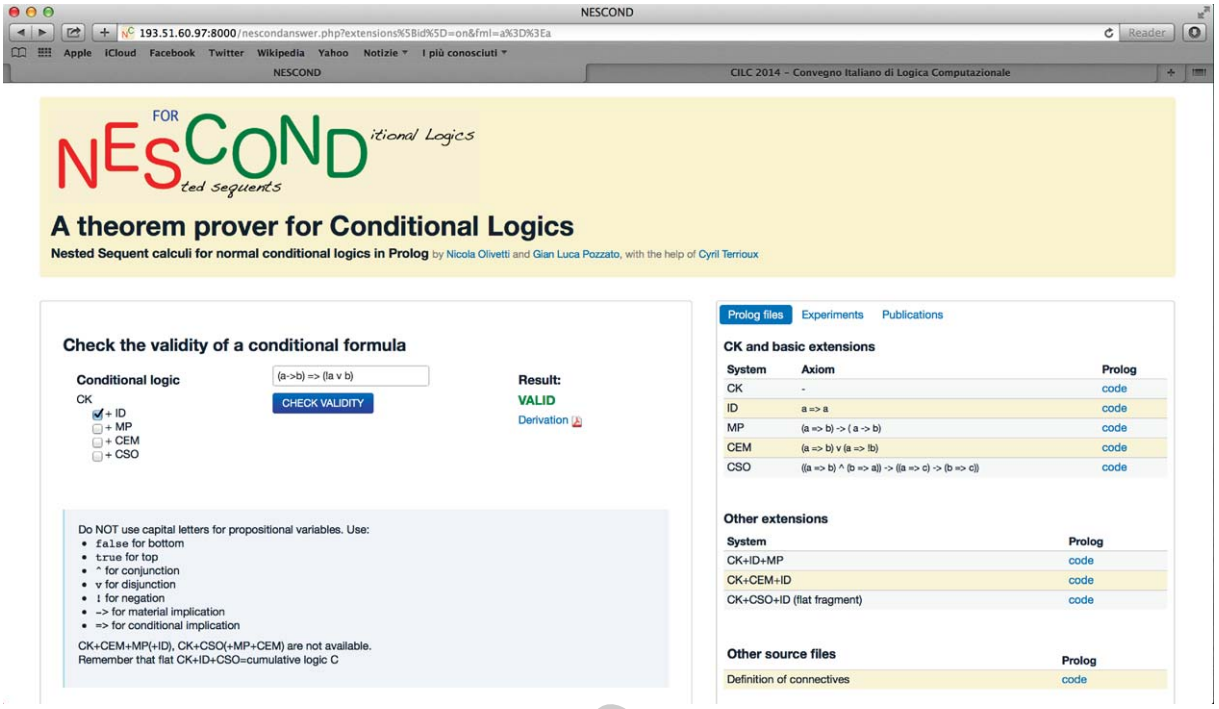

Fig. 15. The home page of NESCOND http://www.di.unito.it/∼pozzato/nescond/

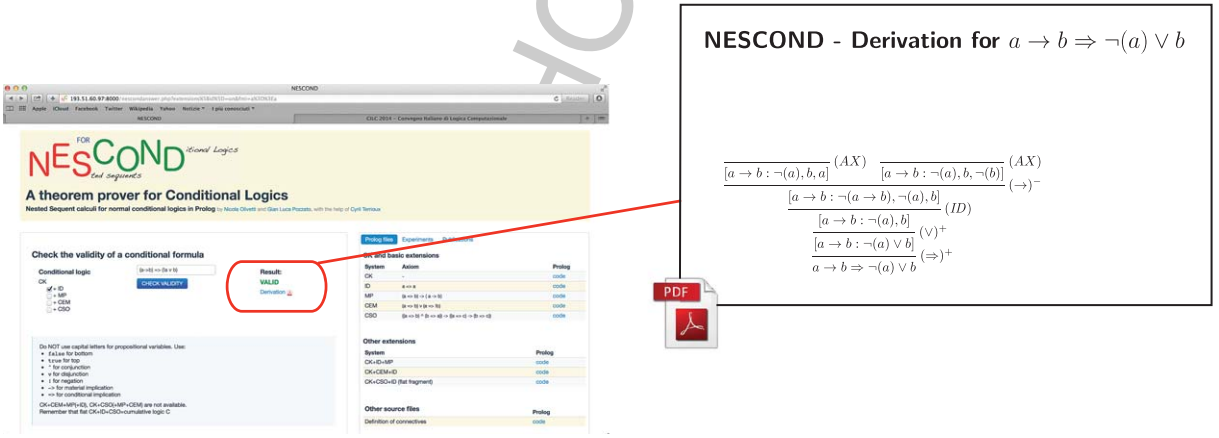

Fig. 16. The pdf file generated by NESCOND for a valid sequent.

6. Conclusions and future issues

We have recalled nested sequent calculi $\mathcal{NS}$ for the basic normal conditional logic CK and some extensions of it with combinations of ID, MP, and CEM, namely all combinations except CK+MP+CEM(+ID). The calculi are analytic and they can be used to obtain a decision procedure, in some cases of optimal complexity. We have also recalled a nested sequent calculus, called $\mathcal{NC}_{KLM}$, for the cumulative logic **C**, corresponding to the flat fragment of CK+CSO+ID for which no internal calculus seem to be known so far.

Moreover, we have presented NESCOND (http://www.di.unito.it/∼pozzato/nescond/), a theorem

prover implementing nested sequent calculi $\mathcal{NS}$. We have compared NESCOND to other theorem provers for conditional logics. Statistics obtained so far show that NESCOND performs reasonably well and in all cases (except for logics including CEM) it outperforms the other theorem provers. We can conclude that nested sequent calculi do not only provide elegant and natural calculi for conditional logics, but they are also well-suited for developing efficient theorem provers for them.

In [6] the authors provide a sound and complete embedding of some conditional logics into classical higher-order logic (HOL), allowing to exploit available HOL reasoners for automated theorem proving in conditional logics. Intuitively, the embedding is based on the natural correspondence between the selection function semantics and HOL. The basic system CK and extensions with ID, MP, and CEM are considered, whereas CSO is not taken into account. The results of some experiments, obtained by running different HOL reasoners over few examples, are also provided.

The authors of [6] introduce a methodology to obtain theorem provers for conditional logics, rather than a "concrete" prover ready to be queried by the users: they do not even compare their solution to existing provers CondLean and GOALD$\mathcal{UCK}$; experimental results for systems combining different conditional axioms are not provided; last, the implemented reasoner is not available and cannot be accessed neither for download nor by means of a web application. As the same authors point out, their objective is to show that the proposed theoretical embedding of conditional logics into HOL can have practical benefits, however HOL is, in general, undecidable, and termination/complexity results for the implemented prototype exploiting HOL reasoners are not mentioned.

In future research we aim to extend $\mathcal{NS}$ and NESCOND to other systems of conditional logics. To this regard, we strongly conjecture that adding a rule for the axiom (CS) $(A \wedge B) \rightarrow (A \Rightarrow B)$ will be enough to cover the whole cube of the extensions generated by axioms (ID), (MP), (CEM) and (CS).

References

[1] R. Alenda, N. Olivetti and G.L. Pozzato. Nested Sequent Calculi for Conditional Logics. In Jérôme Mengin Luis Fariñas del Cerro, Andreas Herzig, editor, *Logics in Artificial Intelligence - 13th European Conference, JELIA 2012*, volume 7519 of *Lecture Notes in Artificial Intelligence (LNAI)*, Toulouse, France, 2012, p. 14–27, Springer-Verlag.

[2] R. Alenda, N. Olivetti and G.L. Pozzato. Nested Sequents Calculi for Normal Conditional Logics, *Journal of Logic and Computation*, first published online, **2013** (2013), 1–48.

[3] A. Baltag and S. Smets. The logic of conditional doxastic actions, *Texts in Logic and Games, Special Issue on New Perspectives on Games and Interaction* **4** (2008), 9–31.

[4] B. Beckert and R. Goré. Free variable tableaux for prepositional modal logics. In *Proceedings of the International Conference on Theorem Proving with Analytic Tableaux and Related Methods*, volume 1227 of LNCS, Pont-a-Mousson, France, 1997, p. 91–106, Springer.

[5] B. Beckert and J. Posegga. leantap: Lean tableau-based deduction, *Journal of Automated Reasoning* **15**(3) (1995), 339–358.

[6] C. Benzmüller, D.M. Gabbay, V. Genovese and D. Rispoli. Embedding and automating conditional logics in classical higherorder logic, *Ann Math Artif Intell* **66**(1-4) (2012), 257–271.

[7] O. Board. Dynamic interactive epistemology, *Games and Economic Behavior* **49**(1) (2004), 49–80.

[8] C. Boutilier. Conditional logics of normality: a modal approach, *Artificial Intelligence* **68**(1) (1994), 87–154.

[9] K. Brünnler and T. Studer. Syntactic cut-elimination for common knowledge, *Annals of Pure and Applied Logic* **160**(1) (2009), 82–95.

[10] G. Crocco and P. Lamarre. On the connection between nonmonotonic inference systems and conditional logics. In B. Nebel and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference KR'92*, 1992, p. 565–571.

[11] James P. and Delgrande. A first-order conditional logic for prototypical properties, *Artificial Intelligence* **33**(1) (1987), 105–130.

[12] M. Fitting. Prefixed tableaus and nested sequents, *Annals of Pure Applied Logic* **163**(3) (2012), 291–313.

[13] D.M. Gabbay, L. Giordano, A. Martelli, N. Olivetti and M.L. Sapino. Conditional reasoning in logic programming, *Journal of Logic Programming* **44**(1-3) (2000), 37–74.

[14] P. Gardenförs. *Knowledge in Flux*. MIT Press, 1988.

[15] V. Genovese, L. Giordano, V. Gliozzi and G.L. Pozzato. Logics in access control: A conditional approach, *Journal of Logic and Computation* **24**(4) (2012), 705–762.

[16] M.L. Ginsberg. Counterfactuals, *Artificial Intelligence* **30**(1) (1986), 35–79.

[17] L. Giordano, V. Gliozzi and N. Olivetti. Iterated belief revision and conditional logic, *Studia Logica* **70**(1) (2002), 23–47.

[18] L. Giordano, V. Gliozzi and N. Olivetti. Weak AGM postulates and strong ramsey test: A logical formalization, *Artificial Intelligence* **168**(1-2) (2005), 1–37.

[19] L. Giordano, V. Gliozzi and G.L. Pozzato. KLMLean 2.0: A theorem prover for klm logics of nonmonotonic reasoning. In Nicola Olivetti, editor, *Proceedings of TABLEAUX 2007 (Automated Reasoning with Analytic Tableaux and Related Methods)*, volume 4548 of *Lecture Notes in Computer Intellingence*, 2007, p. 238–244. Springer.

[20] L. Giordano and C. Schwind. Conditional logic of actions and causation, *Artificial Intelligence* **157**(1-2) (2004), 239–279.

[21] R. Goré, L. Postniece and A. Tiu. Cut-elimination and proof search for bi-intuitionistic logic using nested sequents. In Carlos Areces and Robert Goldblatt, editors, *Advances in Modal Logic*, volume 7, College Publications, 2008. http://www.aiml.net/volumes/volume7/Gore-Postniece-Tiu.pdf p. 43–66.

[22] G. Grahne. Updates and counterfactuals, *Journal of Logic and Computation* **8**(1) (1998), 87–117.

[23] D. Hausmann and L. Schröder. Optimizing Conditional Logic Reasoning within CoLoSS, *Electronic Notes in Theoretical Computer Science* **262** (2010), 157–171.
[24] R. Kashima. Cut-free sequent calculi for some tense logics, *Studia Logica* **53**(1) (1994), 119–136.
[25] H. Katsuno and A.O. Mendelzon. On the difference between updating a knowledge base and revising it. In J.F. Allen, R. Fikes and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Morgan Kaufmann, 1991, p. 387–394.
[26] S. Kraus, D. Lehmann and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics, *Artificial Intelligence* **44**(1-2) (1990), 167–207.
[27] D. Lewis. Counterfactuals. *Basil Blackwell Ltd*, 1973.
[28] D. Nute. Topics in conditional logic. *Reidel, Dordrecht*, 1980.
[29] N. Obeid. Model-based diagnosis and conditional logic, *Applied Intelligence* **14** (2001), 213–230.
[30] N. Olivetti and G.L. Pozzato. CondLean 3.0: Improving Condlean for Stronger Conditional Logics. In Bernhard Beckert, editor, *Proceedings of TABLEAUX 2005 (Automated Reasoning with Analytic Tableaux and Related Methods)*, volume 3702 of LNAI, Koblenz, Germany, 2005, p. 328–332. Springer-Verlag.
[31] N. Olivetti and G.L. Pozzato. Theorem Proving for Conditional Logics: CondLean and GoalDuck, *Journal of Applied Non-Classical Logics (JANCL)* **18**(4) (2008), 427–473.
[32] N. Olivetti and G.L. Pozzato. Nescond: An implementation of nested sequent calculi for conditional logics. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Proceedings of IJCAR (International Joint Conference on Automated Reasoning)*, volume 8562 of *Lecture Notes in Artificial Intellingence*,Springer, 2014, p. 511–518.
[33] C.B. Schwind. Causality in action theories, *Electronic Transactions on Artificial Intelligence (ETAI)* **3**(A) (1999), 27–50.