VINTE: an Implementation of Internal Calculi for Lewis' Logics of Counterfactual Reasoning

Marianna Girlando, Bjoern Lellmann, Nicola Olivetti, Gian Luca Pozzato, Quentin Vitalis

VINTE: an Implementation of Internal Calculi for Lewis' Logics of Counterfactual Reasoning*

Marianna Girlando[1], Björn Lellmann[2], Nicola Olivetti[1], Gian Luca Pozzato[3], and Quentin Vitalis[4]

[1] Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, 13397, Marseille, France - {marianna.girlando,nicola.olivetti}@univ-amu.fr
[2] Technische Universität Wien, Austria - lellmann@logic.at
[3] Dipartimento di Informatica, Universitá di Torino, Italy - pozzato@di.unito.it
[4] École Spéciale Militaire de Saint-Cyr - Département Informatique - quentin.vitalis@protonmail.com

Abstract. We present VINTE, a theorem prover for conditional logics for counterfactual reasoning introduced by Lewis in the seventies. VINTE implements some internal calculi recently introduced for the basic system $\mathbb{V}$ and some of its significant extensions with axioms $\mathbb{N}$, $\mathbb{T}$, $\mathbb{C}$, $\mathbb{W}$ and $\mathbb{A}$. VINTE 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 VINTE are promising.

1 Introduction

Conditional logics are extensions of classical logic by a *conditional* operator $\Box\!\!\rightarrow$. They have a long history [11], and recently they have found an interest in several fields of AI and knowledge representation. Just to mention a few (see [1] for a complete bibliography), they have been used to reason about prototypical properties, to model belief change [8], to reason about access control policies [5], to formalize epistemic change in a multi-agent setting [2, 4]. Conditional logics can also provide an axiomatic foundation of nonmonotonic reasoning [9]: here a conditional $A \Box\!\!\rightarrow B$ is read "normally, if A then B".

In early seventies, Lewis proposed a formalization of conditional logics in order to represent a kind of hypothetical reasoning that cannot be captured by the material implication of classical logic [10]. His original motivation was to formalize counterfactuals, that is to say, conditionals of the form "if A were the case then B would be the case", where A is false. The family of logics studied by Lewis is semantically characterized by sphere models, a particular kind of neighbourhood models introduced by Lewis himself. In Lewis' terminology, a *sphere* denotes a set of worlds; in sphere models, each world is equipped with

* Supported by the Project TICAMORE ANR-16-CE91-0002-01, by the EU under Marie Skłodowska-Curie Grant Agreement No. [660047], and by the Project "ExceptionOWL", Università di Torino and Compagnia di San Paolo, call 2014.

a nested system of such spheres. From the viewpoint of the given world, inner sets represent the "most plausible worlds", while worlds belonging only to outer sets are considered as less plausible. In order to treat the conditional operator, Lewis takes as primitive the comparative plausibility connective $\preccurlyeq$: a formula $A \preccurlyeq B$ means "A is at least as plausible as B". The conditional $A \boxright B$ can be then defined as "A is impossible" or "$A \wedge \neg B$ is less plausible than $A \wedge B$". However, the latter assertion is equivalent to the simpler one "$A \wedge \neg B$ is less plausible than A"[5].

In previous works [16, 6] we have introduced internal, standard, cut-free calculi for most logics of the Lewis family, namely logics $\mathbb{V}$, $\mathbb{VN}$, $\mathbb{VT}$, $\mathbb{VW}$, $\mathbb{VC}$, $\mathbb{VA}$ and $\mathbb{VNA}$. Here we describe a Prolog implementation of the invertible calculi $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$ introduced in [6]. The program, called VINTE, gives a decision procedure for the respective logics, and it is inspired by the methodology of $\mathsf{lean}T^{A}P$ [3]. The idea is that each axiom or rule of the sequent calculi is implemented by a Prolog clause of the program. The resulting code is therefore simple and compact: the implementation of VINTE for $\mathbb{V}$ consists of only 3 predicates, 21 clauses and 57 lines of code. We provide experimental results by comparing VINTE with the following theorem provers for conditional logics: CondLean [12], GoalD$\mathcal{U}$CK [13] and NESCOND [14, 15], and we show that the performances of VINTE are quite promising. The program VINTE, as well as all the Prolog source files, are available for free usage and download at http://193.51.60.97:8000/vinte/.

2 Lewis' Conditional Logics

We consider the *conditional logics* defined by Lewis in [10]. The set of *conditional formulae* is given by $\mathcal{F} ::= p \mid \bot \mid \mathcal{F} \to \mathcal{F} \mid \mathcal{F} \preccurlyeq \mathcal{F}$, where $p \in \mathcal{V}$ is a propositional variable. The other boolean connectives are defined in terms of $\bot, \to$ as usual. Intuitively, a formula $A \preccurlyeq B$ is interpreted as "A is at least as plausible as B".

As mentioned above, Lewis' counterfactual implication $\boxright$ can be defined in terms of comparative plausibility $\preccurlyeq$ as

$$A \boxright B \;\equiv\; (\bot \preccurlyeq A) \vee \neg((A \wedge \neg B) \preccurlyeq A).$$

The semantics of this logic is defined by Lewis in terms of *sphere semantics*:

Definition 1. *A* sphere model *(or* model*) is a triple* $\langle W, \mathsf{SP}, [\![.]\!] \rangle$*, consisting of a non-empty set* W *of elements, called* worlds*, a mapping* $\mathsf{SP} : W \to \mathcal{P}(\mathcal{P}(W))$*, and a propositional valuation* $[\![.]\!] : \mathcal{V} \to \mathcal{P}(W)$*. Elements of* $\mathsf{SP}(x)$ *are called* spheres*. We assume the following conditions: for every* $\alpha \in \mathsf{SP}(w)$ *we have* $\alpha \neq \emptyset$*, and for every* $\alpha, \beta \in \mathsf{SP}(w)$ *we have* $\alpha \subseteq \beta$ *or* $\beta \subseteq \alpha$*. The latter condition is called* sphere nesting*.*

The valuation $[\![.]\!]$ *is extended to all formulae by:* $[\![\bot]\!] = \emptyset$*;* $[\![A \to B]\!] = (W - [\![A]\!]) \cup [\![B]\!]$*;* $[\![A \preccurlyeq B]\!] = \{w \in W \mid$ *for all* $\alpha \in \mathsf{SP}(w)$*. if* $[\![B]\!] \cap \alpha \neq \emptyset$*, then* $[\![A]\!] \cap \alpha \neq \emptyset\}$*. For* $w \in W$ *we also write* $w \Vdash A$ *instead of* $w \in [\![A]\!]$*. As for spheres,*

[5] It is worth noticing that in turn the connective $\preccurlyeq$ can be defined in terms of $\boxright$.

$\mathsf{CPR}\ \dfrac{\vdash B \to A}{\vdash A \preccurlyeq B}$	$\mathsf{CPA}\ (A \preccurlyeq A \vee B) \vee (B \preccurlyeq A \vee B)$
$\mathsf{TR}\ (A \preccurlyeq B) \wedge (B \preccurlyeq C) \to (A \preccurlyeq C)$	$\mathsf{CO}\ (A \preccurlyeq B) \vee (B \preccurlyeq A)$
$\mathsf{N}\ \neg(\bot \preccurlyeq \top)$	$\mathsf{W}\ A \to (A \preccurlyeq \top)$
$\mathsf{T}\ (\bot \preccurlyeq \neg A) \to A$	$\mathsf{A1}\ (A \preccurlyeq B) \to \big(\bot \preccurlyeq \neg(A \preccurlyeq B)\big)$
$\mathsf{C}\ (A \preccurlyeq \top) \to A$	$\mathsf{A2}\ \neg(A \preccurlyeq B) \to \big(\bot \preccurlyeq (A \preccurlyeq B)\big)$

$$\mathcal{A}_{\mathbb{V}} := \{\mathsf{CPR}, \mathsf{CPA}, \mathsf{TR}, \mathsf{CO}\}$$

$$\mathcal{A}_{\mathbb{VN}} := \mathcal{A}_{\mathbb{V}} \cup \{\mathsf{N}\} \quad \mathcal{A}_{\mathbb{VT}} := \mathcal{A}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{T}\} \quad \mathcal{A}_{\mathbb{VW}} := \mathcal{A}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{T}, \mathsf{W}\}$$
$$\mathcal{A}_{\mathbb{VC}} := \mathcal{A}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{T}, \mathsf{W}, \mathsf{C}\} \quad \mathcal{A}_{\mathbb{VA}} := \mathcal{A}_{\mathbb{V}} \cup \{\mathsf{A1}, \mathsf{A2}\} \quad \mathcal{A}_{\mathbb{VNA}} := \mathcal{A}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{A1}, \mathsf{A2}\}$$

Table 1. Lewis' logics and axioms.

we write $\alpha \Vdash^{\forall} A$ *meaning* $\forall x \in \alpha.\ x \Vdash A$ *and* $\alpha \Vdash^{\exists} A$ *meaning* $\exists x \in \alpha.\ x \Vdash A$[6]. *Validity and satisfiability of formulae in a class of models are defined as usual.* Conditional logic $\mathbb{V}$ *is the set of formulae valid in all sphere models.*

Extensions of $\mathbb{V}$ are semantically given by specifying additional conditions on the class of sphere models, namely:

- *normality*: for all $w \in W$ we have $\mathsf{SP}(w) \neq \emptyset$;
- *total reflexivity*: for all $w \in W$ we have $w \in \bigcup \mathsf{SP}(w)$;
- *weak centering*: normality holds and for all $\alpha \in \mathsf{SP}(w)$ we have $w \in \alpha$;
- *centering*: for all $w \in W$ we have $\{w\} \in \mathsf{SP}(w)$;
- *absoluteness*: for all $w, v \in W$ we have $\mathsf{SP}(w) = \mathsf{SP}(v)$[7].

Extensions of $\mathbb{V}$ are denoted by concatenating the letters for these properties: $\mathbb{N}$ for normality, $\mathbb{T}$ for total reflexivity, $\mathbb{W}$ for weak centering, $\mathbb{C}$ for centering, and $\mathbb{A}$ for absoluteness. All the above logics can be characterized by axioms in a Hilbert-style system [10, Chp. 6]. The modal axioms formulated in the language with only the comparative plausibility operator are presented in Table 1 (where $\vee$ and $\wedge$ bind stronger than $\preccurlyeq$). The propositional axioms and rules are the standard ones.

3 Internal Calculi for Conditional Logics

Table 2 presents calculi $\mathcal{I}^{!}_{\mathcal{L}}$, where $\mathcal{L}$ ranges over the logics $\mathbb{V}, \mathbb{VN}, \mathbb{VT}, \mathbb{VW}, \mathbb{VC}, \mathbb{VA}, \mathbb{VNA}$, introduced in [6]. The basic constituent of sequents are *blocks* of the form $[A_1, \ldots, A_m \lhd A]$, with $A_1, \ldots, A_m, A$ formulas, representing disjunctions of $\preccurlyeq$-formulas. A *sequent* is a tuple $\Gamma \Rightarrow \Delta$, where Γ is a multiset of conditional formulae, and Δ is a multiset of conditional formulae and blocks. The *formula interpretation* of a sequent is given by:

$$\iota(\Gamma \Rightarrow \Delta', [\Sigma_1 \lhd A_1], \ldots, [\Sigma_n \lhd A_n]) := \bigwedge \Gamma \to \bigvee \Delta' \vee \bigvee_{1 \leq i \leq n} \bigvee_{B \in \Sigma_i} (B \preccurlyeq A_i)$$

[6] Employing this notation, satisfiability of a $\preccurlyeq$-formula in a model becomes the following: $x \Vdash A \preccurlyeq B$ iff for all $\alpha \in \mathsf{SP}(x)$. $\alpha \Vdash^{\forall} \neg B$ or $\alpha \Vdash^{\exists} A$.

[7] It is worth noticing that absoluteness can be equally stated as *local absoluteness*: $\forall w \in W, \forall v \in \bigcup \mathsf{SP}(w)$ it holds $\mathsf{SP}(w) = \mathsf{SP}(v)$.

$$\frac{}{\Gamma, \bot \Rightarrow \Delta}\ \bot_L \qquad \frac{}{\Gamma, p \Rightarrow \Delta, p}\ \mathsf{init} \qquad \frac{\Gamma, B \Rightarrow \Delta \quad \Gamma \Rightarrow \Delta, A}{\Gamma, A \to B \Rightarrow \Delta}\ \to_L \qquad \frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B}\ \to_R$$

$$\frac{\Gamma \Rightarrow \Delta, [A \lhd B]}{\Gamma \Rightarrow \Delta, A \preccurlyeq B}\ \preccurlyeq_R \qquad \frac{\Gamma, A \preccurlyeq B \Rightarrow \Delta, [B, \Sigma \lhd C] \quad \Gamma, A \preccurlyeq B \Rightarrow \Delta, [\Sigma \lhd A], [\Sigma \lhd C]}{\Gamma, A \preccurlyeq B \Rightarrow \Delta, [\Sigma \lhd C]}\ \preccurlyeq^{\mathsf{i}}_L$$

$$\frac{}{\Gamma \Rightarrow \Delta, \top}\ \top_R \qquad \frac{\Gamma \Rightarrow \Delta, [\Sigma_1, \Sigma_2 \lhd A], [\Sigma_2 \lhd B] \quad \Gamma \Rightarrow \Delta, [\Sigma_1 \lhd A], [\Sigma_1, \Sigma_2 \lhd B]}{\Gamma \Rightarrow \Delta, [\Sigma_1 \lhd A], [\Sigma_2 \lhd B]}\ \mathsf{com}^{\mathsf{i}}$$

$$\frac{\bot \preccurlyeq A, \Gamma \Rightarrow \Delta \quad \Gamma \Rightarrow \Delta, [A \wedge \neg B \lhd A]}{A \boxdot\!\!\to B, \Gamma \Rightarrow \Delta}\ \boxdot\!\!\to_L \qquad \frac{(A \preccurlyeq \neg B) \preccurlyeq A, \Gamma \Rightarrow \Delta, [\bot \lhd A]}{\Gamma \Rightarrow \Delta, A \boxdot\!\!\to B}\ \boxdot\!\!\to_R$$

$$\frac{A \Rightarrow \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ \mathsf{jump} \qquad \frac{\Gamma \Rightarrow \Delta, [\bot \lhd \top]}{\Gamma \Rightarrow \Delta}\ \mathsf{N}$$

$$\frac{\Gamma, A \preccurlyeq B \Rightarrow \Delta, B \qquad \Gamma, A \preccurlyeq B \Rightarrow \Delta, [\bot \lhd A]}{\Gamma, A \preccurlyeq B \Rightarrow \Delta}\ \mathsf{T}^{\mathsf{i}} \qquad \frac{\Gamma \Rightarrow \Delta, [\Sigma \lhd A], \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd A]}\ \mathsf{W}^{\mathsf{i}}$$

$$\frac{\Gamma, A \preccurlyeq B \Rightarrow \Delta, B \quad \Gamma, A \preccurlyeq B, A \Rightarrow \Delta}{\Gamma, A \preccurlyeq B \Rightarrow \Delta}\ \mathsf{C}^{\mathsf{i}} \qquad \frac{\Gamma^{\preccurlyeq}, B \Rightarrow \Delta^{\preccurlyeq}, [\Sigma \lhd B], \Sigma}{\Gamma \Rightarrow \Delta, [\Sigma \lhd B]}\ \mathsf{A}^{\mathsf{i}}$$

Here $\Gamma^{\preccurlyeq} \Rightarrow \Delta^{\preccurlyeq}$ is $\Gamma \Rightarrow \Delta$ restricted to formulae of the form $C \preccurlyeq D$ and blocks.

$$\mathcal{I}^{\mathsf{i}}_{\mathbb{V}} := \{\bot_L, \top_R, \mathsf{init}, \to_L, \to_R, \boxdot\!\!\to_L, \boxdot\!\!\to_R, \preccurlyeq_R, \preccurlyeq^{\mathsf{i}}_L, \mathsf{com}^{\mathsf{i}}, \mathsf{jump}\}$$

$$\mathcal{I}^{\mathsf{i}}_{\mathbb{V}\mathsf{N}} := \mathcal{I}^{\mathsf{i}}_{\mathbb{V}} \cup \{\mathsf{N}\} \qquad \mathcal{I}^{\mathsf{i}}_{\mathbb{V}\mathsf{W}} := \mathcal{I}^{\mathsf{i}}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{T}^{\mathsf{i}}, \mathsf{W}^{\mathsf{i}}\} \qquad \mathcal{I}^{\mathsf{i}}_{\mathbb{V}\mathbb{A}} := \mathcal{I}^{\mathsf{i}}_{\mathbb{V}} \cup \{\mathsf{A}^{\mathsf{i}}\}$$
$$\mathcal{I}^{\mathsf{i}}_{\mathbb{V}\mathsf{T}} := \mathcal{I}^{\mathsf{i}}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{T}^{\mathsf{i}}\} \quad \mathcal{I}^{\mathsf{i}}_{\mathbb{V}\mathbb{C}} := \mathcal{I}^{\mathsf{i}}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{T}^{\mathsf{i}}, \mathsf{W}^{\mathsf{i}}, \mathsf{C}^{\mathsf{i}}\} \quad \mathcal{I}^{\mathsf{i}}_{\mathbb{V}\mathsf{N}\mathbb{A}} := \mathcal{I}^{\mathsf{i}}_{\mathbb{V}} \cup \{\mathsf{N}, \mathsf{A}^{\mathsf{i}}\}$$

Table 2. The calculus $\mathcal{I}^{\mathsf{i}}_{\mathbb{V}}$ and its extensions.

As usual, given a formula $G \in \mathcal{L}$, in order to check whether G is valid we look for a derivation of $\Rightarrow G$. Given a sequent $\Gamma \Rightarrow \Delta$, we say that it is derivable if it admits a *derivation*, namely a tree whose root is $\Gamma \Rightarrow \Delta$, every leaf is an instance of init or $\bot_L$ or $\top_R$, and every non-leaf node is (an instance of) the conclusion of a rule having (an instance of) the premises of the rule as children.

In [6] it is shown that:

Theorem 2. *The calculi $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$ are sound and complete for the respective logics.*

In [6] it has been also shown that the calculi $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$ can be used in a decision procedure for the logic $\mathcal{L}$ as follows. Since contractions and weakenings are admissible we may assume that a derivation of a duplication-free sequent (containing duplicates neither of formulae nor of blocks) only contains duplication-free sequents: whenever a (backwards) application of a rule introduces a duplicate of a formula already in the sequent, it is immediately deleted in the next step using a backwards application of weakening. While officially our calculi do not contain the weakening rules, the proof of admissibility of weakening yields a procedure to transform a derivation with these rules into one without. Since all rules have the subformula property, the number of duplication-free sequents possibly relevant to a derivation of a sequent is bounded in the number of subformulae of that sequent, and hence enumerating all possible loop-free derivations of the above form yields a decision procedure for the logic.

Theorem 3. *Proof search with the blocking technique above for a sequent $\Gamma \Rightarrow \Delta$ in calculus $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$ always comes to an end in a finite number of steps.*

As usual, in order to implement such a decision procedure, we have to control the application of the rules to avoid the introduction of duplicated sequents. Concerning the rule $\preccurlyeq^{\mathsf{i}}_L$, the principal formula $A \preccurlyeq B$ is copied into the premisses, then we have to avoid that, in a backward application of the rule, such formula is redundantly applied by using the same block $[\Sigma \lhd C]$. Since no rule, with the exception of jump, remove formulas from blocks, we allow a backward application of $\preccurlyeq^{\mathsf{i}}_L$ to a sequent $\Gamma, A \preccurlyeq B \Rightarrow \Delta, [\Sigma \lhd C]$ if neither $[B, \Sigma \lhd C]$ nor $[\Sigma' \lhd C]$, where $B, \Sigma \subset \Sigma'$ belong to Δ, and neither $[\Sigma \lhd A]$ nor $[\Sigma' \lhd A]$, where $\Sigma \subset \Sigma'$ belong to Δ. Similarly for the $\mathsf{com}^{\mathsf{i}}$ rule, which can be applied backward to blocks $[\Sigma_1 \lhd A]$ and $[\Sigma_2 \lhd B]$ if neither $[\Sigma_1, \Sigma_2 \lhd A]$ nor $[\Sigma_1, \Sigma_2 \lhd B]$ are introduced redundantly in the premisses. For rules like W^{i}, whose premisses contain the principal formula, we just need to check whether the formulas introduced in the premisses by a backward application of the rule already belong to such premisses or not. In the first case, the application of the rule is blocked. As an example, if W^{i} is applied to $\Rightarrow [A \vee B \lhd C]$, then the premiss is $\Rightarrow [A \vee B \lhd C], A \vee B$, that becomes $(*) \;\; \Rightarrow [A \vee B \lhd C], A, B$ after an application of the rule $\vee_R$. The rule W^{i} can be further applied to $(*)$, since $A \vee B$ does not belong to the right-hand side of the sequent, then obtaining the premiss $\Rightarrow [A \vee B \lhd C], A, B, A \vee B$, and at this point neither W^{i} nor $\vee_R$ can be further applied.

4 Design of VINTE

In this section we present a Prolog implementation of the internal calculi $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$ recalled in Section 3. The program, called VINTE ($\mathbb{V}$: INTernal calculi and Extensions), is inspired by the "lean" methodology of $\mathsf{lean}\, T^A P$, 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 $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism.

VINTE represents a sequent with a pair of Prolog lists `[Gamma,Delta]`, where `Gamma` and `Delta` represent the left-hand side and the right-hand side of the sequent, respectively. Elements of `Gamma` are formulas, whereas elements of `Delta` can be either formulas or pairs `[Sigma,A]`, where `Sigma` is a Prolog list, representing a block $[\Sigma \lhd A]$. Symbols $\top$ and $\bot$ are represented by constants `true` and `false`, respectively, whereas connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\preccurlyeq$, and $\Box\!\!\rightarrow$ are represented by `-`, `^`, `?`, `->`, `<`, and `=>`. Propositional variables are represented by Prolog atoms. As an example, the Prolog pair

```
[  [-(p?q), p, p -> q, p < r],  [q, p => (q^r), [ [true, p, q], r] ]  ]
```

is used to represent the sequent

$$\neg(P \vee Q), P, P \rightarrow Q, P \preccurlyeq R \Rightarrow Q, P \Box\!\!\rightarrow (Q \wedge R), [\top, P, Q \lhd R]\,.$$

The calculi $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$ are implemented by the predicate

```
prove([Gamma,Delta],ProofTree).
```

This predicate succeeds if and only if the sequent $\Gamma \Rightarrow \Delta$ represented by the pair of lists `[Gamma,Delta]` 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 $(A \preccurlyeq B) \vee (B \preccurlyeq A)$ is valid in $\mathbb{V}$, one queries VINTE with the goal: `prove([[],[(a<b)?(b<a)]],ProofTree).` Each clause of `prove` implements an axiom or rule of $\mathcal{I}^{\mathsf{i}}_{\mathcal{L}}$. To search a derivation of a sequent $\Gamma \Rightarrow \Delta$, VINTE proceeds as follows. First of all, if $\Gamma \Rightarrow \Delta$ is an instance of either $\bot_L$ or $\top_R$ or init, the goal will succeed immediately by using one of the following clauses for the axioms:

```
prove([Gamma,Delta],tree(axb):-member(false,Gamma),!.
prove([Gamma,Delta],tree(axt)):-member(true,Delta),!.
prove([Gamma,Delta],tree(init)):-member(P,Gamma),member(P,Delta),!.
```

If $\Gamma \Rightarrow \Delta$ is not an instance of the ending rules, then the first applicable rule will be chosen, e.g. if Δ contains a formula `A < B`, then the clause implementing the $\preccurlyeq_R$ rule will be chosen, and VINTE will be recursively invoked on the unique premise of such a rule. VINTE 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, with the exception of the rule jump which is the last rule to be applied. As an example, the clause implementing $\preccurlyeq^{\mathsf{i}}_L$ is as follows:

```
1. prove([Gamma,Delta],tree(precL,Sub1,Sub2)):-
2.    member(A < B,Gamma),
3.    select([Sigma,C],Delta,NewDelta),
4.    remove_duplicates([B|Sigma],NewSigma),
5.    \+memberOrdSet([NewSigma,C],Delta),
6.    \+memberOrdSet([Sigma,A],Delta), !,
7.    prove([Gamma,[[NewSigma,C]|NewDelta]],Sub1),
8.    prove([Gamma,[[Sigma,A]|Delta]],Sub2).
```

In line 4, the auxiliary predicate `remove_duplicates` is invoked in order to remove duplicated formulas in the multiset of formulas B, Σ. This is equivalent to apply weakening if the formula B already belongs to Γ. Another auxiliary predicate, `memberOrdSet`, is then invoked in lines 5 and 6 in order to implement the decision procedure described at the end of Section 3: Prolog ordsets are used in order to deal with the equivalence of lists where formulas occur in different orders. Since the rule is invertibile, Prolog cut `!` is used in line 6 to eventually block backtracking. The jump rule is implemented as follows:

```
1.  prove([Gamma,Delta],tree(jump,SubTree)):-
2.      member([Sigma,A],Delta),
3.      prove([[A],Sigma],SubTree).
```

This is the only non invertible rule, and a backtracking point is introduced by the choice of the block $[\Sigma \lhd A]$ in Δ to which apply the rule.

The implementation of the calculi for extensions of $\mathbb{V}$ is very similar. The only significant difference is in the more sophisticated mechanism needed to ensure

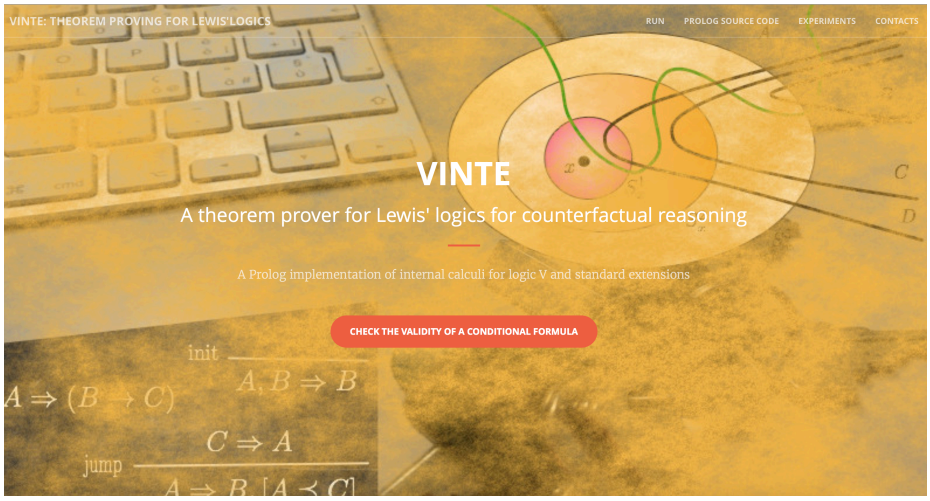

Fig. 1. Home page of VINTE.

termination. As an example, in system implementing the calculus for $\mathbb{VC}$, the predicate `prove` is equipped by a further parameter, called `AppliedC`, containing the list of formulas of the form $A \preccurlyeq B$ to which the rule C^{i} has been already applied in the current branch. The code implementing the rule C^{i} is as follows:

```
1. prove([Gamma,Delta],tree(c,Sub1,Sub2),AppliedC):-
2.  member(A < B,Gamma),
3.  \+member(A < B,AppliedC),
4.  (\+member(B,Delta);\+member(A,Gamma)), !,
5.  prove([Gamma,[B|Delta]],Sub1,[A<B|AppliedC]),
6.  prove([[A|Gamma],Delta],Sub2,[A<B|AppliedC]).
```

Line 3 shows how this parameter is used in order to avoid multiple application of C^{i} to the same formula $A \preccurlyeq B$ in a given branch, then the consequent loop in the proof search: if $A \preccurlyeq B$ belongs to `AppliedC`, then the rule C^{i} has been already applied to it in the current branch and it is no longer applied; otherwise, the predicate `prove` is recursively invoked on the premisses of the rule, and $A \preccurlyeq B$ is added to the list of formulas already employed for applications of C^{i}.

VINTE can be used by means of a simple web interface, implemented in php and allowing the user to check whether a conditional formula is valid by using his computer as well as his mobile device. The web interface also allows the user to choose the conditional system to adopt, namely $\mathbb{V}$ or one of the extensions mentioned in section 2. When a formula is valid, VINTE builds a pdf file showing a derivation in the invertible calculi recalled in section 3 as well as the LaTeX source file. Prolog source codes and experimental results are also available. Some pictures of VINTE are shown in Figures 3 and 4.

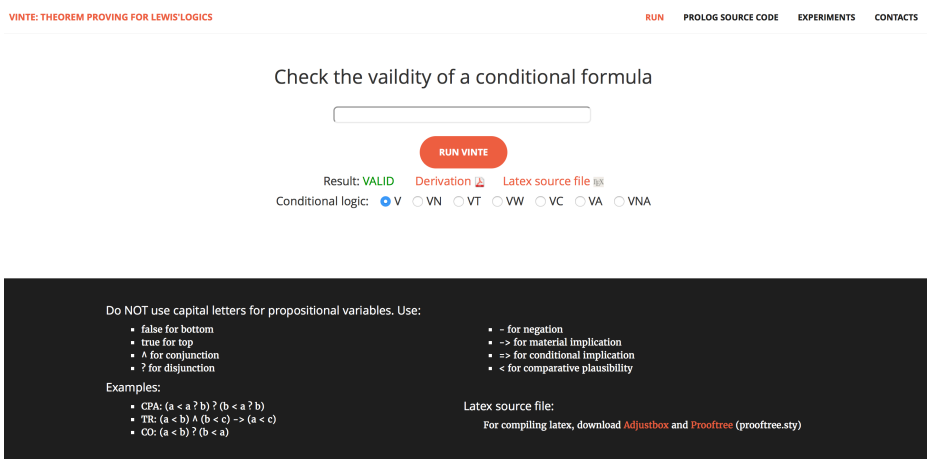

Fig. 2. When the user wants to check whether a formula F is valid, then (i) he selects the conditional logic to use, (ii) he types F in the form and (iii) clicks the button in order to execute the calculi presented in Section 3.

5 Performance of VINTE

The performance of VINTE seems to be 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. In absence of theorem provers specifically tailored for Lewis' logics, we have compared the performances of VINTE with those of the following theorem provers for conditional logics:

- CondLean 3.1, implementing labelled sequent calculi [12];
- GoalD$\mathcal{U}$CK, implemented a goal-directed proof procedure [13];
- NESCOND, implementing nested sequent calculi [14, 15].

All the above mentioned theorem provers take into account conditional logics based on the *selection function semantics* [11], namely conditional logic CK and extensions with axioms ID, MP, CEM, CSO, that are weaker than the ones considered by VINTE, then the experimental results are only partially significant and only aim at conjecturing that the performance of VINTE is promising.

We have performed two kinds of experiments: 1. we have tested the four provers on randomly generated formulas, fixing different time limits; 2. we have tested VINTE for system $\mathbb{VN}$ and NESCOND over a set of valid formulas in the logic CK, therefore also valid in $\mathbb{VN}$ [10].

Concerning 1, we have tested the four provers over 2000 random sequents with 20 formulas built from 7 different atomic variables and with a high level of nesting (10): both VINTE and NESCOND are able to answer in all cases within 1 second, whereas CondLean 3.1 is not able to conclude anything in 55 cases over 1000. Performance of GoalD$\mathcal{U}$CK is even worse, since it fails to answer in 174 cases. The differences seem much more significant when considering sequents with more formulas (100) and with a higher level of nesting (20): with a time

limit of 5 ms, GoalDUCK is faster than CondLean 3.1 and NESCOND, since it is not able to answer only in 136 cases over 1000, against 173 timeouts of CondLean 3.1 and 479 timeouts of NESCOND. VINTE is able to answer again in all cases, and only NESCOND is also able to complete all the tests, when the time limit is extended to 1 second. We have repeated the above experiments by considering implementations of VINTE for extensions of $\mathbb{V}$, obtaining the results summarized in Table 4.

As mentioned, since the four provers take into account different logics, in general they give a different answer over the same - randomly generated - sequent. Then, this kind of tests over CK formulas could be considered not very significant. Instead, we should test VINTE over a set of significant formulas for the specific Lewis' logics that it is designed for: to this aim, we are currently developing a set of benchmarks for VINTE drawn by valid instances of Lewis' axioms.

Seq. with 20 formulas (nesting lev.10)		
Prover	**Limit 5 ms**	**Limit 1 s**
VINTE	3	0
CondLean 3.1	55	14
GoalDUCK	249	174
NESCOND	35	0

Seq. with 100 formulas (nesting lev.20)		
Prover	**Limit 5 ms**	**Limit 1 s**
VINTE	45	0
CondLean 3.1	173	141
GoalDUCK	136	133
NESCOND	479	0

Table 3. Number of timeouts over 1000 random sequents using VINTE for $\mathbb{V}$.

Seq. with 20 formulas (nesting lev.10)		
Prover	**Limit 5 ms**	**Limit 1 s**
VINTE	8	2
CondLean 3.1	65	17
GoalDUCK	276	198
NESCOND	46	5

Seq. with 100 formulas (nesting lev.20)		
Prover	**Limit 5 ms**	**Limit 1 s**
VINTE	1	0
CondLean 3.1	180	80
GoalDUCK	327	18
NESCOND	19	6

Table 4. Number of timeouts of VINTE for extensions of $\mathbb{V}$ (average of different systems) over 1000 random sequents.

Concerning 2, we have considered 76 valid formulas obtained by translating K valid formulas provided by Heuerding in conditional formulas: $\Box A$ is replaced by $\top \Box\!\!\rightarrow A$[8], whereas $\Diamond A$ is replaced by $\neg(\top \Box\!\!\rightarrow \neg A)$. We have compared the performance of VINTE, implementation for $\mathbb{VN}$, with those of NESCOND, the best prover among those taken into account for conditional logics based on the selection function semantics [15]. As expected, the performance of NESCOND is still significantly better than those of VINTE: fixing a time limit of 1ms, NESCOND is able to check the validity of the considered formula in the 86 % of cases, whereas VINTE is able to answer only in the 11 % of cases. However, VINTE is able to reach a percentage of successes of 37 % by extending the time limit to 1 second, and over 60% in 3 seconds (even if, in this last case, NESCOND is not able to answer only in 2 cases over 76). Obviously, this result is justified by the fact that VINTE supports stronger systems of conditional logics with respect

[8] It is worth noticing that this translation introduces an exponential blowup.

to NESCOND, which is specifically tailored for CK and all the proposed results are restricted to such weaker system supported by both provers.

6 Conclusions and Future Issues

We have presented VINTE, a theorem prover implementing internal calculi for Lewis' conditional logics introduced in [6]. Our long term project is to develop both calculi and theorem provers for the whole family of Lewis' logics. One further step in this direction is represented by the hypersequent calculi for containing both uniformity (all worlds have the same set of accessible worlds) and total reflexivity presented in [7]. Notice that an implementation of *hypersequent* calculi is an interesting task in itself.

We also aim at improving the performances of VINTE by implementing standard refinements and heuristics. We also intend to extend VINTE to handle countermodel generation for unprovable formulas. Last, as mentioned in the previous section, we are currently developing a set of benchmarks for VINTE for a more detailed analysis of the performances of the theorem prover.

References

1. Alenda, R., Olivetti, N., Pozzato, G.L.: Nested sequent calculi for normal conditional logics. Journal of Logic and Computation 26(1), 7–50 (2016)
2. Baltag, A., Smets, S.: The logic of conditional doxastic actions. Texts in Logic and Games, Special Issue on New Perspectives on Games and Interaction 4, 9–31 (2008)
3. Beckert, B., Posegga, J.: leantap: Lean tableau-based deduction. Journal of Automated Reasoning 15(3), 339–358 (1995)
4. Board, O.: Dynamic interactive epistemology. Games and Economic Behavior 49(1), 49–80 (2004)
5. Genovese, V., Giordano, L., Gliozzi, V., Pozzato, G.L.: Logics in access control: a conditional approach. Journal of Logic and Computation 24(4), 705–762 (2014)
6. Girlando, M., Lellmann, B., Olivetti, N., Pozzato, G.L.: Standard sequent calculi for Lewis' logics of counterfactuals. In: Logics in Artificial Intelligence. JELIA 2016., vol. 10021, pp. 272–287. Springer (2016)
7. Girlando, M., Lellmann, B., Olivetti, N., Pozzato, G.L.: Hypersequent calculi for lewis' conditional logics with uniformity and reflexivity. accepted to Tableaux (2017)
8. Grahne, G.: Updates and counterfactuals. Journal of Logic and Computation 8(1), 87–117 (1998)
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. Blackwell (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: TABLEAUX 2005. LNAI, vol. 3702, pp. 328–332. Springer (2005)
13. Olivetti, N., Pozzato, G.L.: Theorem Proving for Conditional Logics: CondLean and GoalDuck. J. of Applied Non-Classical Logics 18(4), 427–473 (2008)
14. Olivetti, N., Pozzato, G.L.: NESCOND: an implementation of nested sequent calculi for conditional logics. In: IJCAR 2014. LNCS, vol. 8562, pp. 511–518. Springer (2014)

15. Olivetti, N., Pozzato, G.L.: Nested sequent calculi and theorem proving for normal conditional logics: The theorem prover NESCOND. Intelligenza Artificiale 9(2), 109–125 (2015)
16. Olivetti, N., Pozzato, G.L.: A standard internal calculus for Lewis' counterfactual logics. In: TABLEAUX 2015, LNAI, vol. 9323, pp. 270–286. Springer (2015)

APPENDIX

Pictures of VINTE

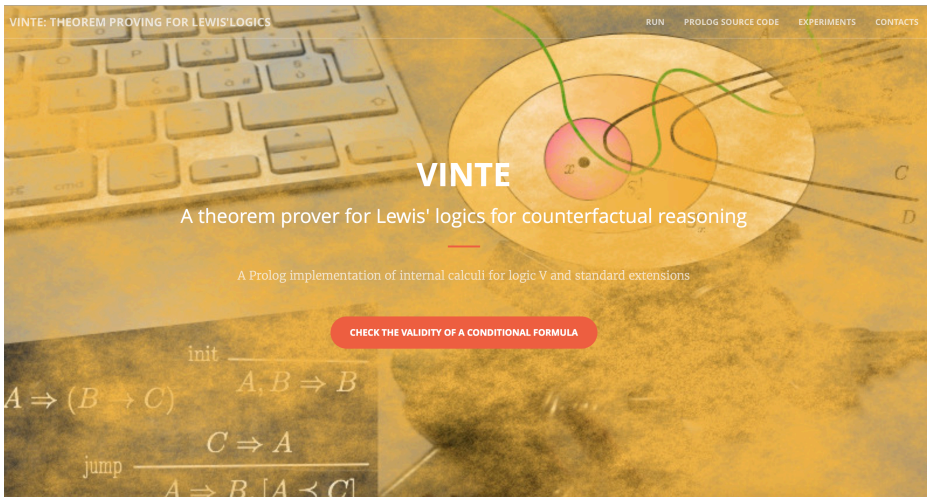

Fig. 3. Home page of VINTE.

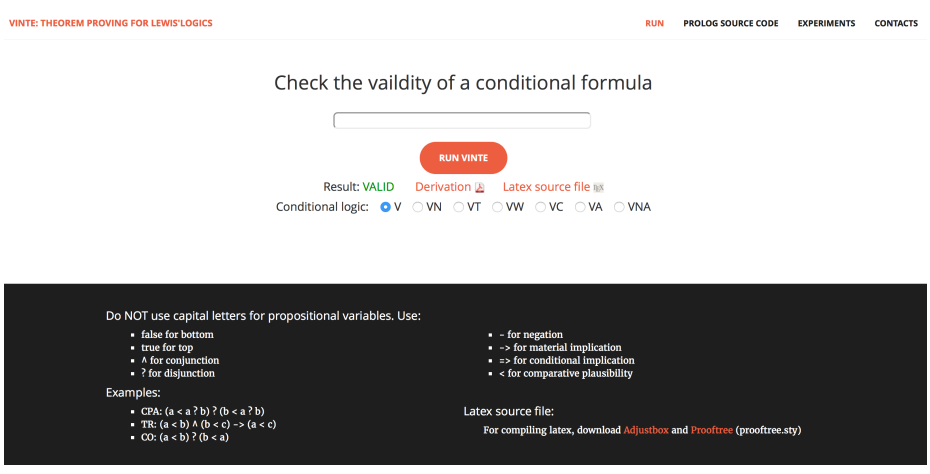

Fig. 4. When the user wants to check whether a formula F is valid, then (i) he selects the conditional logic to use, (ii) he types F in the form and (iii) clicks the button in order to execute the calculi presented in Section 3.

VINTE - Derivation for $a \preccurlyeq b \vee b \preccurlyeq a$ in logic $\mathbb{VN}$

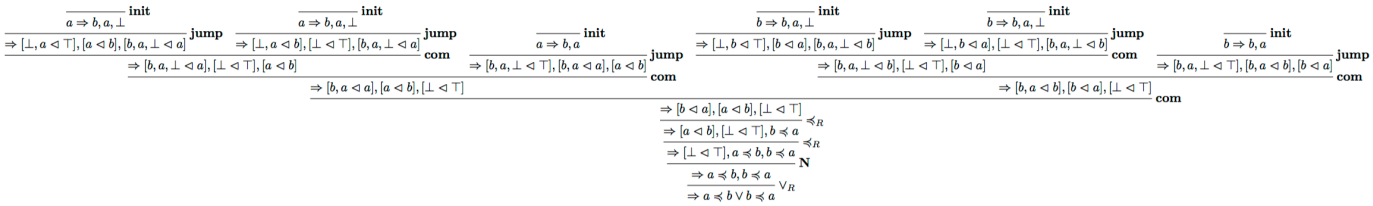

Fig. 5. When the submitted formula is valid, then the user can have a look at the derivation built by VINTE, stored in a pdf file. As an alternative, the user can download the LaTeX source file of the derivation.