NESCOND: an Implementation of Nested Sequent Calculi for Conditional Logics

NESCOND: An Implementation of Nested Sequent Calculi for Conditional Logics

Nicola Olivetti[1] and Gian Luca Pozzato[2]

[1] Aix-Marseille Université, CNRS, LSIS UMR 7296 - France
nicola.olivetti@univ-amu.fr
[2] Dipartimento di Informatica - Universitá di Torino - Italy
gianluca.pozzato@unito.it

Abstract. We present NESCOND, a theorem prover for normal conditional logics. NESCOND implements some recently introduced NESted sequent calculi for propositional CONDitional logics CK and some of its significant extensions with axioms ID, MP and CEM. It also deals with the *flat* fragment of CK+CSO+ID, which corresponds to the logic **C** introduced by Kraus, Lehmann and Magidor. NESCOND is inspired by the methodology of leanT^AP and it is implemented in Prolog. The paper shows some experimental results, witnessing that the performances of NESCOND are promising. The program NESCOND, as well as all the Prolog source files, are available at `http://www.di.unito.it/%7Epozzato/nescond/`.

1 Introduction

Conditional logics are extensions of classical logic by a *conditional* operator $\Rightarrow$. They have a long history [10, 11], and recently they have found an interest in several fields of AI and knowledge representation. Just to mention a few (see [2] for a complete bibliography), they have been used to reason about prototypical properties, to model belief change, to reason about access control policies, to formalize epistemic change in a multi-agent setting. Conditional logics can also provide an axiomatic foundation of nonmonotonic reasoning [9]: here a conditional $A \Rightarrow B$ is read "normally, if A then B".

In previous works [1, 2] we have introduced *nested sequent calculi*, called $\mathcal{NS}$, for propositional conditional logics. Nested sequent calculi [4–6, 8] are a natural generalization of ordinary sequent calculi where sequents are allowed to occur within sequents. However, 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. The basic normal conditional logic CK and its extensions with ID, MP and CEM are considered, as well as the cumulative logic **C** introduced in [9] which corresponds to the *flat* fragment (i.e., without nested conditionals) of the conditional logic CK+CSO+ID.

Here we describe an implementation of $\mathcal{NS}$ in Prolog. The program, called NESCOND, gives a PSPACE decision procedure for the respective logics, and it is inspired by the methodology of leanT^AP [3]. The idea is that each axiom or rule of

S. Demri, D. Kapur, and C. Weidenbach (Eds.): IJCAR 2014, LNAI 8562, pp. 511–518, 2014.

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 provide experimental results by comparing NESCOND with CondLean [12] and GOALD$\mathcal{U}$CK [13]. Performances of NESCOND are promising, and show that nested sequent calculi are not only a proof theoretical tool, but they can be the basis of efficient theorem proving for conditional logics.

2 Conditional Logics and Their Nested Sequent Calculi

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 of ATM are *atomic formulas*; if A and B are formulas, then $\neg A$ and $A \circ B$ are *complex formulas*, where $\circ \in \{\wedge, \vee, \rightarrow, \Rightarrow\}$. 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*, and $[\ ]$ is the* evaluation function*, which assigns to an atom $P \in ATM$ the set of worlds where P is true, and is extended to boolean formulas as follows: $[\top] = \mathcal{W}$; $[\bot] = \emptyset$; $[\neg A] = \mathcal{W} - [A]$; $[A \wedge B] = [A] \cap [B]$; $[A \vee B] = [A] \cup [B]$; $[A \rightarrow B] = [B] \cup (\mathcal{W} - [A])$; $[A \Rightarrow B] = \{w \in \mathcal{W} \mid f(w, [A]) \subseteq [B]\}$. 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 $\mathcal{M}$.*

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 following standard extensions of the basic system CK:

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	$(A \Rightarrow B) \wedge (B \Rightarrow A) \rightarrow ((A \Rightarrow C) \rightarrow (B \Rightarrow C))$	$f(w, [A]) \sqsubseteq [B]$ and $f(w, [B]) \sqsubseteq [A]$ implies $f(w, [A]) = f(w, [B])$

In Figure 1 we present nested sequent calculi $\mathcal{N}S$, where S is an abbreviation for CK{+X}, and X={CEM, ID, MP, ID+MP, CEM+ID}. 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

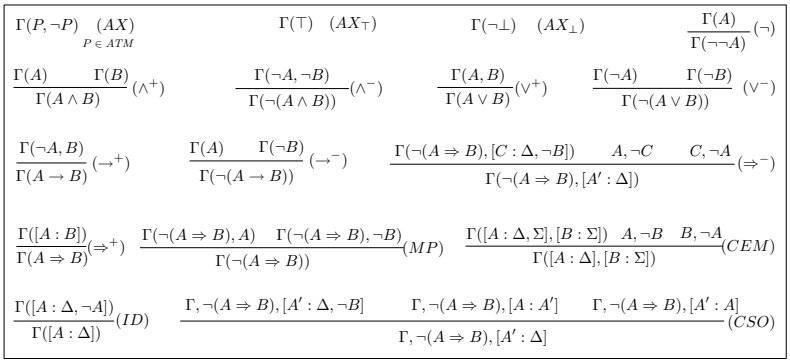

Fig. 1. The nested sequent calculi $\mathcal{NS}$.

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

where $n, m \geq 0$, $A_1, \ldots, A_m, B_1, \ldots, B_n$ are formulas and Γ_1, ..., Γ_n are nested sequents. A nested sequent can be directly interpreted as a formula by replacing "," by $\vee$ and ":" by $\Rightarrow$, i.e. the interpretation of $A_1, \ldots, A_m, [B_1 : \Gamma_1], \ldots, [B_n : \Gamma_n]$ is inductively defined by the formula $\mathcal{F}(\Gamma) = A_1 \vee \ldots \vee A_m \vee (B_1 \Rightarrow \mathcal{F}(\Gamma_1)) \vee \ldots \vee (B_n \Rightarrow \mathcal{F}(\Gamma_n))$.

We have also provided nested sequent calculi for the flat fragment, i.e. without nested conditionals, of CK+CSO+ID, corresponding to KLM logic **C** [9]. The rules of the calculus, called $\mathcal{N}\mathbf{C}_{\text{KLM}}$, are those ones of $\mathcal{N}$CK+ID (restricted to the flat fragment) where the rule $(\Rightarrow^-)$ is replaced by the rule (CSO).

In order to present 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 sequent. We use the symbol $(\ )$ to denote the empty context. A context is defined inductively as follows: $\Gamma(\ ) = \Delta, (\ )$ is a context; if $\Sigma(\ )$ is a context $\Gamma(\ ) = \Delta, [A : \Sigma(\ )]$ is a context. 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: if $\Gamma(\ ) = \Lambda, (\ )$ then $\Gamma(\Delta) = \Lambda, \Delta$; if $\Gamma(\ ) = \Lambda, [A : \Sigma(\ )]$ then $\Gamma(\Delta) = \Lambda, [A : \Sigma(\Delta)]$. The notions of derivation and of derivable sequent are defined as usual. In [1] we have shown that:

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 $\mathcal{NS}$.*

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). We obtain a sound, complete and terminating calculus if we restrict the applications of these rules as follows [1, 2]: $(\Rightarrow^-)$ can be applied only once to each formula $\neg(A \Rightarrow B)$ with a context $[C : \Delta]$ in each branch - the same for (CSO) in the system CK+CSO+ID; (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. For systems with (CEM), we need

a more complicated mechanism: due to space limitations, we refer to [1] for this case. These results give a PSPACE decision procedure for their respective logics.

3 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, even if it does not follow its style in a rigorous manner. The program comprises a set of clauses, each one of them 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, whose elements can be either formulas `F` or pairs `[Context,AppliedConditionals]` where:

- `Context` is also a pair of the form `[F,Gamma]`, where `F` is a formula of $\mathcal{L}$ and `Gamma` is a Prolog list representing a nested sequent;
- `AppliedConditionals` is a Prolog list `[A_1=>B_1,A_2=>B_2,...,A_k=>B_k]`, keeping track of the negated conditionals to which the rule $(\Rightarrow^-)$ has been already applied by using `Context` in the current branch. This is used in order to implement the restriction on the application of the rule $(\Rightarrow^-)$ in order to ensure termination.

Symbols $\top$ and $\bot$ are represented by constants `true` and `false`, respectively, whereas 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 $[P : Q \vee \neg P, [P : P \Rightarrow R], \neg R]$.

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 either (i) the nested sequent `NS` representing a nested sequent Γ contains all the fomulas in the list `Formulas` or (ii) there exists a context `[[A,Delta],AppliedConditionals]` in `NS` such that `deepMember(Formulas,Delta)` succeeds, that is to say there is a nested sequent occurring in `NS` containing all the formulas of `Formulas`.
- `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,-DefNS)` replaces `hole` in `NewNS` with the formulas in the list `Formulas`. `DefNS` is the output term matching the result.

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

implementing (AX), $(AX_{\top})$ and $(AX_{\bot})$, respectively. 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 `AppliedConditionals` 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(+NewNS,+Formulas,-DefNS)` is invoked to replace the `hole` in `NewNS`, 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 `AppliedConditionals` is updated by adding the formula $\neg(A \Rightarrow B)$ itself.

NESCOND for Extensions of CK. The implementation of the calculi for extensions of CK with axioms ID and MP are very similar. For systems allowing ID, 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 `DefNS`, and the flag is set to `false` in order to avoid multiple applications in a backward proof search.

The restriction on the application of the rule (MP) is implemented by equipping the predicate `prove` by a third argument, `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).

The implementation of the calculus for the flat fragment of CK+CSO+ID, corresponding to KLM cumulative logic **C**, is similar to that for CK+ID; the only difference is that $(\Rightarrow^{-})$ 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, such principal formulas are directly selected from the current sequent by easy membership tests (standard Prolog predicates `member` and `select`), without searching inside other contexts. Due tu space limitations, we omit details for extensions with CEM.

4 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. We have compared the performances of NESCOND with the ones of two other provers for conditional logics: CondLean 3.2, implementing labelled sequent calculi [12], and the goal-directed procedure GOALD$\mathcal{U}$CK [13]. We have tested the three provers (i) on randomly generated sequents, obtaining the results shown in Figure 2 and (ii) over a set of valid formulas. Concerning CK, we have considered 88 valid formulas obtained by translating K valid formulas ($\Box A$ is replaced by $\top \Rightarrow A$, whereas $\Diamond A$ is replaced by $\neg(\top \Rightarrow \neg A)$) provided by Heuerding, obtaining the results in Figure 3.

Concerning (i), we have tested the three provers over 2000 random sequents, whose formulas are built from 15 different atomic variables and have a high level of nesting (10): NESCOND is not able to answer only in 0.05% of cases (1 sequent over 2000) within 10 seconds, whereas both GOALD$\mathcal{U}$CK 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, NESCOND answers in 100% of cases, whereas its two competitors have still more than 1.30% of timeouts. The difference is much more significant when considering sequents with a lower level of nesting (3) and whose formulas contain only 3 different atomic variables: with a time limit of 10 seconds, NESCOND is not able to answer only in 9.09% of cases, whereas both CondLean and GOALD$\mathcal{U}$CK are not able to conclude in 16.55% and in 51.15% of cases, respectively: this is 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$\mathcal{U}$CK) within the fixed time limit.

Concerning (ii) the performances of NESCOND are also encouraging. 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$\mathcal{U}$CK; 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 Figure 3: here we have not included GOALD$\mathcal{U}$CK since the most formulas adopted do not belong to the fragments admitting goal-directed proofs [13].

These results show that the performances of NESCOND are encouraging, probably better than the ones of the other existing provers for conditional logics (notice that there

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%

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%	24,65%	0,00%

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%	16,55%
NESCOND	73,2%	17,75%	9,05%

Fig. 2. NESCOND vs CondLean vs GOALD$\mathcal{U}$CK over 2000 random sequents

Timeouts for CK

	100ms	1s	10s	1m
CondLean	12,50%	12,50%	9,09%	9,09%
GoalDUCK	21,71%	18,26%	13,88%	13,88%
NESCOND	12,50%	10,23%	5,68%	4,55%

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. 3. NESCOND vs CondLean vs GOALD$\mathcal{U}$CK: timeouts over valid formulas

is no set of acknowledged benchmarks for them). Figure 3 shows that 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 Conclusions and Future Issues

We have presented NESCOND, a theorem prover for conditional logics implementing nested sequent calculi introduced in [1]. Statistics in section 4 show that nested sequent calculi do not only provide elegant and natural calculi for conditional logics, but they are also significant for developing efficient theorem provers for them. In future research we aim to extend 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 of CK generated by axioms (ID), (MP), (CEM) and (CS). This will be object of subsequent research. We also aim at comparing the performances of NESCOND with those of CoLoSS [7], a generic-purpose theorem prover for coalgebraic modal logics which can handle also basic conditional logics.

References

1. Alenda, R., Olivetti, N., Pozzato, G.L.: Nested Sequent Calculi for Conditional Logics. In: del Cerro, L.F., Herzig, A., Mengin, J. (eds.) JELIA 2012. LNCS, vol. 7519, pp. 14–27. Springer, Heidelberg (2012)
2. Alenda, R., Olivetti, N., Pozzato, G.L.: Nested Sequents Calculi for Normal Conditional Logics. Journal of Logic and Computation (to appear)
3. Beckert, B., Posegga, J.: leantap: Lean tableau-based deduction. JAR 15(3), 339–358 (1995)
4. Brünnler, K., Studer, T.: Syntactic cut-elimination for common knowledge. Annals of Pure and Applied Logic 160(1), 82–95 (2009)
5. Fitting, M.: Prefixed tableaus and nested sequents. A. Pure App. Log. 163(3), 291–313 (2012)
6. Goré, R., Postniece, L., Tiu, A.: Cut-elimination and proof-search for bi-intuitionistic logic using nested sequents. In: Advances in Modal Logic, vol. 7, pp. 43–66 (2008)
7. Hausmann, D., Schröder, L.: Optimizing Conditional Logic Reasoning within CoLoSS. Electronic Notes in Theoretical Computer Science 262, 157–171 (2010)
8. Kashima, R.: Cut-free sequent calculi for some tense logics. St. Logica 53(1), 119–136 (1994)
9. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
10. Lewis, D.: Counterfactuals. Basil Blackwell Ltd. (1973)
11. Nute, D.: Topics in conditional logic. Reidel, Dordrecht (1980)
12. Olivetti, N., Pozzato, G.L.: CondLean 3.0: Improving Condlean for Stronger Conditional Logics. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 328–332. Springer, Heidelberg (2005)
13. Olivetti, N., Pozzato, G.L.: Theorem Proving for Conditional Logics: CondLean and GoalDuck. Journal of Applied Non-Classical Logics (JANCL) 18(4), 427–473 (2008)