Theorem proving for conditional logics: CondLean and GOALD$\mathcal{U}$CK

Nicola Olivetti[*] — **Gian Luca Pozzato**[**]

[*] *LSIS - UMR CNRS 6168*
Université Paul Cézanne — Aix-Marseille 3
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
nicola.olivetti@{univ.u-3mrs.fr,lsis.org}

[**] *Dipartimento di Informatica*
Università degli Studi di Torino
corso Svizzera 185 - 10149 Turin (Italy)
pozzato@di.unito.it

ABSTRACT. *In this paper we focus on theorem proving for conditional logics. First, we give a detailed description of CondLean, a theorem prover for some standard conditional logics. CondLean is a SICStus Prolog implementation of some labeled sequent calculi for conditional logics recently introduced. It is inspired to the so called "lean" methodology, even if it does not fit this style in a rigorous manner. CondLean also comprises a graphical interface written in Java. Furthermore, we introduce a goal-directed proof search mechanism, derived from the above mentioned sequent calculi based on the notion of uniform proofs. Finally, we describe* GOALD$\mathcal{U}$*CK, a simple SICStus Prolog implementation of the goal-directed calculus mentioned here above. Both the programs CondLean and* GOALD$\mathcal{U}$*CK, together with their source code, are available for free download at .*

KEYWORDS: *conditional logics, theorem proving, sequent calculi, goal-directed proof methods, labeled deductive systems.*

DOI:10.3166/JANCL.18.427–473

1. Introduction

Conditional logics have a long history. They have been studied first by Stalnaker and Lewis (Lewis, 1973; Nute, 1980; Chellas, 1975; Stalnaker, 1968) in order to for-

malize a kind of hypothetical reasoning (if A were the case then B), that cannot be captured by classical logic with material implication. Recently, conditional logics have found interesting applications in several areas of computer science and artificial intelligence, such as knowledge representation, non-monotonic reasoning (Delgrande, 1987; Kraus *et al.*, 1990; Crocco *et al.*, 1992; Friedman *et al.*, 2001; Giordano *et al.*, 2005b), deductive databases (Gabbay *et al.*, 2000a), belief revision (Grahne, 1998; Giordano *et al.*, 2005a; Giordano *et al.*, 2002) and natural language semantics (Costello *et al.*, 1999). Conditional logics have also been used to model causal inference and reasoning about action execution in planning (Schwind, 1999; Giordano *et al.*, 2004).

In spite of their significance, very few proof systems have been proposed for these logics (Artosi *et al.*, 2002; de Swart, 1983; Gent, 1992; Lamarre, 1993; Delgrande *et al.*, 1990; Crocco *et al.*, 1995; Giordano *et al.*, 2003). As a direct consequence, theorem provers for conditional logics have been scarcely investigated.

In (Olivetti *et al.*, 2007) labeled sequent calculi SeqS are introduced for CK and extensions with axioms ID, MP, CS (corresponding to the axiom of strong centering), and CEM (corresponding to Stalnaker's axiom of conditional excluded middle); moreover, a goal-directed proof search procedure for a fragment of the basic normal conditional logic CK is presented.

The main contributions of this work are as follows:

1) *Theorem proving for conditional logics*. We describe in detail a theorem proving system based on SeqS calculi. Our program, called CondLean (Olivetti *et al.*, 2003; Olivetti *et al.*, 2005a), is implemented in SICStus Prolog[1] and supports the basic conditional system CK, its extensions with ID, MP, CS, and CEM and all their combinations, except those combining both CEM and MP[2]; as far as we know, this is the first theorem prover for these logics.
For each supported conditional system, CondLean offers two different implementations, namely:

- a simple version, called *constant labels*, where Prolog *constants* are used to represent SeqS's labels;
- a more efficient one, called *free variables*, where labels are represented by Prolog *variables*, inspired by the free-variable tableaux presented in (Beckert *et al.*, 1997).

CondLean follows to a certain extent the "lean" methodology (Beckert *et al.*, 1995; Fitting, 1998). The program implements a predicate `prove`, and each clause of this predicate represents a sequent rule or axiom. Proof search is provided for free by the mere depth-first search mechanism of Prolog, without any meta-level algorithm of search strategy. In this way, the philosophy underlaying the "lean" methodology is "to achieve maximal efficiency from minimal means" (Beckert *et al.*, 1995), that is to say to write short programs and exploit the power of Prolog's engine as much as possible.

1. CondLean also comprises a graphical interface implemented in Java.
2. For systems with both CEM and MP the problem of finding cut-free sequent calculi is open at present (see (Olivetti *et al.*, 2007) for details).

The core of CondLean consists of 93 lines of code, implementing three predicates and 24 clauses.
It is worth noticing that CondLean is only inspired to the "lean" methodology, but it does not fit its style in a rigorous manner; we also present an alternative "*really-lean*" implementation of SeqS calculi, and discuss how CondLean seems a better solution for the calculi implemented in this work.

2) *Goal-directed proof procedures for conditional logics*. We develop a goal-directed proof search mechanisms for conditional logics. This extension might be the base of a language for reasoning hypothetically about change and actions in a logic programming framework. The basic idea of goal-directed proof search is as follows: given a sequent $\Gamma \vdash G$, one can interpret Γ as the program or database, and G as a goal whose proof is searched. The backward proof search of the sequent is driven by the goal G, in the sense that the goal is stepwise decomposed according to its logical structure.
We give a goal-directed proof procedure, called $\mathcal{U}$S', for a fragment of CK and its extensions with ID and MP. Moreover, we introduce GOALD$\mathcal{U}$CK, a very compact SICStus Prolog implementation of the calculi $\mathcal{U}$S'. As far as we know, no other goal-directed theorem prover for the above standard conditional logics has been previously described in the literature. Further investigation could lead to the development of extensions of logic programming based on conditional logics.
A goal-directed proof procedure for a fragment of CK (without implementation) was given in (Olivetti *et al.*, 2007). This work extends the results in (Olivetti *et al.*, 2007) in two directions: (i) we consider extensions of CK with ID and MP; (ii) the fragment of the language considered is significantly richer.

The plan of the paper is as follows: in section 2 we briefly introduce conditional logics, then we recall their sequent calculi SeqS in section 3. In section 4 we describe CondLean, our theorem prover for conditional logics. In this section we also compare CondLean, implementation for CK, with leanCK, an alternative implementation of the sequent calculi for CK following the style of propositional leanT^AP in a more rigorous way; leanT^AP is a "lean" theorem prover for classical logic introduced in (Fitting, 1998). In section 5 we present goal-directed calculi for CK and its extensions with MP and ID; we call these calculi $\mathcal{U}$S'. In section 6 we introduce GOALD$\mathcal{U}$CK, a SICStus Prolog implementation of $\mathcal{U}$S'. We conclude this work by giving some experimental results on the performance of the theorem provers introduced (section 7), furthermore discussing about some related works (section 8).

2. Conditional logics

Conditional logics are extensions of classical logic by the conditional operator $\Rightarrow$. As in (Olivetti *et al.*, 2007), we restrict our concern to propositional conditional logics.

A propositional conditional language $\mathcal{L}$ is defined from:

– a set of propositional variables *ATM*;

– the symbols of *false* $\bot$ and *true* $\top$;
– a set of connectives $\neg$, $\rightarrow$, $\vee$, $\wedge$, $\Rightarrow$.

We define formulas of $\mathcal{L}$ as follows:

– $\bot$, $\top$, and the propositional variables of *ATM* are *atomic formulas*;
– if A is a formula, then $\neg A$ is a *complex formula*;
– if *A* and *B* are formulas, $A \rightarrow B$, $A \vee B$, $A \wedge B$, and $A \Rightarrow B$ are *complex formulas*.

Similarly to modal logics, the semantics of conditional logics can be defined in terms of possible world structures. In this respect, conditional logics can be seen as a generalization of modal logics (or a type of multi-modal logic) where the conditional operator is a sort of modality indexed by a formula of the same language. The two most popular semantics for conditional logics are the so-called *sphere semantics* (Lewis, 1973) and the *selection function semantics* (Nute, 1980). Both of them are possible-world semantics, but are based on different (though related) algebraic notions. As in (Olivetti *et al.*, 2007), here we adopt the more general solution of 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 *closer* to w given 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. — *A selection function model is a triple* $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$ *where:*

– $\mathcal{W}$ *is a non empty set of items called* worlds*;*
– f *is the so-called* selection function *and has the following type:*

$$f\colon \mathcal{W} \times 2^{\mathcal{W}} \longrightarrow 2^{\mathcal{W}}$$

– $[\,]$ *is the* evaluation function, *which assigns to an atom* $P \in ATM$ *the set of worlds where* P *is true, and is extended to the other formulas as follows:*

- $[\bot] = \emptyset$
- $[\top] = \mathcal{W}$
- $[\neg A] = \mathcal{W} - [A]$
- $[A \rightarrow B] = (\mathcal{W} - [A]) \cup [B]$
- $[A \vee B] = [A] \cup [B]$
- $[A \wedge B] = [A] \cap [B]$
- $[A \Rightarrow B] = \{w \in \mathcal{W} \mid f(w, [A]) \subseteq [B]\}$

Notice that we have defined f taking $[A]$ rather than A; 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 normal conditional system*, called CK. An axiomatization of the CK system is given by:

– any axiomatization of classical propositional logic
– (Modus Ponens) $\dfrac{A \quad A \rightarrow B}{B}$

– (RCEA) $\dfrac{A \leftrightarrow B}{(A \Rightarrow C) \leftrightarrow (B \Rightarrow C)}$

– (RCK) $\dfrac{(A_1 \wedge \cdots \wedge A_n) \rightarrow B}{(C \Rightarrow A_1 \wedge \cdots \wedge C \Rightarrow A_n) \rightarrow (C \Rightarrow B)}$

Similarly to modal logic, extensions of the basic system CK are obtained by assuming further properties on the selection function. Following (Olivetti *et al.*, 2007), in this paper we consider the following standard extensions:

Name	Axiom	Model condition
ID	$A \Rightarrow A$	$f(w,[A]) \subseteq [A]$
MP	$(A \Rightarrow B) \rightarrow (A \rightarrow B)$	$w \in [A] \rightarrow w \in f(w,[A])$
CS	$(A \wedge B) \rightarrow (A \Rightarrow B)$	$w \in [A] \rightarrow f(w,[A]) \subseteq \{w\}$
CEM	$(A \Rightarrow B) \vee (A \Rightarrow \neg B)$	$\mid f(w,[A]) \mid \leq 1$

The above axiomatization is complete with respect to the semantics (Nute, 1980). Let AX be the set of axioms considered, *i.e.* AX={CEM, CS, ID, MP}; from now on, we use S to denote any subset of AX, *i.e.* S $\subseteq$ AX.

3. Sequent calculi SeqS

In (Olivetti *et al.*, 2007) labeled sequent calculi SeqS are introduced. These calculi are inspired to labeled deductive systems introduced in (Gabbay, 1996) and in (Viganò, 2000).

In Figure 1 we present SeqS; the calculi make use of *labeled formulas*, where the labels are drawn from a denumerable set $\mathcal{A}$; there are two kinds of formulas:

– *world formulas*, denoted by $x : A$, where $x \in \mathcal{A}$ and $A \in \mathcal{L}$;

– *transition formulas*, denoted by $x \xrightarrow{A} y$, where $x, y \in \mathcal{A}$ and $A \in \mathcal{L}$.

A world formula $x : A$ is used to represent that A holds in the possible world represented by the label x; a transition formula $x \xrightarrow{A} y$ represents that $y \in f(x, [A])$.

In Figure 1, $\Sigma[x, y/u]$ denotes the substitution of labels x and y with the label u wherever they occur in Σ. Rules (ID), (MP), (CS) and (CEM) are only used in corresponding extensions of the basic system SeqCK. Some rules have two premises (introducing a branch in a backward proof search); we denote with $(*)$ and $(**)$ those two premises, respectively. The rules for connectives $\top, \neg, \vee, \wedge$ are as usual and therefore omitted to save space.

A *sequent* is a pair $\langle \Gamma, \Delta \rangle$, usually denoted with $\Gamma \vdash \Delta$, where Γ and Δ are multisets of labeled formulas. The intuitive meaning of $\Gamma \vdash \Delta$ is: every model that satisfies all labeled formulas of Γ in the respective worlds (specified by the labels) satisfies at least one of the labeled formulas of Δ (in those worlds). The definition of sequent validity is as follows:

DEFINITION 2 (SEQUENT VALIDITY (OLIVETTI *et al.*, 2007)). — *Given a model*

$$\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$$

for $\mathcal{L}$, and a label alphabet $\mathcal{A}$, we consider any mapping

$$I : \mathcal{A} \to \mathcal{W}$$

Let F be a labeled formula, we define $\mathcal{M} \models_I F$ as follows:

- $\mathcal{M} \models_I x$*:* A iff $I(x) \in [A]$
- $\mathcal{M} \models_I x \xrightarrow{A} y$ iff $I(y) \in f(I(x), [A])$

We say that $\Gamma \vdash \Delta$ is valid *in $\mathcal{M}$ if for every mapping $I : \mathcal{A} \to \mathcal{W}$, if $\mathcal{M} \models_I F$ for every $F \in \Gamma$, then $\mathcal{M} \models_I G$ for some $G \in \Delta$. We say that $\Gamma \vdash \Delta$ is valid in a system (CK or one of its extensions) if it is valid in every $\mathcal{M}$ satisfying the specific conditions for that system (if any).*

$$(\mathbf{AX}) \quad \Gamma, x : P \vdash \Delta, x : P \quad (P \in ATM) \qquad (\mathbf{A}\bot) \quad \Gamma, x : \bot \vdash \Delta$$

$$(\to \mathbf{L}) \quad \frac{(*)\Gamma \vdash x : A, \Delta \qquad (**)\Gamma, x : B \vdash \Delta}{\Gamma, x : A \to B \vdash \Delta} \qquad (\to \mathbf{R}) \quad \frac{\Gamma, x : A \vdash x : B, \Delta}{\Gamma \vdash x : A \to B, \Delta}$$

$$(\Rightarrow \mathbf{L}) \quad \frac{\begin{array}{c}(*)\Gamma, x : A \Rightarrow B \vdash x \xrightarrow{A} y, \Delta \\ (**)\Gamma, x : A \Rightarrow B, y : B \vdash \Delta\end{array}}{\Gamma, x : A \Rightarrow B \vdash \Delta} \qquad (\Rightarrow \mathbf{R}) \quad \frac{\Gamma, x \xrightarrow{A} y \vdash y : B, \Delta}{\Gamma \vdash x : A \Rightarrow B, \Delta} \; (y \notin \Gamma, \Delta)$$

$$(\mathbf{EQ}) \quad \frac{(*)u : A \vdash u : B \qquad (**)u : B \vdash u : A}{\Gamma, x \xrightarrow{A} y \vdash x \xrightarrow{B} y, \Delta}$$

$$(\mathbf{ID}) \quad \frac{\Gamma, x \xrightarrow{A} y, y : A \vdash \Delta}{\Gamma, x \xrightarrow{A} y \vdash \Delta} \qquad (\mathbf{CEM}) \quad \frac{\begin{array}{c}(*)\Gamma, x \xrightarrow{A} y \vdash \Delta, x \xrightarrow{A} z \\ (**)(\Gamma, x \xrightarrow{A} y \vdash \Delta)[y, z/u]\end{array} \; \begin{array}{l}(y \neq z, \\ u \notin \Gamma, \Delta)\end{array}}{\Gamma, x \xrightarrow{A} y \vdash \Delta}$$

$$(\mathbf{MP}) \quad \frac{\Gamma \vdash x \xrightarrow{A} x, x : A, \Delta}{\Gamma \vdash x \xrightarrow{A} x, \Delta} \qquad (\mathbf{CS}) \quad \frac{\begin{array}{c}(*)\Gamma, x \xrightarrow{A} y \vdash \Delta, x : A \\ (**)\Gamma[x, y/u], u \xrightarrow{A} u \vdash \Delta[x, y/u]\end{array} \; \begin{array}{l}(x \neq y, \\ u \notin \Gamma, \Delta)\end{array}}{\Gamma, x \xrightarrow{A} y \vdash \Delta}$$

Figure 1. *Sequent calculi SeqS*

Systems combining two or more semantic conditions are characterized by all rules capturing those conditions; systems allowing *both* (CEM) and (MP) axioms are not considered in this work. As usual, we say that a sequent $\Gamma \vdash \Delta$ is *derivable* if there is a closed tree having $\Gamma \vdash \Delta$ as a root. A tree is *closed* if all its leaves are instances of the axioms (**AX**) and (**A**$\bot$).

As an example, in Figure 2 we present a derivation in SeqID of the valid sequent $\vdash x : (P \to Q) \Rightarrow (P \to Q)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{x \overset{P\to Q}{\longrightarrow} y, y : P \vdash y : Q, y : P \qquad x \overset{P\to Q}{\longrightarrow} y, y : Q, y :, P \vdash y : Q}{x \overset{P\to Q}{\longrightarrow} y, y : P \to Q, y : P \vdash y : Q}\,(\to L)}{x \overset{P\to Q}{\longrightarrow} y, y : P \vdash y : Q}\,(ID)}{x \overset{P\to Q}{\longrightarrow} y \vdash y : P \to Q}\,(\to R)}{\vdash x : (P \to Q) \Rightarrow (P \to Q)}\,(\Rightarrow R)$$

Figure 2. *A derivation in SeqID of* $\vdash x : (P \to Q) \Rightarrow (P \to Q)$

SeqS calculi are sound and complete with respect to the semantics:

THEOREM 3 (SOUNDNESS AND COMPLETENESS (OLIVETTI *et al.*, 2007)). — *A sequent* $\Gamma \vdash \Delta$ *is valid if and only if* $\Gamma \vdash \Delta$ *is derivable in SeqS.*

In order to obtain a terminating calculus, observe that for all rules, except ($\Rightarrow$ L), (ID), (MP), (CS), and (CEM), the premises have a lower complexity than the conclusion. Therefore, to ensure termination, we have to control the application of these rules. To this regard, in (Olivetti *et al.*, 2007) the following results are obtained:

LEMMA 4 (OLIVETTI *et al.*, 2007). — *SeqS systems are sound and complete even if the* ($\Rightarrow$ *L) rule is applied to* $\Gamma, x : A \Rightarrow B \vdash \Delta$ *with the following restrictions:*

– *(*$\Rightarrow$ *L) is applied at most once by using the same transition* $x \overset{A}{\longrightarrow} y$ *on the same conditional formula* $x : A \Rightarrow B$ *in each branch of a proof tree;*

– *(*$\Rightarrow$ *L) is applied by using a transition formula* $x \overset{A}{\longrightarrow} y$ *such that there exists* $x \overset{C}{\longrightarrow} y \in \Gamma$ *or* $x = y$.

In systems SeqCEM{+CS}{+ID} one can control the application of (CEM) by means of the same restrictions.

The first claim of Lemma 4 is easy to see. The second one says that, in a backward application of $(\Rightarrow\ L)$ to a sequent $\Gamma, x : A \Rightarrow B \vdash \Delta$, the premises of the rule $(*)\Gamma, x : A \Rightarrow B \vdash x \overset{A}{\longrightarrow} y, \Delta$ and $(**)\Gamma, x : A \Rightarrow B, y : B \vdash \Delta$ are such that either there is a transition $x \overset{C}{\longrightarrow} y$ belonging to Γ or $x = y$. This can be (intuitively)

shown as follows. If a derivation of $(*)$ is ended by the introduction (looking forward) of $x \xrightarrow{A} y$ in the right-hand side of the sequent, and $x \xrightarrow{A} y$ has not been introduced by weakening, then $x \xrightarrow{A} y$ is used in an application of either (MP) or (EQ). In the first case, we have that $x = y$. In the second case, a transition $x \xrightarrow{C} y$ must belong to Γ.

LEMMA 5 (OLIVETTI *et al.*, 2007). — *There is no need to apply (ID), (MP), and (CS) on the same transition $x \xrightarrow{A} y$ more than once in each branch of a derivation.*

Intuitively, Lemma 5 says that the contraction on transition formulas is admissible in SeqS. Consider (ID) as an example. If the rule is applied twice on $\Gamma, x \xrightarrow{A} y \vdash \Delta$, then we obtain a sequent of the form $\Gamma, x \xrightarrow{A} y, y : A, y : A \vdash \Delta$. The second instance of $y : A$ is redundant, and so is the second application of (ID) on the same transition $x \xrightarrow{A} y$. Roughly speaking, if the principal connective in A is a classical connective, then we do not need to keep a copy of $y : A$ in the premise(s) when the corresponding classical rule is applied to it. Otherwise, if A is a conditional formula $B \Rightarrow C$, then if we need to apply $(\Rightarrow L)$ on $y : B \Rightarrow C$ more than once, then we do not need to "duplicate" it at this stage too: indeed, the rule $(\Rightarrow L)$ keeps $y : B \Rightarrow C$ in its premises, in order to allow further applications to the same conditional formula. The same for the rules (MP) and (CS).

In a backward proof search, the rules can be only applied a finite number of times in accordance with the restrictions in Lemmas 4 and 5. This ensures termination of SeqS:

THEOREM 6 (TERMINATION OF SEQS (OLIVETTI *et al.*, 2007)). — *SeqS ensure a terminating proof search.*

For the basic system CK and its extension CK+ID, in (Olivetti *et al.*, 2007) it is also shown that one can reformulate the crucial rule ($\Rightarrow$ L) with a non-invertible one, that is to say that the principal formula $x : A \Rightarrow B$ is not copied into the two premises of the rule. Moreover, considering a backward application of ($\Rightarrow$ L) to $\Gamma, x : A \Rightarrow B \vdash \Delta$, the left premise of the rule, *i.e.* the one in which a transition $x \xrightarrow{A} y$ is introduced in the right-hand side of the sequent, can be restricted to a sequent of the form $x \xrightarrow{C} y \vdash x \xrightarrow{A} y$, such that $x \xrightarrow{C} y \in \Gamma$. This refinement is stated by the following theorem:

THEOREM 7 (OLIVETTI *et al.*, 2007). — *SeqCK and SeqID are sound and complete even if ($\Rightarrow$ L) is formulated as follows:*

$$\frac{x \xrightarrow{C} y \vdash x \xrightarrow{A} y \qquad \Gamma, x \xrightarrow{C} y, y : B \vdash \Delta}{\Gamma, x \xrightarrow{C} y, x : A \Rightarrow B \vdash \Delta}\ (\Rightarrow L)$$

4. Design of CondLean

In this section we present CondLean, an implementation of the sequent calculi SeqS (Olivetti *et al.*, 2003; Olivetti *et al.*, 2005a). It is a SICStus Prolog program inspired by the "lean" methodology introduced by leanTAP ((Beckert *et al.*, 1995; Fitting, 1998)). The program CondLean comprises a set of clauses, each one of them represents a sequent rule or axiom. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism.

As mentioned in the Introduction, CondLean offers two different implementations, namely the *constant labels* version and the *free variables* version. We describe each one of them in detail.

4.1. *The constant labels version*

We represent each component of a sequent (antecedent and consequent) by a *list* of formulas, partitioned into three sub-lists: atomic formulas, transitions and complex formulas. Atomic and complex formulas are represented by a list like `[x,a]`, where `x` is a Prolog constant and `a` is a formula. A transition $x \xrightarrow{A} y$ is represented by `[x,a,y]`. SeqS's labels are represented by Prolog's constants. The sequent calculi are implemented by the predicate

prove(Cond, Sigma, Delta, Labels).

which succeeds if and only if $\Sigma \vdash \Delta$ is derivable in SeqS, where `Sigma` and `Delta` are the lists representing the multisets Σ and Δ, respectively and `Labels` is the list of labels introduced in that branch. `Cond` is a list of pairs of the kind $[F, Used]$, where F is a conditional formula `[X,A => B]` and $Used$ is a list of transitions $[[X, C_1, Y_1], ..., [X, C_n, Y_n]]$ such that ($\Rightarrow$ L) has already been applied to $x : A \Rightarrow B$ by using transitions $x \xrightarrow{C_i} y_i$. The list `Cond` is used in order to ensure the termination of the proof search, by applying the restriction stated by Lemma 4 above. In fact, ($\Rightarrow$ L) is applied to $x : A \Rightarrow B$ by introducing $x \xrightarrow{A} y$ such that there exists $x \xrightarrow{C_j} y$ that belongs to `Sigma` and $[X, C_j, Y]$ *does not belong to* `Used`.

For instance, to prove $x : B \vdash x : A \Rightarrow A, x : B \vee B$ in CK, one queries CondLean with the goal

```
prove([],[[[x,b]],[],[]], [[],[],[[x,a=>a],[x,b or b]]], [x]).
```

Each clause of the `prove` predicate implements one axiom or rule of SeqS. The following ones are clauses implementing axioms:

```
prove(_,[LitSigma,_,_],[LitDelta,_,_],_) :-
  member(F,LitSigma),member(F,LitDelta),!.
```

```
prove(_,[LitSigma,_,_],_,_):-
  member([_,false],LitSigma),!.
```

The above clauses succeed if the same atomic formula F belongs to both the left-hand side and the right-hand side of the sequent and if a formula $x : \bot$ belongs to the left-hand side of the sequent, respectively. As another example, one of the clauses[3] implementing ($\Rightarrow$ L) is presented below. The predicate

`put([Y,B],LitSigma,ComplexSigma,NewLitSigma,NewComplexSigma)`

is used to put `[Y,B]` in the proper sub-list of the antecedent of a sequent $\Sigma \vdash \Delta$. The predicate `select(Elem,Cond,TempCond)` succeeds if `Elem` belongs to the list `Cond`; in this case, `TempCond` is obtained by removing `Elem` from `Cond`.

```
prove(Cond,[LitSigma,TransSigma,ComplexSigma],
      [LitDelta,TransDelta,ComplexDelta], Labels):-
  member([X,A => B],ComplexSigma),
  select([[X,A => B],Used],Cond,TempCond),
  member([X,C,Y],TransSigma), \+member([X,C,Y],Used),!,
  put([Y,B],LitSigma,ComplexSigma,NewLitSigma,NewComplexSigma),
  prove([[[X, A => B],[[X,C,Y] | Used]] | TempCond],
     [LitSigma,TransSigma,ComplexSigma],
     [LitDelta,[[X,A,Y]|TransDelta],ComplexDelta],Labels),
  prove([[[X, A => B],[[X,C,Y] | Used]] | TempCond],
     [NewLitSigma,TransSigma,NewComplexSigma],
     [LitDelta,TransDelta,ComplexDelta],Labels).
```

To search a derivation of a sequent $\Sigma \vdash \Delta$, CondLean proceeds as follows. First of all, if $\Sigma \vdash \Delta$ is an axiom, the goal will succeed immediately by using the clauses for the axioms. If it is not, then the first applicable rule will be chosen, *e.g.* if `ComplexDelta` contains a formula `[X,A -> B]`, then the clause for ($\rightarrow$ R) rule will be used, invoking `prove` on the unique premise of ($\rightarrow$ R). CondLean 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.

In systems containing the (CEM) rule, another parameter, called `CEM`, is added to the predicate `prove`. This parameter is needed in order to control the application of (CEM) by means of the restrictions stated by Lemma 4, and it is used in the same way as `Cond` is used to control the applications of ($\Rightarrow$ L). More in detail, `CEM` is a list of pairs $[[X, A, Y], \mathit{Used}]$, where Used is a list of transitions $[[X, C_1, Y_1], ..., [X, C_n, Y_n]]$ such that the (CEM) rule has already been applied to $x \xrightarrow{A} y$ by using transitions $x \xrightarrow{C_i} y_i$. The clause for (CEM) is applied to a transition $x \xrightarrow{A} y$ by introducing

3. There are other clauses implementing ($\Rightarrow$ L), taking care of cases when `Cond` is empty and when $x \xrightarrow{A} x$ is used (systems allowing MP only).

$x \xrightarrow{A} z$ such that there exists $x \xrightarrow{C_j} z$ in `Sigma` and $[X, C_j, Z]$ *does not belong to* `Used`.

As explained at the end of section 3, in order to describe a decision procedure for the logics under consideration, we also need to control the application of (ID), (MP), and (CS) in systems allowing them; in particular, Lemma 5 states that it is useless to apply each of these rules more than once on the same transition $x \xrightarrow{A} y$ in the same branch of a proof tree. Consider the case of SeqID: in order to apply (ID) by means of this restriction, we add to the predicate `prove` a further argument, called `ID`, which is simply the *list of transitions* $x \xrightarrow{A} y$ to which the (ID) rule has already been applied in the current branch. The application of (ID) is restricted to transitions *not belonging* to `ID`.
In systems comprising (MP) or (CS) we adopt the same strategy, by adding arguments `MP` or `CS`, respectively, to the predicate `prove`. In systems implementing combinations of the presented axioms, the predicate `prove` is equipped with all the corresponding additional arguments; for instance, in the Prolog program implementing SeqCEM+ID+CS, the predicate `prove` has the following type:

```
prove(Cond,CEM,CS,ID,Sigma,Delta,Labels).
```

Here below is a clause implementing (CEM) in SeqCEM+ID+CS, considering the situation in which (CEM) has never been applied to $x \xrightarrow{A} y$ in the current branch, *i.e.* the list `CEM` does not contain an item for that transition:

```
prove(Cond,CEM,CS,ID,[LitSigma,TransSigma,ComplexSigma],
    [LitDelta,TransDelta,ComplexDelta],Labels):-
  member([X,A,Y],TransSigma),
  \+member([[X,A,Y],_],CEM),
  member([X,C,Z],TransSigma),
  Y\=Z,!,
  prove(Cond,[[[X,A,Y],[[X,C,Z]]]|CEM],CS,ID,
     [LitSigma,TransSigma,ComplexSigma],
     [LitDelta,[[X,A,Z]|TransDelta],ComplexDelta],Labels),
  generaLabels(U,Labels),
  ...
  labelSubstitution(LitSigma,Z,U,TempLitSigma),
  labelSubstitution(TempLitSigma,Y,U,DefLitSigma),
  ...
  labelSubstitution(ComplexDelta,Z,U,TempComplexDelta),
  labelSubstitution(TempComplexDelta,Y,U,DefComplexDelta),
  prove(NewCond,[[[X,A,Y],[[X,C,Z]]]|CEM],CS,ID,
     [DefLitSigma,DefTransSigma,DefComplexSigma],
     [DefLitDelta,DefTransDelta,DefComplexDelta],[U|Labels]).
```

This clause is applied when a transition `[X,A,Y]` belongs to the list `TransSigma`, and another transition `[X,C,Z]` belongs to the same list. The predicate `labelSubstitu-`

`tion` replaces the labels `Z` and `Y` with the new label `U` wherever they occur in each list `LitSigma`, `TransSigma`, and so on, obtaining the new lists `DefLitSigma`, `DefTransSigma`, ..., on which the predicate `prove` is applied for the second recursive call.

4.2. *The free variables version*

We first discuss an alternative version, called *free variables*, for the basic system CK and its extension with ID. For these systems, CondLean also implements the reformulated version of the calculi for CK{+ID} stated by Theorem 7, obtained by replacing the original ($\Rightarrow$ L) rule with the following one:

$$\frac{x \xrightarrow{C} y \vdash x \xrightarrow{A} y \qquad \Gamma, x \xrightarrow{C} y, y : B \vdash \Delta}{\Gamma, x \xrightarrow{C} y, x : A \Rightarrow B \vdash \Delta} (\Rightarrow L)$$

The clause corresponding to the above non-invertible version of ($\Rightarrow$ L) is as follows:

```
prove([LitSigma,TransSigma,ComplexSigma],[LitDelta,TransDelta,
      ComplexDelta],Labels) :-
   select([X,A => B],ComplexSigma,ResComplexSigma),
   member([X,C,Y],TransSigma),
   prove([[],[[X,C,Y]],[]],[[],[[X,A,Y]],[]],Labels),
   put([Y,B],LitSigma,ResComplexSigma,NewLitSigma,NewComplexSigma),
   prove([NewLitSigma,TransSigma,NewComplexSigma],[LitDelta,TransDelta,
      ComplexDelta],Labels).
```

The predicate `select` removes the formula `[X,A => B]` from the list `ComplexSigma`, then the conditional formula is not copied into the premises.

When the above ($\Rightarrow$ L) clause is used to prove $\Gamma, x : A \Rightarrow B \vdash \Delta$, a backtracking point is introduced by the choice of a label `Y` occurring in the two premises of the rule; in case of failure, Prolog's backtracking tries every instance of the rule with every available label (if more than one). Choosing, sooner or later, the right label to apply ($\Rightarrow$ L) may strongly affect the theorem prover's efficiency: if there are *n* labels to choose for an application of ($\Rightarrow$ L) the computation might succeed only after *n-1* backtracking steps, with a significant loss of efficiency.

Our second implementation, called *free variables*, makes use of *Prolog variables* to represent all the labels that can be used in a single application of the ($\Rightarrow$ L) rule. This version represents labels by integers starting from 1; by using integers we can easily express constraints on the range of the variable-labels. To this regard the library `clpfd` is used to manage free-variable domains. As an example, in order to prove Σ',

1: $A \Rightarrow B \vdash \Delta$ the theorem prover will call `prove` on the following premises: $\Sigma' \vdash \Delta$, *1* $\xrightarrow{A}$ *Y* and *Y: B,* $\Sigma' \vdash \Delta$, where *Y* is a Prolog variable. This variable will be then instantiated by Prolog's pattern matching to apply either the (EQ) rule, or to *close a branch with an axiom*. Here below is the clause implementing the ($\Rightarrow$ L) rule:

```
prove([LitSigma,TransSigma,ComplexSigma],[LitDelta,
      TransDelta,ComplexDelta],Max):-
   select([X,A => B],ComplexSigma,ResComplexSigma),
   domain([Y],1,Max), Y#>X,
   put([Y,B],LitSigma,ResComplexSigma,NewLitSigma,NewComplexSigma),
   prove([NewLitSigma,TransSigma,NewComplexSigma],
      [LitDelta,TransDelta,ComplexDelta],Max),
   prove([LitSigma,TransSigma,ResComplexSigma],
      [LitDelta,[[X,A,Y]|TransDelta],ComplexDelta],Max).
```

The atom `Y#>X` adds the constraint `Y>X` to the constraint store: the constraints solver will verify the consistency of it during the computation. In SeqCK and SeqID we can only use labels introduced *after* the label `X`, thus we introduce the previous constraint.

We have tested CondLean, CK system, on a valid sequent with 65 labels on the antecedent: the free variable version succeeds in less than 60 mseconds, whereas the constant labels version, even considering the reformulated one where ($\Rightarrow$ L) is reformulated as stated by Theorem 7, takes 460 mseconds.

The reformulation presented above is only applicable to systems SeqCK and SeqID, whereas in all the other systems we have to consider the invertible formulation of ($\Rightarrow$ L) given in Figure 1 and rewrited here for a better readability:

$$\frac{\Gamma, x : A \Rightarrow B \vdash \Delta, x \xrightarrow{A} y \qquad \Gamma, x : A \Rightarrow B, y : B \vdash \Delta}{\Gamma, x : A \Rightarrow B \vdash \Delta} (\Rightarrow L)$$

Moreover, Lemma 4 allows to restrict the choice of the label y to use in the rule application in a way such that there exists $x \xrightarrow{C} y \in \Gamma$.
The clause implementing this version of ($\Rightarrow$ L) does not introduce a backtracking point, since the principal formula $x : A \Rightarrow B$ is copied in both the premises. However, if several transitions $x \xrightarrow{C_1} y_1, x \xrightarrow{C_2} y_2, \ldots, x \xrightarrow{C_n} y_n$ belong to Γ, then the choice of the label to use in the premises is crucial for CondLean's performance: indeed, choosing later the right label to use increases the dimension of the proof search space, affecting the performance of the theorem prover.

In order to avoid this problem, we have also developed a free variable version for *all* the systems of conditional logics presented here. The idea is the same as the free variable version described above for the systems SeqCK and SeqID: Prolog variables

are used to represent all the labels that can be used in a single application of the ($\Rightarrow$ L) rule, and labels are represented by integers starting from 1. In order to prove $\Sigma', 1: A \Rightarrow B \vdash \Delta$ the theorem prover will call `prove` on the following premises: $\Sigma', 1 : A \Rightarrow B \vdash \Delta, 1 \xrightarrow{A} Y$ and $\Sigma', 1 : A \Rightarrow B, Y : B \vdash \Delta$, where Y is a Prolog variable. This variable will be then instantiated by Prolog's pattern matching to apply either the (EQ) rule, or to close a branch with an axiom. In order to guarantee termination, we have to take care of Lemma 4, that is to say:

– the domain of Y is restricted to the values of labels $y_1, y_2, \ldots, y_n$ such that $x \xrightarrow{C_1} y_1, x \xrightarrow{C_2} y_2, \ldots x \xrightarrow{C_n} y_n \in \Sigma'$;

– the predicate `prove` is equipped by the additional argument `Cond` described in the previous subsection, needed to avoid multiple applications of ($\Rightarrow$ L) with the same transition.

We present one of the clauses[4] implementing the invertible ($\Rightarrow$ L) in the free variable version:

```
prove(Cond,[LitSigma,TransSigma,ComplexSigma],
      [LitDelta,TransDelta,ComplexDelta],Max) :-
  member([X,A => B],ComplexSigma),
  \+member([[X,A => B],_,_],Cond),
  y_domain(X,TransSigma,YDomain),
  list_to_fdset(YDomain,Y_FD_Domain),
  Y in_set Y_FD_Domain,
  put([Y,B],LitSigma,ComplexSigma,NewLitSigma,NewComplexSigma),
  prove([[[X,A => B],[Y],ok]|Cond],
     [NewLitSigma,TransSigma,NewComplexSigma],
     [LitDelta,TransDelta,ComplexDelta], Max),
  prove([[[X,A => B],[Y],ok]|Cond],[LitSigma,TransSigma,ComplexSigma],
     [LitDelta,[[X,A,Y]|TransDelta],ComplexDelta],Max).
```

The predicate `y_domain` returns a Prolog list, called `YDomain`, corresponding to the domain of `Y`, the free variable introduced by the rule application; for technical reasons, the predicate `list_to_fdset` is applied to convert this list into a set. The line `Y in_set Y_FD_Domain` updates the constraint store with the new free variable and its corresponding domain. In order to avoid further applications of ($\Rightarrow$ L) on `[X,A => B]` by using the same values for `Y`, a pair `[[[X,A => B],YDomain]` is added to the list `Cond`.

4. In order to increase readability, we do not present the other clauses implementing ($\Rightarrow$ L), considering the case when the rule has already been applied to `[X,A => B]` in the current branch. CondLean's SICStus Prolog source code is available at http://www.di.unito.it/~pozzato/condlean3.2/Prolog source code.

4.3. *CondLean* vs. *a* "really-lean" *implementation*

In the previous sections we have described CondLean, a theorem prover for some standard conditional logics. CondLean is *inspired* by the "lean" methodology, introduced by Beckert and Posegga (Beckert *et al.*, 1995) and analyzed by Fitting (Fitting, 1998). In (Beckert *et al.*, 1995) an efficient and elegant first-order theorem prover, called leanT^AP, is presented. leanT^AP is a very short Prolog program consisting of only five clauses, and makes use of Prolog's search mechanism and backtracking as much as possible.

CondLean is only inspired to the "lean" methodology, but it does not fit its style in a rigorous manner. The main differences between our implementation and a "*really-lean*" one are the following:

– CondLean makes use of some auxiliary predicates, such as `put`, `select`, and `member`, whereas leanT^AP only relies on Prolog's clause indexing scheme and backtracking;

– the first argument of the predicate `prove` in leanT^AP is the next formula to be processed, which is always the leftmost formula in a single-sided sequent; this allows it to use the first-argument indexing refinements available in SICStus Prolog. CondLean does not present this characteristic, so it cannot take advantage of this refinement.

In this section we briefly present a possible "*really-lean*" implementation of SeqS calculi, in order to compare its performance with CondLean's. We have restricted our concern to the basic system CK, and considered a single-sided sequent calculus for it obtained from SeqCK. We have then implemented this single-sided sequent calculus in SICStus Prolog, obtaining a program called leanCK. leanCK is shorter than CondLean (without taking into account predicates used to generate new labels and to interact with the graphical interface, leanCK comprises only 19 clauses whereas CondLean needs 30 clauses); however, as we will discuss at the end of this section, leanCK's performance are worse than CondLean's.

We say that an atomic formula or its negation is a *literal*, *i.e.* formulas $x : P$, $x : \neg P$, with $P \in ATM$, are literals. We also represent with $\neg(x \xrightarrow{A} y)$ a transition formula $x \xrightarrow{A} y$ belonging to the right-hand side of a sequent.

We now consider sequents having the form $\Gamma \vdash \Lambda \mid\mid T$, where Γ is a multiset of labeled formulas, Λ is a multiset of literals, and T is a multiset of transition formulas. The basic idea is that complex formulas always belong to the left-hand side of the sequent, whereas the right-hand side only contains literals and transitions, used to close a branch in the derivation and to apply the (EQ) rule, respectively.

For propositional formulas, we adopt the well-known α/β classification of Smullyan (Smullyan, 1968), in which α's are conjunctions and β's are disjunctions:

α	α_1	α_2
$x : A \wedge B$	$x : A$	$x : B$
$x : \neg(A \vee B)$	$x : \neg A$	$x : \neg B$
$x : \neg(A \rightarrow B)$	$x : A$	$x : \neg B$

β	β_1	β_2
$x : \neg(A \wedge B)$	$x : \neg A$	$x : \neg B$
$x : A \vee B$	$x : A$	$x : B$
$x : A \rightarrow B$	$x : \neg A$	$x : B$

The single-sided sequent calculus for CK is presented in Figure 3. We consider $P, Q \in ATM$, and suppose that $P \neq Q$. Moreover, in the thinning for transitions, we suppose that $x \neq u$ or $y \neq w$. $(*)$ and $(**)$ are used to denote the two premises of branching rules.

We have then realized leanCK, a SICStus Prolog implementation of the calculus in Figure 3. As for CondLean, the calculus is implemented by a predicate `prove`, having the form

```
prove(Fml,UnExp,Lits,Trans,Labels,Cond).
```

where the first three arguments are defined as for propositional leanT^AP introduced by Fitting in (Fitting, 1998). In particular, `Fml` is the formula currently being expanded in a given branch, `UnExp` is the list of formulas that have not yet been considered, and `Lits` is the list of literals occurring in that branch. `Trans` is the list of transitions (the multiset T in Figure 3). `Labels` is the list of labels occurring in that branch, and `Cond` is a list of pairs `[[X,A => B],`$Used$`]` needed to control the application of ($\Rightarrow$ L) in the same manner as described for CondLean.

First, the program presents clauses for the classification of Smullyan:

```
type([X,A and B],conjunction,[X,A],[X,B]).
type([X,neg (A or B)],conjunction,[X,neg A],[X,neg B]).
...
type([X,A or B],disjunction,[X,A],[X,B]).
...
```

The rest of the very small Prolog code of leanCK consists in the clauses for the predicate `prove`. As an example, the clause implementing the $\beta-$rule is as follows:

```
prove(Fml,UnExp,Lits,Trans,Labels,Cond) :-
   type(Fml,disjunction,Beta1,Beta2),!,
   prove(Beta1,UnExp,Lits,Trans,Labels,Cond),
   prove(Beta2,UnExp,Lits,Trans,Labels,Cond).
```

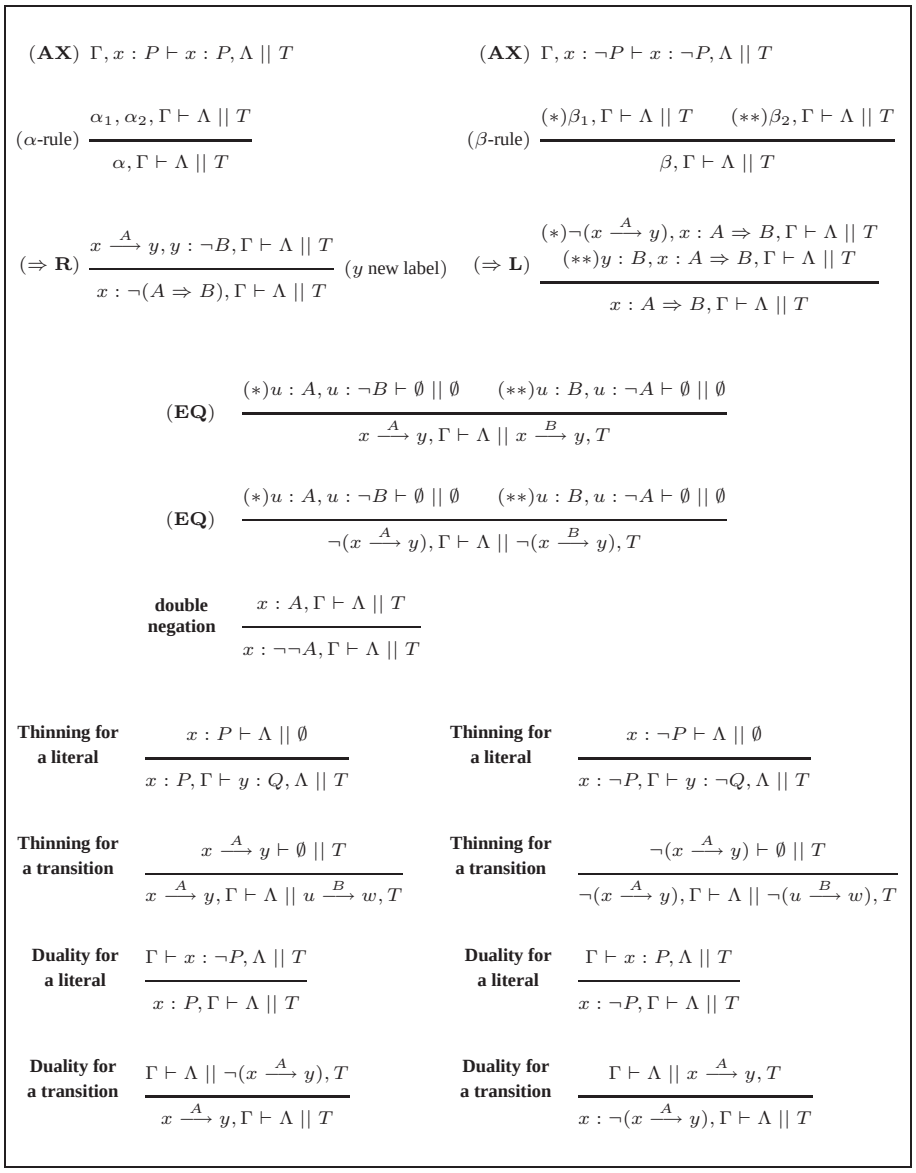

Figure 3. *Single-sided sequent calculus for CK*

leanCK proceeds as follows: if `Fml` is a complex formula, then the corresponding clause is applied, and the proof search goes on with the recursive call(s) on the premise(s) of the rule. If `Fml` is a literal `[X1,L1]`, leanCK first checks if the same literal `[X1,L1]` has already been added to the list `Lits`, in order to conclude the proof by the axioms; in detail:

1) if `[X1,L1]` corresponds to the head of `Lits`, then the computation succeeds;
2) otherwise, the thinning for a literal is applied, in order to find `[X1,L1]` in the tail of the list `Lits`;

If `[X1,L1]` does not belong to `Lits`, then its dual `[X1,neg L1]` is added to `Lits`, corresponding to an application of the rule of duality for a literal. Here are the clauses taking care of literals, implementing the machinery discussed above:

```
prove([X1,L1],_,[[X2,L2]|Lits],_,_,_):-
   (X1=X2,L1=L2,!)  ; prove([X1,L1],[],Lits,[],[],[]).
prove([X,L],[Next|UnExp],Lits,Trans,Labels,Cond):-!,
   prove(Next,UnExp,[[X,neg L]|Lits],Trans,Labels,Cond).
```

leanCK operates in a similar manner if `Fml` is a transition formula `[X,A,Y]` (resp. `[neg,X,A,Y]`); in particular, leanCK tries to find a transition `[X,B,Y]` (resp. `[neg,X,B,Y]`) belonging to the list `Trans`, in order to apply the (EQ) rule. In case of failure, leanCK moves the transition in `Trans` as `[neg,X,A,Y]` (resp. `[X,A,Y]`). The entire SICStus Prolog source code is available at http://www.di.unito.it/~pozzato/condlean 3.2/PrologSource Code/leanck.pl.

Let us briefly discuss about the implementation of the crucial rule ($\Rightarrow$ L). We have tried to respect the "lean" style as much as possible, however we have relaxed some constraints in order to implement the restrictions on the application of the rule (see Lemma 4 above). Here below is the set of clauses used to implement ($\Rightarrow$ L):

```
prove([X,A => B],UnExp,Lits,Trans,Labels,Cond):-
   select([[X,A => B],Used],Cond,NewCond),!,
   (member([X,_,Y],UnExp) ; member([neg,X,_,Y],Trans)),
   \+member(Y,Used),
   append(UnExp,[[X,A => B]],NewUnExp),
   prove([Y,B],NewUnExp,Lits,Trans,Labels,
      [[[X,A => B],[Y|Used]]|NewCond]),
   prove([neg,X,A,Y],NewUnExp,Lits,Trans,Labels,
      [[[X,A => B],[Y|Used]]|NewCond]).
prove([X,A => B],UnExp,Lits,Trans,Labels,Cond):-
   (member([X,_,Y],UnExp) ; member([neg,X,_,Y],Trans)),
   append(UnExp,[[X,A => B]],NewUnExp),
   prove([Y,B],NewUnExp,Lits,Trans,Labels,
```

```
      [[[X,A => B],[Y]]|Cond]),
   prove([neg,X,A,Y],NewUnExp,Lits,Trans,Labels,
      [[[X,A => B],[Y]]|Cond]).
prove([X,A => B],[Next|UnExp],Lits,Trans,Labels,Cond):-!,
   append(UnExp,[[X,A => B]],NewUnExp),
   prove(Next,NewUnExp,Lits,Trans,Labels,Cond).
```

If ($\Rightarrow$ L) has already been applied to `[X,A => B]` in the current branch, then the first clause is invoked, and a label `Y` not belonging to `Used` (*i.e.* not yet been used to apply the rule on that conditional formula) is selected; in order to ensure termination (Lemma 4), `Y` is such that there exists either a transition `[X,_,Y]` in `UnExp` or a transition `[neg,X,_,Y]`, representing $\neg(x \xrightarrow{C} y)$ for any C, in `Trans`. The principal formula `[X,A => B]` is then copied in the two premises of the rule *by appending it at the bottom of the* `UnExp` *list*.

The second clause is similar to the first one, and treats the case when ($\Rightarrow$ L) is applied to `[X,A => B]` for the first time in the current branch.

When the first argument `Fml` is a conditional formula $x : A \Rightarrow B$, leanCK tries to apply ($\Rightarrow$ L) by using a transition $x \xrightarrow{A} y$ such that $x \xrightarrow{C} y$ has been already introduced in the current branch. It could be the case that no transitions $x \xrightarrow{C} y$ are available; as an example, consider the case of `prove([x,a=>(b and c)],[[x,neg(a=>b)]],[],[],[x],[])`: ($\Rightarrow$ L) is not applicable, then we need to *postpone* its application after the application of ($\Rightarrow$ R) on `[x,neg(a=>b)]`.
To this aim, when the first two clauses above fail or are not applicable, then the third and last clause is invoked. It only appends the conditional formula being analyzed at the bottom of `UnExp`, and the computation goes on with the next formula to be expanded (if any).

To test the validity of a conditional formula F one queries leanCK with the following goal:

```
prove([x,neg F],[],[],[],[x],[]).
```

It can be shown that if a formula is valid, *i.e.* there is a proof of it in SeqCK, then leanCK ensures a terminating search; on the contrary, if a formula is not valid, then leanCK does not guarantee termination (with a negative answer) by the presence of the third clause for ($\Rightarrow$ L) discussed above. Consider the following example: one queries leanCK with the goal `prove([x,a=>a],[[x,a=>b]],[],[],[x],[])`; the third clause of ($\Rightarrow$ L) is applied, and the proof search goes on with the recursive call on `prove([x,a=>b],[[x,a=>a]],[],[],[x],[])`, but ($\Rightarrow$ L) is once again not applicable, then `prove([x,a=>a],[[x,a=>b]],[],[],[x],[])` is invoked, and so on, incurring in a loop. To avoid this situation, a loop-checking mechanism is needed.

However, we have not implemented any loop-checking machinery, which would obviously affects the performance of the theorem prover. At this point, we believe

we have enough elements to compare leanCK with CondLean, and conclude that the *"really-lean"* solution is worse than the other.

We have tested both CondLean, constant labels version for CK, and leanCK (with no loop checking mechanism) over 90 sequents valid in CK, obtaining that CondLean offers better performance: in less than 5 seconds, CondLean succeeds in 86 cases, whereas leanCK only answers in 72 cases. The experimental results on the comparison between CondLean and leanCK are presented in detail in section 7.1.

As a consequence, we believe that, in order to implement a labeled calculus like SeqS, a solution only *inspired* to the "lean" methodology as CondLean offers better performance than an implementation following this style in a rigorous manner. CondLean makes a systematic use of auxiliary predicates such as `member`, `select`, and `put` in order to control the derivation. Moreover, by partitioning formulas of each sequent into three sub-lists (atoms, transitions, complex formulas) and by using the `select` (resp. the `member`) predicate, we can have a better control of proof search. For instance we can postpone the application of branching rules (including the critical rule of left conditional) since we can choose the rule to apply by selecting the next complex formula to process, instead of processing always the leftmost one, as it happens in a *"really-lean"* implementation. This reduces the search space, increasing the theorem prover's performance.

5. Goal-directed proof procedure for conditional logics

In this section we investigate how SeqS calculi can be used in order to develop goal-directed proof procedures for conditional logics, following the paradigm of Uniform Proof by Miller and others (Miller *et al.*, 1991; Gabbay *et al.*, 2000b). Some preliminary results are given in (Olivetti *et al.*, 2007).

The paradigm of uniform proof can be seen as a generalization of conventional logic programming. Given a sequent $\Gamma \vdash G$, one can interpret Γ as the *program* or the *knowledge base* or the *database*, whereas G can be seen as a *goal* whose proof is searched. Intuitively, the basic idea is that the backward proof of $\Gamma \vdash G$ is driven by the goal G; roughly speaking, the goal G is stepwise decomposed according to its logical structure by the rules of the calculus, until its atomic constituents are reached. To prove an atomic goal Q, the proof search mechanism checks if Γ contains a "clause" whose head matches with Q and then tries to prove the "body" of the clause.

The logical connectives in G can be interpreted operationally as simple and fixed search instructions. In our case, the situation will be as follows: a database consists of formulas labeled by worlds and transitions between worlds labeled by formulas, and we have a goal to be proved in a specific world. In this respect, for instance, the operational meaning of the rule for conditional goals is to expand the database with a new transition.

Given a sequent calculus, not every valid sequent admits a uniform proof of this kind; in order to describe a goal-directed proof search one must identify a fragment of the corresponding logic that allows uniform proofs.

In (Olivetti *et al.*, 2007) a simple goal-directed calculus $\mathcal{U}$CK for a fragment of CK has been presented. In this work, we extend it to the systems with axioms ID and MP (and the combination of them) presenting a goal-directed calculus called $\mathcal{U}$S', where S' stands for {CK, CK+ID, CK+MP, CK+ID+MP}. Moreover, here we consider a broader fragment of the conditional language $\mathcal{L}$ allowing uniform proofs.

First of all, we specify the fragment of CK{+ID}{+MP} we consider[5]. We distinguish between the formulas which can occur in the program (or knowledge base or database), called D-formulas, and the formulas that can be asked as goals, called G-formulas.

DEFINITION 8 (LANGUAGE FOR UNIFORM PROOFS). — *We consider the fragment of CK{+ID}{+MP}, called* $\mathcal{LU}(CK\{+ID\}\{+MP\})$*, comprising:*

- database formulas, *denoted with* D
- goal formulas, *denoted with* G
- transition formulas *of the form* $x \xrightarrow{A} y$

defined as follows ($Q \in ATM$*):*

$$D = G \rightarrow Q \mid A \Rightarrow D$$

$$G = Q \mid \top \mid G \wedge G \mid G \vee G \mid A \Rightarrow G$$

$$A = Q \mid A \wedge A \mid A \vee A$$

We define a database Γ *as a set of* D*-formulas and transition formulas.*

As a difference from the calculus presented in (Olivetti *et al.*, 2007), formulas of kind A can be either atomic formulas or combinations of conjunctions and disjunctions of atomic formulas[6]. As mentioned above, here we restrict our concern to the basic system CK and its extensions with axioms ID and MP. We denote sequent calculi for these systems with SeqS', *i.e.* we use SeqS' to refer to all the following systems: SeqCK, SeqID, SeqMP, and SeqID+MP. We call $\mathcal{U}$S' the goal-directed proof procedures introduced for S' systems.

5. We present here a fragment of the language that allows uniform proofs and that might have some practical interest. We do not claim that this is a maximal fragment allowing uniform proofs. The determination of a maximal fragment is an issue that we shall explore in future research.

6. The fragment considered can be easily extended by allowing formulas of type A of the form $A \Rightarrow A$ and allowing falsity $\bot$ as a head of a D-formula, without changing essentially the proof procedure displayed below.

The rules of the calculi $\mathcal{U}$S' are presented in Figure 4. $\mathcal{U}$S' 's rules are able to prove a goal which is either a formula $x : G$ or a transition formula $x \xrightarrow{A} y$; from now on we use γ to denote a goal of both kinds. Given a goal γ whose proof is searched from a database Γ, we call $\Gamma \vdash_{GD} \gamma$ a *query* . We write $\Gamma \vdash_{GD} \gamma \Rrightarrow \Gamma_i \vdash_{GD} \gamma_i$ to denote that the sequent $\Gamma \vdash_{GD} \gamma$ is reduced to sequents $\Gamma_i \vdash_{GD} \gamma_i$. The rule ($\mathcal{U}$ **prop**) is used when D-formulas have the form $A_1 \Rightarrow A_2 \Rightarrow ... \Rightarrow A_n \Rightarrow (G \rightarrow Q)$, where G could be $\top$.

The rule $(\mathcal{U} \Rightarrow)_{\mathbf{CK}}$ belongs to $\mathcal{U}$CK and $\mathcal{U}$MP only, whereas $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$ is used in $\mathcal{U}$ID and $\mathcal{U}$ID+MP. The rule ($\mathcal{U}$ **trans**)$_{\mathbf{MP}}$ only belongs to the calculi $\mathcal{U}$MP and $\mathcal{U}$ID+MP. Here below we give a detailed list of the rules belonging to each goal-directed calculus:

- $\mathcal{U}$CK : ($\mathcal{U}\top$), ($\mathcal{U}$**ax**), ($\mathcal{U}$ **prop**), ($\mathcal{U}\wedge$), ($\mathcal{U}\vee$), ($\mathcal{U}$ **trans**), $(\mathcal{U}\Rightarrow)_{\mathbf{CK}}$

- $\mathcal{U}$ID : ($\mathcal{U}\top$), ($\mathcal{U}$**ax**), ($\mathcal{U}$ **prop**), ($\mathcal{U}\wedge$), ($\mathcal{U}\vee$), ($\mathcal{U}$ **trans**), $(\mathcal{U}\Rightarrow)_{\mathbf{ID}}$

- $\mathcal{U}$MP : ($\mathcal{U}\top$), ($\mathcal{U}$**ax**), ($\mathcal{U}$ **prop**), ($\mathcal{U}\wedge$), ($\mathcal{U}\vee$), ($\mathcal{U}$ **trans**), $(\mathcal{U}\Rightarrow)_{\mathbf{CK}}$, ($\mathcal{U}$ **trans**)$_{\mathbf{MP}}$

- $\mathcal{U}$ID+MP : ($\mathcal{U}\top$), ($\mathcal{U}$**ax**), ($\mathcal{U}$ **prop**), ($\mathcal{U}\wedge$), ($\mathcal{U}\vee$), ($\mathcal{U}$ **trans**), $(\mathcal{U}\Rightarrow)_{\mathbf{ID}}$, ($\mathcal{U}$ **trans**)$_{\mathbf{MP}}$

Given a formula of type A, *i.e.* either an atomic formula or a boolean combination of conjunctions and disjunctions of atomic formulas, the operation $\mathsf{Flat}(x : A)$ has the effect of *flatten* the conjunction/disjunction into a set of atomic D-formulas:

DEFINITION 9 (FLAT OPERATION). —

- *Flat*$(x : Q) = \{\{x : Q\}\}$, *with* $Q \in ATM$;
- *Flat*$(x : F \vee G) =$ *Flat*$(F) \cup$ *Flat*(G)
- *Flat*$(x : F \wedge G) = \{S_F \cup S_G \mid S_F \in$ *Flat*(F) *and* $S_G \in$ *Flat*$(G)\}$

This operation is needed when *rules introducing a formula of type A in the database* are applied, since an A-formula might not be a D-formula. These rules, namely ($\mathcal{U}$ **trans**) and $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$, introduce a formula of type A in the left-hand side of the sequent, *i.e.* a formula possibly being a combination of conjunctions and disjunctions of atoms. This formula needs to be decomposed in its atomic components. Indeed, in order to search a derivation for a transition formula $x \xrightarrow{A} y$, when ($\mathcal{U}$ **trans**) is applied by considering a transition $x \xrightarrow{A'} y$ in the program (or database), then the calculus leads to search a derivation for both $u : A' \vdash_{GD} u : A$ and $u : A \vdash_{GD} u : A'$; intuitively, this step corresponds to an application of the (EQ) rule in SeqS. As an example, suppose that A has the form $Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n$; in this case, in order to prove $u : A \vdash_{GD} u : A'$, we need to flat the database, thus proving the following sequent: $u : Q_1, u : Q_2, \ldots, u : Q_n \vdash_{GD} u : A'$. In case A has the form $Q_1 \vee Q_2$, then we need to prove both $u : Q_1 \vdash u : A'$ and $u : Q_2 \vdash u : A'$. The same happens when $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$ is applied to $\Gamma \vdash_{GD} x : A \Rightarrow B$, and the computation steps to prove $\Gamma, x \xrightarrow{A} y, y : A \vdash_{GD} y : B$, and $y : A$ needs to be decomposed as defined here above.

Moreover, we write $\Gamma, \mathsf{Flat}(x : A) \vdash_{GD} \gamma$ to denote that the goal γ can be derived from *all* the databases obtained by applying Flat to $x : A$ (and adding formulas in Γ), that is to say:

DEFINITION 10. — *Given a database* Γ*, a formula* A *and a goal* G*, let Flat*$(x : A) = \{S_1, S_2, \ldots, S_n\}$.

We write

$$\Gamma, \textit{Flat}(x : A) \vdash_{GD} \gamma$$

if and only if

$$\forall i = 1, 2, \ldots, n \textit{ we have that } \Gamma, S_i \vdash_{GD} \gamma$$

We restrict our concern to computations beginning with initial queries. Intuitively, an *initial query* comprises a database of the form $x : D_1, x : D_2, \ldots, x : D_n$, where $D_1, D_2, \ldots, D_n$ are D-formulas, and a goal $x : G$. These queries are the only ones having a direct logical interpretation, namely to prove that G is a logical consequence of $D_1, D_2, \ldots, D_n$.

DEFINITION 11 (INITIAL QUERY). — *Given a database* $x : D_1, x : D_2, \ldots, x : D_n$*, where* $D_1, D_2, \ldots, D_n$ *are formulas of type* D*, we call* initial query *a query in which the proof of* $x : G$ *is searched, where* G *is a formula of type* G.

We can observe that no rule of the calculi $\mathcal{U}$S' remove a formula from the database. In particular, starting with an initial query, the rules $(\mathcal{U} \Rightarrow)_{\mathbf{CK}}$ and $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$ introduce a transition $x \xrightarrow{A} y$ in the database, where y is a new label. As a direct consequence, the set of transitions in the database forms a *tree*. This is stated by Definitions 12 and 13 below.

DEFINITION 12 (MULTIGRAPH OF TRANSITIONS). — *Given a database* Γ*, where* $\Gamma = \Gamma', T$ *and* T *is the multiset of transition formulas and* Γ' *does not contain transition formulas, and a goal* γ*,* i.e. *either a formula* $x : G$ *or a transition* $x \xrightarrow{A} y$*, we define the multigraph* $\mathcal{G} = \langle V, E \rangle$ *associated to* $\Gamma \vdash_{GD} \gamma$*, with vertexes* V *and edges* E*.* V *is the set of labels occurring in* $\Gamma \vdash_{GD} \gamma$*, and* $< u, v > \in E$ *whenever* $u \xrightarrow{A'} v \in T$.

We call *regular query* a query $\Gamma \vdash_{GD} \gamma$ whose multiset of transitions forms a tree:

DEFINITION 13 (REGULAR QUERY). — *A query* $\Gamma \vdash_{GD} \gamma$ *is called* regular *if the multigraph of transitions* $\mathcal{G}$ *associated with* $\Gamma \vdash_{GD} \gamma$ *is a tree.*

As mentioned, we can show that, given an initial query, the computation leads only to regular queries:

LEMMA 14. — *Every computation started with an initial query and obtained by applying* $\mathcal{U}$*S's rules contains only regular queries.*

$$(\mathcal{U}\top) \quad \Gamma \vdash_{GD} x : \top$$

$$(\mathcal{U}\,\mathbf{ax}) \quad \Gamma \vdash_{GD} x : Q \quad \text{if} \quad x : Q \in \Gamma$$

$$(\mathcal{U}\,\mathbf{prop}) \quad \Gamma \vdash_{GD} x : Q \;\Rightarrow\; \Gamma \vdash_{GD} y \xrightarrow{A_1} x_1,\ \Gamma \vdash_{GD} x_1 \xrightarrow{A_2} x_2,\ \ldots \ldots, \Gamma \vdash_{GD} x_{n-1} \xrightarrow{A_n} x \quad \text{and} \quad \Gamma \vdash_{GD} x : G \quad \text{if} \quad y : A_1 \Rightarrow A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q) \in \Gamma$$

$$(\mathcal{U}\wedge) \quad \Gamma \vdash_{GD} x : G_1 \wedge G_2 \;\Rightarrow\; \Gamma \vdash_{GD} x : G_1 \quad \text{and} \quad \Gamma \vdash_{GD} x : G_2$$

$$(\mathcal{U}\vee) \quad \Gamma \vdash_{GD} x : G_1 \vee G_2 \;\Rightarrow\; \Gamma \vdash_{GD} x : G_1 \quad \text{or} \quad \Gamma \vdash_{GD} x : G_2$$

$$(\mathcal{U}\,\mathbf{trans}) \quad \Gamma \vdash_{GD} x \xrightarrow{A} y \;\Rightarrow\; \mathsf{Flat}(u : A') \vdash_{GD} u : A \quad \text{and} \quad \mathsf{Flat}(u : A) \vdash_{GD} u : A' \quad \text{if } x \xrightarrow{A'} y \in \Gamma, \text{ and } x \neq y$$

$$(\mathcal{U} \Rightarrow)_{\mathbf{CK}} \quad \Gamma \vdash_{GD} x : A \Rightarrow G \;\Rightarrow\; \Gamma, x \xrightarrow{A} y \vdash_{GD} y : G \ (y \text{ new})$$

$$(\mathcal{U}\,\mathbf{trans})_{\mathbf{MP}} \quad \Gamma \vdash_{GD} x \xrightarrow{A} x \;\Rightarrow\; \Gamma \vdash_{GD} x : A$$

$$(\mathcal{U} \Rightarrow)_{\mathbf{ID}} \quad \Gamma \vdash_{GD} x : A \Rightarrow G \;\Rightarrow\; \Gamma, x \xrightarrow{A} y, \mathsf{Flat}(y : A) \vdash_{GD} y : G \ (y \text{ new})$$

Figure 4. *Rules for the calculi $\mathcal{U}$S'*

PROOF. — Given a proof beginning with an initial query, we proceed by induction on the distance between the query we consider and the initial query. For the base case (distance=0), we consider the initial query itself. It only contains one label x, and the database does not contain transitions. Therefore, the multigraph of the transitions is a tree (containing only the vertex x and no edges). For the inductive step, we consider each rule of the calculi $\mathcal{U}$S'. Rules ($\mathcal{U}$ **prop**), ($\mathcal{U} \wedge$), ($\mathcal{U} \vee$), and ($\mathcal{U}$ **trans**)$_{\mathbf{MP}}$ do not add any formula in the database, therefore the database obtained by an application of these rules is a tree by inductive hypothesis. If $(\mathcal{U}\Rightarrow)_{\mathbf{CK}}$ is applied to $\Gamma \vdash_{GD} x : A \Rightarrow G$, we have a query $\Gamma, x \xrightarrow{A} y \vdash_{GD} y : B$, where y is a new label. By inductive hypothesis, Γ is a tree, and so $\Gamma, x \xrightarrow{A} y$ since x occurs in Γ (no rule

removes formulas, thus labels) and y is a new label, then the rule adds both a vertex and an edge to the tree. The same for the rule $(\mathcal{U}\Rightarrow)_{\mathbf{ID}}$. If ($\mathcal{U}$ **trans**) is applied to $\Gamma \vdash x \xrightarrow{A} y$, given $x \xrightarrow{A'} y \in \Gamma$, then the computation leads to prove $\mathsf{Flat}(u : A') \vdash_{GD} u : A$ and $\mathsf{Flat}(u : A) \vdash_{GD} u : A'$: in both cases, the multigraph of transitions only contains a vertex (u) and no edges, *i.e.* it is a tree. ■

In the following we prove that the calculi $\mathcal{U}$S' are sound and complete with respect to the semantics. These calculi for uniform proofs are grounded on the properties of the corresponding calculi SeqS', when derivations are restricted to the fragment $\mathcal{LU}$(CK{+ID}
{+MP}) and to *regular sequents*. Regular sequents are those whose multigraph of transitions forms a *forest* (Definition 5.2 in (Olivetti *et al.*, 2007)). By Lemma 14, in $\mathcal{U}$S' we can restrict our concern to regular queries. Regular queries are obviously regular sequents, then the properties described in section 5 in (Olivetti *et al.*, 2007) also hold for regular queries. These properties are consequences of the refinements on the calculi SeqS', where the two rules (ID) and (MP) are reformulated with the non-invertible ones shown in Figure 5.

$$\frac{\Gamma \vdash \Delta, x : A}{\Gamma \vdash \Delta, x \xrightarrow{A} x}\,(MP) \qquad\qquad \frac{\Gamma, y : A \vdash \Delta}{\Gamma, x \xrightarrow{A} y \vdash \Delta}\,(ID)$$

Figure 5. *Reformulations of the rules (ID) and (MP)*

First, we state the following *disjunction property*, whose proof is similar to the one of Proposition 6.3 in (Olivetti *et al.*, 2007), since that proof is based on the properties of regular sequents, and holding also for systems with axioms ID and MP: if $\Gamma \vdash x : A, y : B$ is derivable, then either $\Gamma \vdash x : A$ or $\Gamma \vdash y : B$ is derivable. Moreover, this property is *height-preserving*, in the sense that the derivation of $\Gamma \vdash x : A$ (respectively $\Gamma \vdash y : B$) has no greater height than $\Gamma \vdash x : A, y : B$. Notice that $x : A$ and $y : B$ are *not* necessarily conditional formulas. This property is stated as follows:

PROPOSITION 15 (HEIGHT-PRESERVING DISJUNCTION PROPERTY). — *Let Γ be a database. If $\Gamma \vdash x_1 : G_1, x_2 : G_2, ..., x_n : G_n$ is derivable in SeqS' with a proof of height h, then there exists $i, i = 1, 2, ..., n$, such that $\Gamma \vdash x_i : G_i$ is derivable in SeqS' with a proof of no greater height than h.*

Now we have all the elements to prove that $\mathcal{U}$S' is sound and complete with respect to the semantics.

THEOREM 16 (SOUNDNESS OF $\mathcal{U}$S'). — *If Γ is a database, γ is a goal (*i.e. *either a formula $x : G$ or a transition formula $x \xrightarrow{A} y$), and $\Gamma \vdash_{GD} \gamma$ is derivable in $\mathcal{U}$S', then $\Gamma \vdash \gamma$ is derivable in the corresponding system SeqS'.*

PROOF. — By induction on the height of the derivation of $\Gamma \vdash_{GD} \gamma$ in $\mathcal{U}$S'.

In the base case, we have that either (1) the goal G is $\top$, then $\Gamma \vdash x : \top$ is also derivable in SeqS', or (2) we have that $\Gamma \vdash_{GD} x : Q$ and $x : Q \in \Gamma$, then $\Gamma \vdash x : Q$ is also derivable in SeqS'. Indeed, $\Gamma \vdash x : \top$ (respectively $\Gamma \vdash x : Q$ with $x : Q \in \Gamma$) is an instance of (AX). For the inductive step, we have to consider each rule of $\mathcal{U}$S' applied to $\Gamma \vdash_{GD} x : G$. To save space, we only present the most interesting cases of ($\mathcal{U}$ **prop**), ($\mathcal{U} \Rightarrow)_{\mathbf{ID}}$, and ($\mathcal{U}$ **trans**)$_{\mathbf{MP}}$. The other cases are similar and left to the reader.

– the proof of $\Gamma \vdash_{GD} x : Q$ is ended by an application of ($\mathcal{U}$ **prop**), *i.e.* $\Gamma \vdash_{GD} y \xrightarrow{A_1} x_1,\ \Gamma \vdash_{GD} x_1 \xrightarrow{A_2} x_2, \ldots, \Gamma \vdash_{GD} x_{n-1} \xrightarrow{A_n} x$ are derivable in $\mathcal{U}$S', and so $\Gamma \vdash_{GD} x : G$. Moreover, we have that $y : A_1 \Rightarrow A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q) \in \Gamma$. By inductive hypothesis, we have that $(1)\Gamma \vdash y \xrightarrow{A_1} x_1,\ (2)\Gamma \vdash x_1 \xrightarrow{A_2} x_2, \ldots, (n)\Gamma \vdash x_{n-1} \xrightarrow{A_n} x$, and $(*)\Gamma \vdash x : G$ are derivable in SeqS'. By $(*)$ and the fact that weakening is admissible in SeqS' (see Theorem 3.10 in (Olivetti *et al.*, 2007)), we have that $(**)\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_n : A_n \Rightarrow (G \to Q) \vdash x : G, x : Q$ is also derivable in SeqS'. We have also that $(***)\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_n : A_n \Rightarrow (G \to Q), x : Q \vdash x : Q$ has a derivation in SeqS', since it is an instance of (AX). We can conclude as follows:

$$\dfrac{(1')\Gamma \vdash y \xrightarrow{A_1} x_1, x : Q \qquad \dfrac{(2')\Gamma\ldots \vdash x_1 \xrightarrow{A_2} x_2, x : Q \qquad \dfrac{\dfrac{(**)\Gamma\ldots \vdash x : G, x : Q \qquad (***)\Gamma\ldots x : Q \vdash x : Q}{(n')\Gamma\ldots \vdash x_{n-1} \xrightarrow{A_n} x, x : Q \mid \Gamma\ldots x : G \to Q \vdash x : Q}(\to L)}{\Gamma\ldots x_{n-1} : A_n \Rightarrow (G \to Q) \vdash x : Q}(\Rightarrow L) \qquad \vdots \qquad \Gamma\ldots x_2 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q) \vdash x : Q}{\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q) \vdash x : Q}(\Rightarrow L)}{\Gamma \vdash x : Q}(\Rightarrow L)$$

where $(1'), (2'), \ldots, (n')$ can be obtained from $(1), (2), \ldots, (n)$ since weakening is height preserving admissible in SeqS';

– the proof of $\Gamma \vdash_{GD} x : A \Rightarrow G$ is ended by an application of $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$, *i.e.* $\Gamma, x \xrightarrow{A} y, \mathsf{Flat}(y : A) \vdash_{GD} y : G$ has a derivation in $\mathcal{U}$ID{+MP}. By inductive hypothesis, we have that $\Gamma, x \xrightarrow{A} y, \mathsf{Flat}(y : A) \vdash y : G$ is derivable in SeqID{+MP}, from we can conclude that $\Gamma, x \xrightarrow{A} y, y : A \vdash y : G$ is derivable in SeqS' by applying the rules ($\wedge$ L) and ($\vee$ L). As an example, suppose that A is a formula of the kind $A_1 \wedge (A_2 \vee A_3)$, then $\mathsf{Flat}(y : A) = \{\{y : A_1, y : A_2\}, \{y : A_1, y : A_3\}\}$. By inductive hypothesis we have that the sequents $\Gamma, x \xrightarrow{A} y, y : A_1, y : A_2 \vdash y : G$ and $\Gamma, x \xrightarrow{A} y, y : A_1, y : A_3 \vdash y : G$ are both derivable in SeqS'; by applying ($\wedge$ L) and ($\vee$ L) we can find the following derivation of $\Gamma, x \xrightarrow{A} y, y : A \vdash y : G$ in SeqS':

$$\dfrac{\dfrac{\Gamma, x \xrightarrow{A} y, y : A_1, y : A_2 \vdash y : G \qquad \Gamma, x \xrightarrow{A} y, y : A_1, y : A_3 \vdash y : G}{\Gamma, x \xrightarrow{A} y, y : A_1, y : A_2 \vee A_3 \vdash y : G}(\vee L)}{\Gamma, x \xrightarrow{A} y, y : A_1 \wedge (A_2 \vee A_3) \vdash y : G}(\wedge L)$$

Since $\Gamma, x \xrightarrow{A} y, y : A \vdash y : G$ is derivable in SeqS, we can conclude by an application of (ID), followed by an application of $(\Rightarrow \text{R})$, since by definition of $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$ the label y is new:

$$\dfrac{\dfrac{\Gamma, x \xrightarrow{A} y, y : A \vdash y : G}{\Gamma, x \xrightarrow{A} y \vdash y : G}\,(ID)}{\Gamma \vdash x : A \Rightarrow G}\,(\Rightarrow R)$$

– the proof of $\Gamma \vdash_{GD} x \xrightarrow{A} x$ is ended by an application of $(\mathcal{U}$ **trans**$)_{\mathbf{MP}}$, *i.e.* $\Gamma \vdash_{GD} x : A$ has a derivation in $\mathcal{U}${ID+}MP. By inductive hypothesis, $\Gamma \vdash x : A$ is derivable in Seq{ID+}MP, and so $\Gamma \vdash x \xrightarrow{A} x, x : A$ since weakening is admissible. We can therefore conclude that there is a closed tree for $\Gamma \vdash x \xrightarrow{A} x$ by applying (MP) as follows:

$$\dfrac{\Gamma \vdash x \xrightarrow{A} x, x : A}{\Gamma \vdash x \xrightarrow{A} x}\,(MP)$$

■

The soundness with respect to the semantics immediately follows from the fact that SeqS' calculi are sound with respect to selection function models.

In order to prove the completeness of $\mathcal{U}$S' we also need to prove the following lemma:

LEMMA 17. — *Given a goal γ and a database Γ, if the following conditions hold:*

– $x_0 : A_1 \Rightarrow A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \rightarrow Q) \in \Gamma$*;*

– $(*)$ $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \rightarrow Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \rightarrow Q), \ldots, x_{n-1} : A_n \Rightarrow (G \rightarrow Q) \vdash_{GD} \gamma$ *is derivable in $\mathcal{U}$S';*

– $\Gamma \vdash_{GD} x_0 \xrightarrow{A_1} x_1, \Gamma \vdash_{GD} x_1 \xrightarrow{A_2} x_2, \ldots \Gamma \vdash_{GD} x_{n-1} \xrightarrow{A_n} x_n$ *are derivable in $\mathcal{U}$S'*

then $\Gamma \vdash_{GD} \gamma$ is also derivable in $\mathcal{U}$S'.

PROOF. — By induction on the height of the uniform proof of $(*)$. In the base case, we have that $(*)$ is an instance of $(\mathcal{U}$ **ax**$)$, that is to say:

– the goal γ has the form $x : P$, with $P \in ATM$;
– there exists $x : P \in \Gamma$.

It is easy to conclude that also $\Gamma \vdash_{GD} x : P$ is an instance of $(\mathcal{U}$ **ax**$)$, then we are done.
Otherwise, $\gamma = x : \top$, and we obviously conclude that $\Gamma \vdash_{GD} x : \top$ is also derivable.

For the inductive step, we consider each rule of $\mathcal{U}$S' that can be applied as the last step of the derivation of $(*)$.

– $(\mathcal{U} \wedge)$ applied to $\gamma = x : G_1 \wedge G_2$: we have that $(*)$ is obtained by an application of $(\mathcal{U} \wedge)$ from (1) $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \rightarrow Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow$

$(G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : G_1$ and (2) $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : G_2$, respectively. Since the proofs of (1) and (2) have a smaller height than $(*)$, we can apply the inductive hypothesis to observe that $\Gamma \vdash_{GD} x : G_1$ and $\Gamma \vdash_{GD} x : G_2$ are derivable in $\mathcal{U}$S'. We can conclude that $\Gamma \vdash_{GD} x : G_1 \wedge G_2$ is also derivable by an application of ($\mathcal{U} \wedge$);

– ($\mathcal{U} \vee$) applied to $\gamma = x : G_1 \vee G_2$: we have that either (1) $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : G_1$ or (2) $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : G_2$ are derivable in $\mathcal{U}$S' (and with proofs with smaller height than $(*)$). Suppose that (1) is derivable in $\mathcal{U}$S' (the case in which (2) is derivable is symmetric): we can apply the inductive hypothesis, to obtain that $\Gamma \vdash_{GD} x : G_1$ is derivable in $\mathcal{U}$S', then we can conclude that $\Gamma \vdash_{GD} x : G_1 \vee G_2$ is derivable by an application of ($\mathcal{U} \vee$);

– (**$\mathcal{U}$ trans**) applied to $\gamma = x \xrightarrow{A} y$, with $x \neq y$: in this case, we have that there is $x \xrightarrow{A'} y \in \Gamma$ and that $\mathsf{Flat}(u : A') \vdash_{GD} u : A$ and $\mathsf{Flat}(u : A) \vdash_{GD} u : A'$ are derivable in $\mathcal{U}$S'. It is easy to observe that the conditional formulas $x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q)$ are not used in an application of (**$\mathcal{U}$ trans**). Therefore, we can conclude that $\Gamma \vdash_{GD} x \xrightarrow{A} y$ is derivable in $\mathcal{U}$S' by applying (**$\mathcal{U}$ trans**) to $\mathsf{Flat}(u : A') \vdash_{GD} u : A$ and $\mathsf{Flat}(u : A) \vdash_{GD} u : A'$;

– (**$\mathcal{U}$ trans**)$_{\mathbf{MP}}$ applied to $\gamma = x \xrightarrow{A} x$: we have that $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : A$ is derivable with a derivation of a smaller height than $(*)$, then we can apply the inductive hypothesis. We have that $\Gamma \vdash_{GD} x \xrightarrow{A} x$ is derivable in $\mathcal{U}$S', and we conclude by an application of (**$\mathcal{U}$ trans**)$_{\mathbf{MP}}$;

– ($\mathcal{U} \Rightarrow$)$_{\mathbf{CK}}$ applied to $\gamma = x : A \Rightarrow G$: we have that $(*)$ $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : A \Rightarrow G$ has been derived from $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q), x \xrightarrow{A} y \vdash_{GD} y : G$, where y does not occur in Γ and it is different from $x_0, x_1, \ldots, x_n$. We can then apply the inductive hypothesis, obtaining that $\Gamma, x \xrightarrow{A} y \vdash_{GD} y : G$ is derivable, and we are done by an application of ($\mathcal{U} \Rightarrow$)$_{\mathbf{CK}}$;

– ($\mathcal{U} \Rightarrow$)$_{\mathbf{ID}}$ applied to $\gamma = x \xrightarrow{A} y$: the proof is as for the case of ($\mathcal{U} \Rightarrow$)$_{\mathbf{CK}}$, with the only difference that we have that, by inductive hypothesis, $\Gamma, \mathsf{Flat}(y : A), x \xrightarrow{A} y \vdash_{GD} y : G$ is derivable in $\mathcal{U}$S';

– (**$\mathcal{U}$ prop**) is applied to $\gamma = x : P$, with $P \in ATM$: we have to distinguish two different cases:

- P is different from Q: in this case, the rule (**$\mathcal{U}$ prop**) ending the derivation of $(*)$ has been applied by using a clause $y : B_1 \Rightarrow B_2 \Rightarrow \ldots \Rightarrow B_m \Rightarrow (G' \to P) \in \Gamma$, and we have that $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} y \xrightarrow{B_1} y_1, \ldots,$

$\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q)$, $x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots,$ $x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} y_{m-1} \xrightarrow{B_m} x$, and $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q)$, $x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : G'$ are derivable in $\mathcal{U}$S'. By inductive hypothesis, we have that $\Gamma \vdash_{GD} y \xrightarrow{B_1} y_1$, $\ldots$, $\Gamma \vdash_{GD} y_{m-1} \xrightarrow{B_m} x$, and $\Gamma \vdash_{GD} x : G'$ are derivable in $\mathcal{U}$S', then we conclude that also $\Gamma \vdash_{GD} x : P$ is derivable by applying the ($\mathcal{U}$ **prop**) rule;

- ($\mathcal{U}$ **prop**) is applied to $\gamma = x : Q$: here we consider two further alternatives. In the first one, the ($\mathcal{U}$ **prop**) rule is applied by using a clause of the database different from all the clauses $x_i : A_{i+1} \Rightarrow \ldots \Rightarrow (G \to Q)$, with $i = 0, 1, \ldots, n-1$: in this case, we conclude as in the case when ($\mathcal{U}$ **prop**) is applied to $\gamma = x : P$, with P different from Q.

In the second case, we have that a clause $x_i : A_{i+1} \Rightarrow \ldots \Rightarrow (G \to Q)$, $i = 0, 1, \ldots, n-1$ is involved in the application of ($\mathcal{U}$ **prop**), then we proceed as follows. First, we observe that $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q)$, $x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x_i \xrightarrow{A_{i+1}} x'$; $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q)$, $x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x' \xrightarrow{A_{i+2}} x''; \ldots;$ $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q)$, $x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x^\circ \xrightarrow{A_n} x$ are derivable with proofs of a smaller height than $(*)$. We can the apply the inductive hypothesis, and we obtain that also $\Gamma \vdash_{GD} x_i \xrightarrow{A_{i+1}} x'$; $\Gamma \vdash_{GD} x' \xrightarrow{A_{i+2}} x''; \ldots;$ $\Gamma \vdash_{GD} x^\circ \xrightarrow{A_n} x$ are derivable in $\mathcal{U}$S'. Moreover, we have that $\Gamma, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q)$, $x_2 : A_3 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \to Q), \ldots, x_{n-1} : A_n \Rightarrow (G \to Q) \vdash_{GD} x : G$ is derivable with no greater height than $(*)$, then we apply the inductive hypothesis to obtain that $(**)$ $\Gamma \vdash_{GD} x : G$ is derivable in $\mathcal{U}$S'.

By the initial hypothesis, we have that $\Gamma \vdash_{GD} x_0 \xrightarrow{A_1} x_1$, $\Gamma \vdash_{GD} x_1 \xrightarrow{A_2} x_2$, $\ldots$, $\Gamma \vdash_{GD} x_{i-1} \xrightarrow{A_i} x_i$ are also derivable in $\mathcal{U}$S'. We conclude by an application of ($\mathcal{U}$ **prop**) to $\Gamma \vdash_{GD} x_0 \xrightarrow{A_1} x_1$, $\Gamma \vdash_{GD} x_1 \xrightarrow{A_2} x_2$, $\ldots$, $\Gamma \vdash_{GD} x_{i-1} \xrightarrow{A_i} x_i$, $\Gamma \vdash_{GD} x_i \xrightarrow{A_{i+1}} x'$, $\Gamma \vdash_{GD} x' \xrightarrow{A_{i+2}} x''$, $\ldots$, $\Gamma \vdash_{GD} x^\circ \xrightarrow{A_n} x$, and $(**)$ $\Gamma \vdash_{GD} x : G$, obtaining that $\Gamma \vdash_{GD} x : Q$ is derivable in $\mathcal{U}$S'. ■

Now we can prove the completeness of the goal-directed calculi.

THEOREM 18 (COMPLETENESS OF $\mathcal{U}$S'). — *If Γ is a database, γ is a goal (*i.e. *either a formula $x : G$ or a transition $x \xrightarrow{A} y$), and $\Gamma \vdash \gamma$ is derivable in SeqS', then $\Gamma \vdash_{GD} \gamma$ is derivable in $\mathcal{U}$S'.*

PROOF. — By induction on the height of the proof tree in SeqS' for $\Gamma \vdash x : G$. For the base case, we have that if $G = \top$, then we are done by the rule ($\mathcal{U}$ $\top$), otherwise if G is an atom P, then $x : G$ must belong to Γ and we are done by applying the rule ($\mathcal{U}$ **ax**). For the inductive step, we consider all the cases:

– $\Gamma \vdash x : G_1 \wedge G_2$: since the rule ($\wedge$ R) is height-preserving invertible (Theorem 3.11 in (Olivetti *et al.*, 2007) can be easily extended to the rules for boolean connectives) in SeqS', we can consider a proof ended as follows:

$$\frac{\Gamma \vdash x : G_1 \qquad \Gamma \vdash x : G_2}{\Gamma \vdash x : G_1 \wedge G_2}\,(\wedge R)$$

By inductive hypothesis, $\Gamma \vdash_{GD} x : G_1$ and $\Gamma \vdash_{GD} x : G_2$ are derivable in $\mathcal{U}$S', then we conclude by an application of ($\mathcal{U} \wedge$);

– $\Gamma \vdash x : G_1 \vee G_2$: since ($\vee$ R) is height-preserving invertible, there exists a proof ended with an application of ($\vee$ R) to $x : G_1 \vee G_2$:

$$\frac{\Gamma \vdash x : G_1, x : G_2}{\Gamma \vdash x : G_1 \vee G_2}\,(\vee R)$$

By Proposition 15, we can observe that either $\Gamma \vdash x : G_1$ or $\Gamma \vdash x : G_2$ is derivable with a proof of (at most) the same height, thus we can apply the inductive hypothesis obtaining a derivation in $\mathcal{U}$S' of either $\Gamma \vdash_{GD} x : G_1$ or $\Gamma \vdash_{GD} x : G_2$, then we can conclude by an application of ($\mathcal{U} \vee$);

– $\Gamma \vdash x : A \Rightarrow G$: we have to distinguish the cases of (i) $\mathcal{U}$CK and $\mathcal{U}$MP, comprising the rule $(\mathcal{U} \Rightarrow)_{\mathbf{CK}}$, and (ii) $\mathcal{U}$ID and $\mathcal{U}$ID+MP, comprising the rule $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$.

(i) the rule ($\Rightarrow$ R) is height-preserving invertible in SeqCK (resp. SeqMP), therefore we can assume, without loss of generality, that there is a derivation of $\Gamma \vdash x : A \Rightarrow G$ ending as follows:

$$\frac{\Gamma, x \xrightarrow{A} y \vdash y : G}{\Gamma \vdash x : A \Rightarrow G}\,(\Rightarrow R)$$

By inductive hypothesis, $\Gamma, x \xrightarrow{A} y \vdash_{GD} y : G$ is also derivable in $\mathcal{U}$CK (resp. $\mathcal{U}$MP), then we can conclude by an application of $(\mathcal{U} \Rightarrow)_{\mathbf{CK}}$ since y is a new label in an application of ($\Rightarrow$ R);

(ii) since both the rules ($\Rightarrow$ R) and (ID) are height-preserving invertible in SeqS', we can consider a proof tree ended with an application of (ID), immediately followed by an application of ($\Rightarrow$ R), as follows:

$$\dfrac{\dfrac{\Gamma, x \xrightarrow{A} y, y : A \vdash y : G}{\Gamma, x \xrightarrow{A} y \vdash y : G}\,(ID)}{\Gamma \vdash x : A \Rightarrow G}\,(\Rightarrow R)$$

If A is a boolean combination of disjunctions/conjunctions, then there is a proof tree of $\Gamma, x \xrightarrow{A} y, y : A \vdash y : G$ in SeqS' ended by several applications of ($\vee$ L) and ($\wedge$ L), since these rules are height-preserving invertible in SeqS' (Theorem 3.11 in (Olivetti *et al.*, 2007)), according to the connectives in A. As an example, let A be $A_1 \vee A_2$: we have that $\Gamma, x \xrightarrow{A_1 \vee A_2} y, y : A_1 \vdash y : G$ and $\Gamma, x \xrightarrow{A_1 \vee A_2} y, y : A_2 \vdash y : G$ are derivable

in SeqS'. By inductive hypothesis, we have that all the sequents obtained by applying ($\vee$ L) and ($\wedge$ L) to $y : A$ (looking backward) are also derivable in $\mathcal{U}$ID{+MP}; in the above example, $\Gamma, x \xrightarrow{A_1 \vee A_2} y, y : A_1 \vdash_{GD} y : G$ and $\Gamma, x \xrightarrow{A_1 \vee A_2} y, y : A_2 \vdash_{GD} y : G$ are derivable in $\mathcal{U}$S', and this corresponds to the fact that $\Gamma, x \xrightarrow{A} y, \mathsf{Flat}(y : A) \vdash_{GD} y : G$ is derivable in $\mathcal{U}$S'. We then conclude by an application of $(\mathcal{U}\Rightarrow)_{\mathbf{ID}}$;

■

– $\Gamma \vdash x : Q, Q \in ATM$: by Proposition 5.3 in (Olivetti *et al.*, 2007), we can restrict our concern to regular sequents, then $\Gamma \vdash x : Q$ is a sequent whose multigraph of transitions is a forest. Let Γ^*_x be the multiset of formulas in Γ whose labels occur either on the path from the root of the tree containing x and the label x itself, or in the tree having x as a root. Similarly to Theorem 6.5 in (Olivetti *et al.*, 2007), it can be shown that $\Gamma^*_x \vdash x : Q$ is also derivable in SeqS', and with a proof of no greater height than $\Gamma \vdash x : Q$'s. One can observe the following fact: if $\Gamma^*_x \vdash x : Q$ is derivable in SeqS', then there exists $x_0 : A_1 \Rightarrow A_2 \Rightarrow \ldots \Rightarrow A_k \Rightarrow (G' \rightarrow Q) \in \Gamma^*_x$ such that $\Gamma^*_x \vdash x_0 \xrightarrow{A_1} x_1$ and $\Gamma^*_x \vdash x_1 \xrightarrow{A_2} x_2$ and $\ldots$ and $\Gamma^*_x \vdash x_{k-1} \xrightarrow{A_k} x_k$, where $x_k = x$, are also derivable in SeqS' with proofs of no greater height than $\Gamma^*_x \vdash x : Q$. By this fact, we conclude that there is a proof of $\Gamma^*_x \vdash x : Q$ ended by several applications of ($\Rightarrow$ L) as follows (we denote with Δ the set of conditionals $x_1 : A_2 \Rightarrow A_3 \Rightarrow \ldots \Rightarrow A_k \Rightarrow (G' \rightarrow Q), x_2 : A_3 \Rightarrow \ldots \Rightarrow A_k \Rightarrow (G' \rightarrow Q), \ldots, x_{k-1} : A_k \Rightarrow (G' \rightarrow Q)$):

$$\dfrac{(1)\Gamma^*_x \vdash x_0 \xrightarrow{A_1} x_1, x : Q \qquad \dfrac{(2)\Gamma^*_x, x_1 : \ldots \vdash x_1 \xrightarrow{A_2} x_2, x : Q \qquad \cdots \qquad \dfrac{(k)\Gamma^*_x, \Delta \vdash x_{k-1} \xrightarrow{A_k} x, x : Q \quad (\circ)\Gamma^*_x, \Delta, x : G' \rightarrow Q \vdash x : Q}{\Gamma^*_x, x_{k-1} : A_k \Rightarrow \ldots, x_2 : A_3 \Rightarrow \ldots \vdash x : Q}(\Rightarrow L)}{\Gamma^*_x, x_1 : A_2 \Rightarrow \ldots \Rightarrow A_k \Rightarrow (G' \rightarrow Q) \vdash x : Q}(\Rightarrow L)}{\Gamma^*_x \vdash x : Q}(\Rightarrow L)$$

where $(1), (2), \ldots (k)$ are derivable in SeqS' since they can be obtained by weakening from $\Gamma^*_x \vdash x_0 \xrightarrow{A_1} x_1$, $\Gamma^*_x \vdash x_1 \xrightarrow{A_2} x_2$, $\ldots$, $\Gamma^*_x \vdash x_{k-1} \xrightarrow{A_k} x_k$, respectively.
Again by weakening, we can also obtain that $\Gamma \vdash x_0 \xrightarrow{A_1} x_1$, $\Gamma \vdash x_1 \xrightarrow{A_2} x_2$, $\ldots$, $\Gamma \vdash x_{k-1} \xrightarrow{A_k} x$ are also derivable in SeqS', and with derivations of no greater height than $\Gamma \vdash x : Q$. By inductive hypothesis, we can therefore conclude that $(1')\Gamma \vdash_{GD} x_0 \xrightarrow{A_1} x_1$, $(2')\Gamma \vdash_{GD} x_1 \xrightarrow{A_2} x_2$, $\ldots$, $(k')\Gamma \vdash_{GD} x_{k-1} \xrightarrow{A_k} x$ are also derivable in $\mathcal{U}$S'.
Since ($\rightarrow$ L) is height-preserving invertible in SeqS', from $(\circ)$ we can find a proof with at most the same height of $\Gamma^*_x, \Delta \vdash x : G', x : Q$, then of $(\circ\circ)\Gamma, \Delta \vdash x : G', x : Q$ since weakening is height preserving admissible in SeqS'. By applying the height-preserving disjunction property (Proposition 15) to $(\circ\circ)$ we can find a proof of either $\Gamma, \Delta \vdash x : Q$ or $\Gamma, \Delta \vdash x : G'$, to which we can apply the inductive hypothesis obtaining a proof in $\mathcal{U}$S' of either $(i)\Gamma, \Delta \vdash_{GD} x : Q$ or $(ii)\Gamma, \Delta \vdash_{GD} x : G'$, respectively. Moreover, since $(1')$, $(2')$, $\ldots$, and (k') are derivable in $\mathcal{U}$S', by Lemma

17 we have that either $(i')\Gamma \vdash_{GD} x : Q$ is derivable in $\mathcal{U}$S' or $(ii')\Gamma \vdash_{GD} x : G'$ is derivable in $\mathcal{U}$S'. In case (i') we are done. In case (ii'), since we have proofs of $(1')\Gamma \vdash x_0 \xrightarrow{A_1} x_1, ..., (k')\Gamma \vdash x_{k-1} \xrightarrow{A_k} x$, we can conclude by an application of ($\mathcal{U}$ **prop**) on these ones and on (ii').

– $\Gamma \vdash x \xrightarrow{A} y$, with $x \neq y$: in (Olivetti *et al.*, 2007), section 5, it is shown that a transition formula $x \xrightarrow{A} y$, with $x \neq y$, in the right-hand side of a sequent can only be proved by an application of (EQ) as follows:

$$\frac{u : A \vdash u : A' \qquad u : A' \vdash u : A}{\Gamma \vdash x \xrightarrow{A} y}\,(EQ)$$

if there is $x \xrightarrow{A'} y \in \Gamma$. In the fragment of the language we are considering, transition formulas have the form $x \xrightarrow{A} y$, where A is either an atomic formula or a combination of conjunctions and disjunctions of atomic formulas. Since the rules ($\wedge$ L) and ($\vee$ L) are both height-preserving invertible in SeqS', we can consider a proof of at most the same height of $u : A \vdash u : A'$ ended (looking forward) by several applications of ($\wedge$ L) and ($\vee$ L), according to each formula in A. The same for the proof of $u : A' \vdash u : A$. As an example, let A be $Q_1 \wedge (Q_2 \vee Q_3)$, and let A' be $P_1 \vee ((P_2 \vee P_3) \wedge P_4)$. Let $\Gamma = \Gamma', x \xrightarrow{A'} y$. We can consider a proof tree ending as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{u : P_2, u : P_4 \vdash u : A \quad u : P_3, u : P_4 \vdash u : A}{u : P_2 \vee P_3, u : P_4 \vdash u : A}(\vee L)}{u : P_1 \vdash u : A \;\; u : (P_2 \vee P_3) \wedge P_4 \vdash u : A}(\wedge L) \quad \dfrac{\dfrac{u : Q_1, u : Q_2 \vdash u : A' \quad u : Q_1, u : Q_3 \vdash u : A'}{u : Q_1, u : Q_2 \vee Q_3 \vdash u : A'}(\vee L)}{u : Q_1 \wedge (Q_2 \vee Q_3) \vdash u : A'}(\wedge L)}{u : P_1 \vee ((P_2 \vee P_3) \wedge P_4) \vdash u : A}(\vee L)}{\Gamma', x \xrightarrow{A'} y \vdash x \xrightarrow{A} y}(EQ)$$

By inductive hypothesis, all sequents whose antecedents have been decomposed in their atomic components are also derivable in $\mathcal{U}$S'. In the above example, we have that $u : P_1 \vdash_{GD} u : A; u : P_2, u : P_4 \vdash_{GD} u : A; u : P_3, u : P_4 \vdash_{GD} u : A; u : Q_1, u : Q_2 \vdash_{GD} u : A'$; and $u : Q_1, u : Q_3 \vdash_{GD} u : A'$ are derivable in $\mathcal{U}$S'. All the databases of these sequents correspond to the elements of the sets obtained by applying Flat to $u : A$ and $u : A'$, that is to say $\mathsf{Flat}(u : A) \vdash_{GD} u : A'$ and $\mathsf{Flat}(u : A') \vdash_{GD} u : A$ are derivable, then we can conclude by an application of ($\mathcal{U}$ **trans**).

– $\Gamma \vdash x \xrightarrow{A} x$: by the reformulation of (MP) shown in Figure 5, we can assume, without loss of generality, that there is a derivation in SeqS' ending as follows:

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash x \xrightarrow{A} x}\,(MP)$$

By inductive hypothesis, we have a derivation in $\mathcal{U}$S' of $\Gamma \vdash_{GD} x : A$, then we can conclude that also $\Gamma \vdash_{GD} x \xrightarrow{A} x$ is derivable in $\mathcal{U}$S' by an application of $(\mathcal{U}$ **trans**$)_{\mathbf{MP}}$. ■

The completeness with respect to the semantics immediately follows from the fact that SeqS' calculi are complete with respect to selection function models.

5.1. *Examples in $\mathcal{U}$S'*

As a first example of usage of the goal-directed proof procedures $\mathcal{U}$S', consider the following knowledge base Γ, representing the functioning of a simple DVD recorder:

- $(1)\ x : seton \Rightarrow ((pressRecButton \Rightarrow recording) \rightarrow readyToRec)$
- $(2)\ x : seton \Rightarrow pressRecButton \Rightarrow (sourceSelected \rightarrow recording)$
- $(3)\ x : seton \Rightarrow pressRecButton \Rightarrow (\top \rightarrow sourceSelected)$

We should interpret a conditional formula $A \Rightarrow B$ as "if the current state is updated with A then B holds" or, if A were an action, as "as an effect of A, B holds", or "having performed A, B holds". In this respect, clauses in Γ can be interpreted as: (1) "having set the device on, it is ready to record whenever it starts to record after pressing the REC button"; (2) "having set the device on and then having pressed the REC button, if the registration source has been selected, then the device will start to record"; (3) "having set the device on and then having pressed the REC button, it results that the registration source has been selected (the default source)". For a broader discussion on plausible interpretations of conditionals, we remind the reader to (Schwind, 1999; Giordano *et al.*, 2004; Gabbay *et al.*, 2000a); here we just observe that they have been widely used to express update/action/causation.

We show that the goal "Having set the DVD recorder on, is it ready to record?", formalized as follows:

$$x : seton \Rightarrow readyToRec$$

derives from Γ in $\mathcal{U}$CK. The derivation is presented in Figure 6.

As another example, consider the following database Γ:

- $(1)\ x : A \Rightarrow (B \vee C) \Rightarrow (D \rightarrow E)$
- $(2)\ x : A \Rightarrow (((B \vee C) \Rightarrow E) \rightarrow F)$
- $(3)\ x : A \Rightarrow (B \vee C) \Rightarrow (B \vee G) \Rightarrow (\top \rightarrow D)$
- $(4)\ x : A \Rightarrow (B \vee C) \Rightarrow (C \vee H) \Rightarrow (\top \rightarrow D)$

In $\mathcal{U}$ID+MP one can show that the goal

$$x : A \Rightarrow F$$

can be derived by the above database. A derivation is presented in Figure 7.

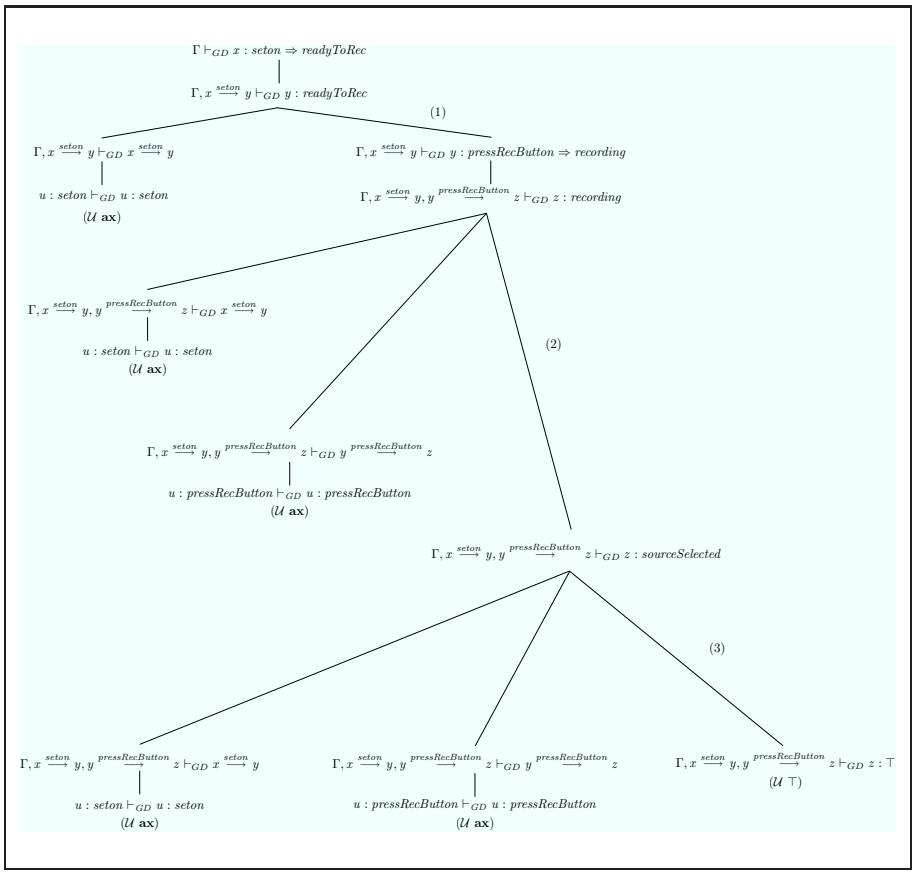

Figure 6. *A derivation of the goal $x : seton \Rightarrow readyToRec$. When* **($\mathcal{U}$ prop)** *is applied, the corresponding edge is labeled with the number of the formula of the initial database used*

6. Design of GOALD$\mathcal{U}$CK

In this section we present GOALD$\mathcal{U}$CK, a very simple implementation of the calculi $\mathcal{U}$S'. GOALD$\mathcal{U}$CK is a SICStus Prolog program consisting of only eight clauses (resp. nine clauses in systems allowing MP), each one of them implementing a rule of the calculus[7], with the addiction of some auxiliary predicates and of a predicate implementing the Flat operation. The SICStus Prolog program only consists of 52 lines of code.

Here again, the goal-directed proof search is implemented by a predicate

7. For technical reasons, GOALD$\mathcal{U}$CK split the rule ($\mathcal{U}$ **prop**) in two clauses, one taking care of applying it to the specific case of a goal $x : Q$ by using a clause $x : G \rightarrow Q$.

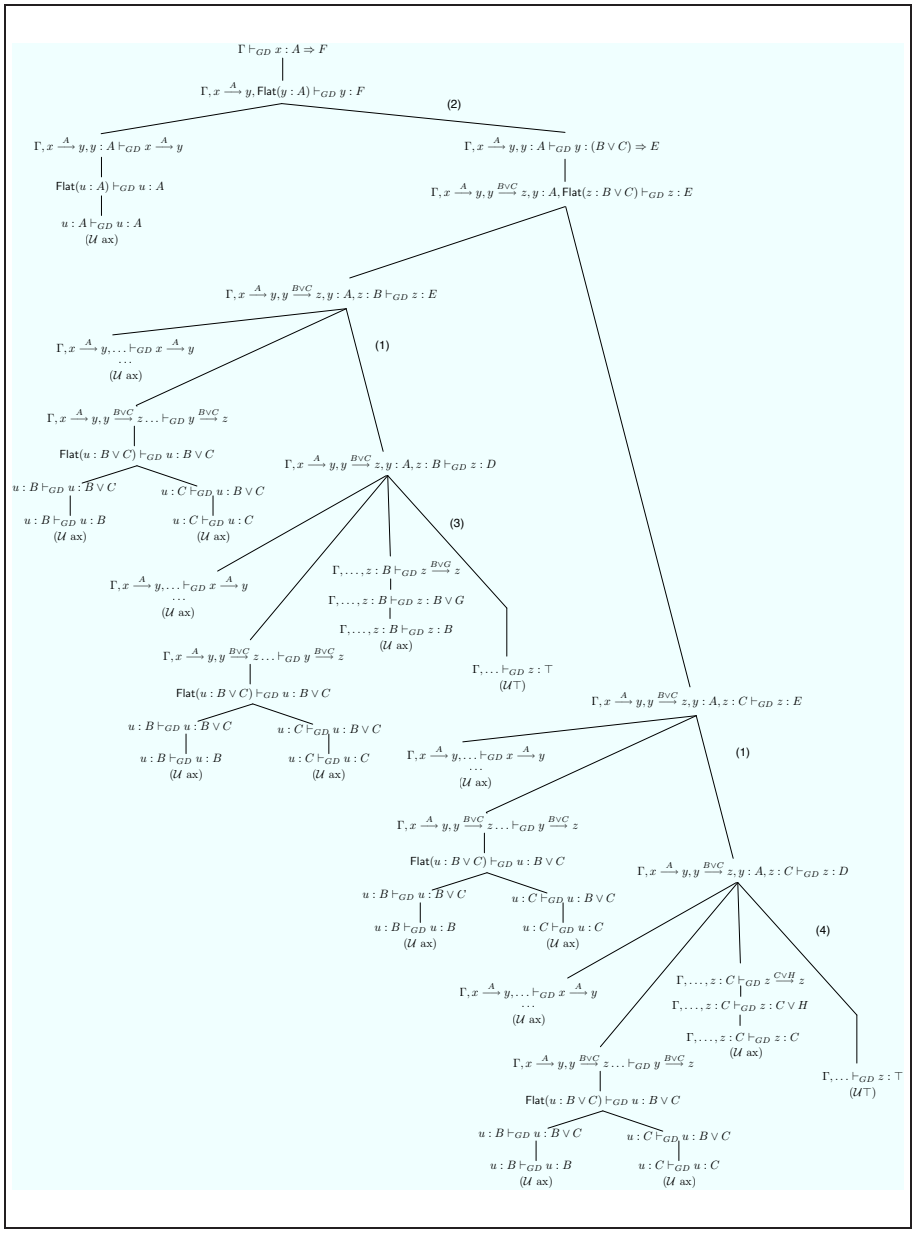

Figure 7. *A derivation in $\mathcal{U}$ID+MP of the goal $x : A \Rightarrow F$. When* **($\mathcal{U}$ prop)** *is applied, the corresponding edge is labeled with the number of the formula of the initial database used.*

prove(Goal,Gamma,Trans,Labels).

which succeeds if and only if the goal G, represented by `Goal`, derives from the knowledge base Γ, represented by the list of world formulas `Gamma` and by the list of transition formulas `Trans`. Labeled formulas (both world and transition formulas) are represented by Prolog lists `[X,A]` and `[X,A,Y]`, respectively, exactly as in CondLean. `Labels` is the list of labels occurring in the current branch of the proof.

For instance, to prove if $x : A \Rightarrow B$ derives from the database $x : A \Rightarrow C \Rightarrow (\top \rightarrow B), x : B \Rightarrow ((A \Rightarrow B) \rightarrow C)$, one queries GOALD$\mathcal{U}$CK with the following goal:

```
prove([x,a => b],[[x, a => c => (true -> b)],
      [x,b => ((a => b) -> c)]],[],[x]).
```

As for CondLean (see the Appendix for details), the predicate `prove` is also equipped with an additional argument `Tree`, instantiated by a functor `tree` and used to store the steps of a derivation and then give a justification of the proof. As we will discuss below, GOALD$\mathcal{U}$CK naturally implements a free variable version, by the presence of the clause implementing ($\mathcal{U}$ **prop**).

As mentioned above, each clause of the predicate `prove` implements an axiom or rule of the calculus $\mathcal{U}$S'. Here below we present the clauses implementing $(\mathcal{U} \Rightarrow)_{\mathbf{CK}}$, ($\mathcal{U}$ **prop**), and ($\mathcal{U}$ **trans**) as examples:

prove([X,A => G],Gamma,Trans,Labels):-

```
generateLabel(Y,Labels),
prove([Y,G],Gamma,[[X,A,Y]|Trans],[Y|Labels]).
```

prove([X,Q],Gamma,Trans,Labels):-

```
member([Y,F=>(G->Q)],Gamma),
atom(Q),
prove([X,G],Gamma,Trans,Labels),
extract(F,List),
proveList(X,Y,List,Gamma,Trans,Labels).
```

prove([X,A,Y],_,Trans,_):-

```
member([X,AP,Y],Trans),
flat(A,FlattenedA),
flat(AP,FlattenedAP),
proveFlat([x,A],[],[],[x],x,FlattenedAP),
proveFlat([x,AP],[],[],[x],x,FlattenedA).
```

The clause implementing $(\mathcal{U} \Rightarrow)_{\mathbf{CK}}$ is very intuitive: if $A \Rightarrow G$ is the current goal, then the `generateLabel` predicate introduces a new label `Y`, then the predicate `prove` is called to prove the goal $y : G$ from a knowledge base enriched by the transition $x \xrightarrow{A} y$. Obviously, in the versions of GOALD$\mathcal{U}$CK supporting CK+ID{+MP}, *i.e.* implementing the goal-directed calculi $\mathcal{U}$ID{+MP}, this clause is replaced by the one implementing the rule $(\mathcal{U} \Rightarrow)_{\mathbf{ID}}$.
The clause implementing ($\mathcal{U}$ **prop**) proceeds as follows: first, it searches for a clause $y : F \Rightarrow (G \rightarrow Q)$ in the database Γ, then it checks if Q is an atom, *i.e.* if $Q \in ATM$; second, it makes a recursive call to `prove` in order to find a derivation for the goal $x : G$; finally, it invokes two auxiliary predicates, `extract` and `proveList`, having the following functions:

– `extract` builds a list of the form $[A_1, A_2, \ldots, A_n]$, where $F = A_1 \Rightarrow A_2 \Rightarrow \ldots \Rightarrow A_n$;

– `proveList` invokes recursively the predicate `prove` in order to find a uniform proof for each goal $x_i \xrightarrow{A_{i+1}} x_{i+1}$ such that A_{i+1} belongs to the list generated by `extract`. The clauses implementing this predicate is presented here below:

```
proveList(K,Y,[A],Gamma,Trans,Labels,SubTree):-
   prove([Y,A,K],Gamma,Trans,Labels,SubTree).
proveList(X,Y,[A|Tail],Gamma,Trans,Labels,[SubTree|SubList]):-
   prove([K,A,X],Gamma,Trans,Labels,SubTree),
   proveList(K,Y,Tail,Gamma,Trans,Labels,SubList).
```

In order to prove a goal $x : Q$, if a clause $y : A_1 \Rightarrow A_2 \Rightarrow \ldots \Rightarrow A_n \Rightarrow (G \rightarrow Q)$ belongs to the database, then the predicate `extract` builds the list $[A_n, \ldots, A_2, A_1]$, and the predicate `proveList` is invoked. The first three arguments of this predicate represent the label of the current goal x, the label y of the clause of the database and the list built by `extract`, respectively. We can observe that the clause ($\mathcal{U}$ **prop**) is implemented using free variables for the labels. Thus, the implementation follows the "free-variables" style as the free variables version of CondLean. This is the most natural choice for a goal-directed procedure. Indeed, when `proveList` is applied, it first invokes `prove` in order to accomplish the goal $K_1 \xrightarrow{A_n} x$, where K_1 is a free variable, which will be instantiated by the Prolog's pattern matching to apply ($\mathcal{U}$ **trans**). Furthermore, `proveList` is recursively applied to invoke `prove` on a goal $K_2 \xrightarrow{A_{n-1}} K_1$, where K_2 is also a free variable, and so on, until a goal $y \xrightarrow{A_1} K_n$ is proved.

Given a goal `[X,A,Y]`, the clause implementing ($\mathcal{U}$ **trans**) is invoked. It first searches for a transition formula `[X,AP,Y]` in the database (*i.e.* in the list `Trans`), then it calls the predicate `flat` on both formulas `A` and `AP`; this predicate implements the Flat operation, building a *list of databases* obtained by flattening `A` (resp. `AP`) as defined in Definition 9.
In order to prove $\mathsf{Flat}(u : A) \vdash_{GD} u : AP$ and $\mathsf{Flat}(u : AP) \vdash_{GD} u : A$, another auxiliary predicate, called `proveFlat`, is finally invoked by the clause implementing

($\mathcal{U}$ **prop**); `proveFlat` recursively invokes the predicate `prove` by using *all different databases* built by `flat`.

The performance of GOALD$\mathcal{U}$CK are also promising. In the next section we present some experimental results, and we also compare GOALD$\mathcal{U}$CK's performance with CondLean's.

7. Statistics

7.1. *CondLean*

The performance of CondLean are promising. We have tested it running SICStus Prolog 3.12.0 on an AMD Atholn XP 2400+ 512MB RAM machine, obtaining the following results: in less than 2 seconds, the constant labels version succeeds in 79 tests over 90, whereas the free-variables one in 73 (but 67 in less than 10 mseconds). The test samples have been generated by modifying the samples from (Beckert *et al.*, 1997) and (Viganò, 2000). Considering the sequent-degree (defined as the maximum level of nesting of the $\Rightarrow$ operator) as a parameter, the free-variables version succeeds in less than 1 second for sequents of degree 11 and in less than 2 seconds for sequents of degree 15.

As described in section 4.3, we have also made a comparison between CondLean and leanCK, a theorem prover for conditional logic CK following the "lean" style in a more rigorous manner. We have observed that CondLean offers better performance than leanCK's; intuitively, this can be explained by the fact that CondLean makes use of auxiliary predicates such as `member` and `select`, and partitions formulas in complex, atomic, and transition formulas: this allows it to have a better control on the evolution of a derivation, resulting in a more efficient theorem prover. A "*really-lean*" implementation does not seem to be the best choice for implementing labeled calculi like SeqS.

The table below shows the number of proofs successfully completed in searching a derivation of *valid* sequents with respect to a fixed time limit. We have tested all implemented versions for CK over other 90 test samples.

	Time to succeed				
Implementation	$1ms$	$100ms$	$1s$	$2s$	$5s$
CondLean - constant labels version	80	81	83	84	86
CondLean - const. labels - ($\Rightarrow$ L) not invertible	69	71	73	73	73
CondLean - free variable version	76	76	78	79	80
leanCK	68	71	71	71	72

We have also tested CondLean over 100 sequents *not valid* in CK. CondLean concludes its proof search, answering that the sequent is not valid, in less than 2 seconds

for 92 sequents over 100; for 90 of them, CondLean answers in less than 200 mseconds, whereas it requires only 100 mseconds for concluding proofs for 88 ones.

From the above table, it seems to be reasonable to conclude that the constant labels version offers better performance. This result could be a direct consequence of the structures of the tested sequents. However, we believe that the free variables version also deserves some interest; indeed, as described above in the paper, it offers better performance on sequents of high degree or presenting a high number of different labels in the antecedent (see for instance the example mentioned in section 4.2).

7.2. GOALD$\mathcal{U}$*CK*

The performance of GOALD$\mathcal{U}$CK are also promising. We have implemented a SICStus Prolog program testing both GOALD$\mathcal{U}$CK and CondLean, versions for CK, in order to compare their performance. This program randomly generates a database containing a specific set of formulas, obtained by combining a fixed set of propositional variables ATM. Moreover, the program builds a set of goals, whose cardinality is specified as a parameter, that can be either derivable or not from the generated database. Each goal is obtained from a set ATM° of variables which is a *subset* of the set ATM of variables in the database, in order to study situations in which several formulas in the database are *useless* to prove a goal.

Here below we present the experimental results we have obtained. Given a database, a set of goals to prove, and a fixed time limit, we present the following information for both GOALD$\mathcal{U}$CK and CondLean:

– the number of executions ended with a success, that is to say: the goal is derivable from the database, and the theorem prover answers positively within the fixed time limit (column “positive answers”);

– the number of executions ended with a finite failure, that is to say: the goal is *not* derivable from the database, and the theorem prover answers negatively within the fixed time limit (column “negative answers”);

– the number of executions ended with a time out, that is to say the theorem prover is not able to conclude its work within the time limit (column “timeouts”).

Test A

– 100 goals
– Number of propositional variables ATM in the database: 9
– Number of propositional variables $ATM^{\circ} \subseteq ATM$ in the goals: 3
– Database size: 100 formulas
– Time limit: 1 ms

Theorem prover	**positive answers**	**negative answers**	**timeouts**
CondLean	72	0	28
GOALD$\mathcal{U}$CK	81	16	3

Test B

- 100 goals
- Number of propositional variables ATM in the database: 9
- Number of propositional variables $ATM^{\circ} \subseteq ATM$ in the goals: 2
- Database size: 200 formulas
- Time limit: 100 ms

Theorem prover	positive answers	negative answers	timeouts
CondLean	68	4	28
GOALD$\mathcal{U}$CK	65	11	24

Test C

- 100 goals
- Number of propositional variables ATM in the database: 20
- Number of propositional variables $ATM^{\circ} \subseteq ATM$ in the goals: 3
- Database size: 200 formulas
- Time limit: 100 ms

Theorem prover	positive answers	negative answers	timeouts
CondLean	47	0	53
GOALD$\mathcal{U}$CK	67	16	17

As expected, the above tests suggest that GOALD$\mathcal{U}$CK offers better performance when the database contains several clauses which are useless to derive a goal (as mentioned, this situation is implemented by imposing that the set of variables occurring in the goals is a subset of the set of variables in the database). In Test A, GOALD$\mathcal{U}$CK is able to prove 81 goals, whereas CondLean answers positively in 72 cases. Moreover, GOALD$\mathcal{U}$CK concludes its work in 97 cases over 100 within the fixed time limit of 1 ms, even with a finite failure in 16 cases, whereas CondLean results in a time out in 28 cases.
In Test B, CondLean seems to offer better performance in discovering derivable goals: it succeeds in 68 cases, against 65 of GOALD$\mathcal{U}$CK. Surprisingly enough, in this case GOALD$\mathcal{U}$CK runs faster in answering negatively, finding 11 not derivable goals, against 4 by CondLean.
In Test C, we have tested the theorem provers with a database containing 200 formulas and 20 propositional variables, whereas goals are built from a subset of ATM of cardinality 3. In this case, GOALD$\mathcal{U}$CK offers better performance, proving 20 goals more than CondLean in 100 ms.

We conclude this section by remarking that goal-directed proof methods usually do not ensure a terminating proof search. GOALD$\mathcal{U}$CK does not ensure termination too. Indeed, given a database Γ and a goal $x : G$, it could happen that, after several steps, the goal-driven computation leads to search a derivation for the same goal $x : G$

from a database Γ' such that $\Gamma' \supseteq \Gamma$ (that is to say: Γ' either corresponds to Γ or it is obtained from Γ by adding new facts). This problem is also well known in standard logic programming. As an example, consider the database containing the following fact:

$$x : (Q \wedge \top) \rightarrow Q$$

Querying GOALD$\mathcal{U}$CK with the goal $x : Q$, one can observe that the computation does not terminate: GOALD$\mathcal{U}$CK tries to apply ($\mathcal{U}$ **prop**) by using the only clause of the program (database), then it searches a derivation of the two subgoals $x : \top$ (and the computation succeeds) and $x : Q$. In order to prove this goal, GOALD$\mathcal{U}$CK repeats the above steps, then incurring in a loop.

This justifies the fact that, in the statistics above, GOALD$\mathcal{U}$CK produces several time outs, especially when the database contains a large number of clauses, increasing the probability of having a loop.

8. Conclusions and comparison with other works

This work extends the results presented in (Olivetti *et al.*, 2007), where a sequent calculus SeqS for standard conditional logics is introduced. We have provided extensions in two different directions: 1. efficient theorem proving for conditional logics; 2. goal-directed proof procedures for conditional logics.

Concerning the direction 1., we have presented CondLean, a theorem prover for some standard conditional logics, namely the basic system CK and its extensions with ID, MP, CS, and CEM. CondLean also supports all the combinations of the mentioned axioms, with the exception of those combining both CEM and MP. CondLean is a SICStus Prolog implementation of labeled sequent calculi SeqS for propositional conditional logics introduced in (Olivetti *et al.*, 2007). It is inspired to the "lean" methodology of leanT^{A}P, even if it does not follow its style in a very rigorous manner. For each conditional logic supported, CondLean offers two different implementations: a constant labels version, where Prolog constants are used to represent SeqS's labels, and a free variables versions, inspired to the free variable tableaux introduced in (Beckert *et al.*, 1997), where labels are also represented by Prolog variables.

Concerning the direction 2., we have extended the goal-directed calculus $\mathcal{U}$CK, already introduced in (Olivetti *et al.*, 2007) and restricted to a fragment of CK, to support the conditional logics CK+ID, CK+MP, and CK+ID+MP. We have provided goal-directed calculi called $\mathcal{U}$S', and we have shown that they are sound and complete with respect to the semantics, if the language considered is restricted to a specific fragment allowing uniform proofs. Moreover, we have presented GOALD$\mathcal{U}$CK, a SICStus Prolog implementation of the calculi $\mathcal{U}$S'. GOALD$\mathcal{U}$CK is a simple program, consisting of only seven clauses. Each clause of GOALD$\mathcal{U}$CK implements an axiom or rule of the calculi $\mathcal{U}$S'.

To the best of our knowledge, CondLean and GOALD$\mathcal{U}$CK are the first theorem provers for the basic normal conditional logic CK and for the mentioned extensions with MP, ID, CS, and CEM. Most of the works in the literature have concentrated on extensions of CK and do not present implementations of the deductive systems introduced.

De Swart (de Swart, 1983) and Gent (Gent, 1992) give sequent/tableau calculi for the strong conditional logics VC and VCS. Their proof systems are based on the entrenchment connective $\leq$, from which the conditional operator can be defined.

Crocco and Fariñas (Crocco *et al.*, 1995) give sequent calculi for some conditional logics including CK, CEM, CO and others. Their calculi comprise two levels of sequents: principal sequents with $\vdash_P$ correspond to the basic deduction relation, whereas auxiliary sequents with $\vdash_a$ correspond to the conditional operator: thus the constituents of $\Gamma \vdash_P \Delta$ are sequents of the form $X \vdash_a Y$, where X, Y are sets of formulas.

Artosi, Governatori, and Rotolo (Artosi *et al.*, 2002) develop labeled tableau for the *first-degree* fragment (*i.e.* without nested conditionals) of the conditional logic CU that corresponds to cumulative non-monotonic logics. In their work they use labels similarly to SeqS. Formulas are labeled by path of worlds containing also variable worlds (see also our free-variable implementation).

Lamarre (Lamarre, 1993) presents tableau systems for the conditional logics V, VN, VC, and VW. Lamarre's method is a consistency-checking procedure which tries to build a system of sphere falsifying the input formulas. The method makes use of a subroutine to compute the *core*, that is defined as the set of formulas characterizing the minimal sphere. The computation of the core needs in turn the consistency checking procedure. Thus there is a mutual recursive definition between the procedure for checking consistency and the procedure to compute the core.

Groeneboer and Delgrande (Delgrande *et al.*, 1990) have developed a tableau method for the conditional logic VN which is based on the translation of this logic into the modal logic S4.3.

In (Giordano *et al.*, 2003) and (Giordano *et al.*, 2005c) a labeled tableau calculus for the logic CE and some of its extensions is presented. The flat fragment of CE corresponds to the nonmonotonic preferential logic P and admits a semantics in terms of preferential structures (possible worlds together with a family of preference relations). The tableau calculus makes use of pseudo-formulas, that are modalities in a hybrid language indexed on worlds.

In (Giordano *et al.*, 2005b; Giordano *et al.*, 2006) tableau calculi for nonmonotonic KLM logics are presented. These calculi, called $\mathcal{T}S^T$, are obtained by introducing suitable modalities to interpret conditional assertions. Moreover, these calculi have been implemented in SICStus Prolog: the program, called KLMLean, is also inspired by the "*lean*" methodology and follows the style of CondLean (Olivetti *et al.*, 2005b).

In future research, we intend to extend CondLean to other systems of conditional logics. We also intend to address the problem of extending CondLean to the first order case.

We aim to develop goal-directed calculi for all conditional logics we have considered, in order to extend GOALD$\mathcal{U}$CK to support all of them. The difficulty in extending the calculi $\mathcal{U}$CK to the other extensions of CK mentioned in this work, namely CS and CEM, is that the disjunction property and the restriction on the application of the ($\Rightarrow$ L) rule, which are crucial properties in order to define a goal-directed calculus, do not hold for these systems. One possibility to extend the calculus to these systems is to adopt "restart" rules as in the calculi introduced in (Gabbay *et al.*, 2000b). We also intend to investigate a broader language allowing uniform proofs.

Finally, we intend to evaluate the performance of our theorem provers by executing some other tests. As a consequence, in our future research we plan to increase the performance of both theorem provers by experimenting standard refinements and heuristics.

Acknowledgements

We are grateful to the anonymous referee for his careful reading, stimulating suggestions, and constructive criticisms, which greatly helped us to improve our work. This research has been partially supported by the projects "*MIUR PRIN05: Specification and verification of agent interaction protocols*" and "*GALILEO 2006: Interazione e coordinazione nei sistemi multi-agenti*".

9. References

Artosi A., Governatori G., Rotolo A., "Labelled Tableaux for Non-monotonic Reasoning: Cumulative Consequence Relations", *Journal of Logic and Computation*, vol. 12, num. 6, pp. 1027-1060, 2002.

Beckert B., Goré R., "Free Variable Tableaux for Propositional Modal Logics", *in* D. Galmiche (ed.), *Prooceedings of TABLEAUX 1997 (Automated Reasoning with Analytic Tableaux and Related Methods)*, vol. 1227 of *LNAI*, Springer-Verlag, Pont-á-Mousson, France, pp. 91-106, May, 1997.

Beckert B., Posegga J., "leanTAP: Lean Tableau-based Deduction", *Journal of Automated Reasoning*, vol. 15, num. 3, pp. 339-358, 1995.

Chellas B. F., "Basic Conditional logics", *Journal of Philosophical Logic*, vol. 4, pp. 133-153, 1975.

Costello T., McCarthy J., "Useful Counterfactuals", *ETAI (Electronic Transactions on Artificial Intelligence)*, vol. 3, pp. Section A, 1999.

Crocco G., del Cerro L. F., "Structure, Consequence Relation and Logic", *in* D. M. Gabbay (ed.), *What is a Logical System*, vol. 4, Oxford University Press, pp. 239-259, 1995.

Crocco G., Lamarre P., "On the Connection between Non-Monotonic Inference Systems and Conditional Logics", *in* B. Nebel, C. Rich, W. R. Swartout (eds), *Proceedings of KR'92 (Principles of Knowledge Representation and Reasoning)*, Morgan Kaufmann, Cambridge, MA, pp. 565-571, October, 1992.

de Swart H. C. M., "A Gentzen-or Beth-type System, a Practical Decision Procedure and a Constructive Completeness Proof for the Counterfactual Logics VC and VCS", *Journal of Symbolic Logic*, vol. 48, num. 1, pp. 1-20, 1983.

Delgrande J. P., "A First-order Conditional Logic for Prototypical Properties", *Artificial Intelligence*, vol. 33, num. 1, pp. 105-130, 1987.

Delgrande J. P., Groeneboer C., "A General Approach for Determining the Validity of Commonsense Assertions Using Conditional Logics", *International Journal of Intelligent Systems*, vol. 5, num. 5, pp. 505-520, 1990.

Fitting M., "leanTAP Revisited", *Journal of Logic and Computation*, vol. 8, num. 1, pp. 33-47, 1998.

Friedman N., Halpern J. Y., "Plausibility measures and default reasoning", *Journal of the ACM*, vol. 48, num. 4, pp. 648–685, 2001.

Gabbay D. M., *Labelled Deductive Systems (vol I)*, Oxford Logic Guides, Oxford University Press, 1996.

Gabbay D. M., Giordano L., Martelli A., Olivetti N., Sapino M. L., "Conditional Reasoning in Logic Programming", *Journal of Logic Programming*, vol. 44, num. 1-3, pp. 37-74, 2000a.

Gabbay D. M., Olivetti N., *Goal-directed Proof Theory*, Kluwer Academic Publishers, 2000b.

Gent I. P., "A Sequent or Tableaux-style System for Lewis's Counterfactual Logic VC", *Notre Dame Journal of Formal Logic*, vol. 33, num. 3, pp. 369-382, 1992.

Giordano L., Gliozzi V., Olivetti N., "Iterated Belief Revision and Conditional Logic", *Studia Logica*, vol. 70, num. 1, pp. 23-47, 2002.

Giordano L., Gliozzi V., Olivetti N., "Weak AGM Postulates and Strong Ramsey Test: a logical formalization", *Artificial Intelligence*, vol. 168, num. 1-2, pp. 1-37, 2005a.

Giordano L., Gliozzi V., Olivetti N., Pozzato G. L., "Analytic Tableaux for KLM Preferential and Cumulative Logics", *in* G. Sutcliffe, A. Voronkov (eds), *Prooceedings of LPAR 2005 (12th Conference on Logic for Programming, Artificial Intelligence, and Reasoning)*, vol. 3835 of *LNAI*, Springer-Verlag, Montego Bay, Jamaica, pp. 666-6681, December, 2005b.

Giordano L., Gliozzi V., Olivetti N., Pozzato G. L., "Analytic Tableaux Calculi for KLM Rational Logic R", *in* M. Fisher, W. van der Hoek, B. Konev, A. Lisitsa (eds), *Proceedings of JELIA 2006 (10th European Conference on Logics in Artificial Intelligence)*, vol. 4160 of *LNAI*, Springer-Verlag, Liverpool, England, pp. 190-202, September, 2006.

Giordano L., Gliozzi V., Olivetti N., Schwind C., "Tableau Calculi for Preference-Based Conditional Logics", *in* C. Meyer, Marta, F. Pirri (eds), *Proceedings of TABLEAUX 2003 (Automated Reasoning with Analytic Tableaux and Related Methods)*, vol. 2796 of *LNAI*, Springer, Roma, Italy, pp. 81–101, September, 2003.

Giordano L., Gliozzi V., Olivetti N., Schwind C., "Extensions of Tableau Calculi for Preference-based Conditional Logics", *in* H. Schlingloff (ed.), *Proceedings of the 4th International*

Workshop on Methods for Modalities (M4M-4), Informatik-Bericht 194, Fraunhofer Institute FIRST, Berlin, Germany, pp. 220-234, December, 2005c.

Giordano L., Schwind C., "Conditional Logic of Actions and Causation", *Artificial Intelligence*, vol. 157, num. 1-2, pp. 239-279, 2004.

Grahne G., "Updates and Counterfactuals", *Journal of Logic and Computation*, vol. 8, num. 1, pp. 87-117, 1998.

Kraus S., Lehmann D., Magidor M., "Nonmonotonic Reasoning, Preferential Models and Cumulative Logics", *Artificial Intelligence*, vol. 44, num. 1-2, pp. 167-207, 1990.

Lamarre P., "A Tableaux Prover for Conditional Logics", *Proceedings of KR'93 (Principles of Knowledge Representation and Reasoning)*, pp. 572-580, 1993.

Lewis D., *Counterfactuals*, Basil Blackwell Ltd, 1973.

Miller D., Nadathur G., Pfenning F., Scedrov A., "Uniform Proofs as a Foundation for Logic Programming", *In Annals of Pure and Applied Logic*, vol. 51, num. 1-2, pp. 125-157, 1991.

Nute D., *Topics in Conditional Logic*, Reidel, Dordrecht, 1980.

Olivetti N., Pozzato G. L., "CondLean: A Theorem Prover for Conditional Logics", *in* M. Cialdea Meyer, F. Pirri (eds), *Proceedings of TABLEAUX 2003 (Automated Reasoning with Analytic Tableaux and Related Methods)*, vol. 2796 of *LNAI*, Springer, Roma, Italy, pp. 264–270, September, 2003.

Olivetti N., Pozzato G. L., "CondLean 3.0: Improving CondLean for Stronger Conditional Logics", *in* B. Beckert (ed.), *Proceedings of TABLEAUX 2005 (Automated Reasoning with Analytic Tableaux and Related Methods)*, vol. 3702 of *LNAI*, Springer-Verlag, Koblenz, Germany, pp. 328–332, September, 2005a.

Olivetti N., Pozzato G. L., "KLMLean 1.0: a Theorem Prover for Logics of Default Reasoning", *in* H. Schlingloff (ed.), *Proceedings of the 4th International Workshop on Methods for Modalities (M4M-4)*, Informatik-Bericht 194, Fraunhofer Institute FIRST, Berlin, Germany, pp. 235-245, December, 2005b.

Olivetti N., Pozzato G. L., Schwind C. B., "A Sequent Calculus and a Theorem Prover for Standard Conditional Logics", *ACM Transactions on Computational Logics (TOCL)*, 2007.

Schwind C. B., "Causality in Action Theories", *Electronic Transactions on Artificial Intelligence (ETAI)*, vol. 3, pp. 27–50, 1999.

Smullyan R., *First-Order Logic*, Springer-Verlag, Berlin, 1968.

Stalnaker R., "A Theory of Conditionals", *in* N. Rescher (ed.), *Studies in Logical Theory*, vol. 2 of *Monograph Series, American Philosophical Quarterly*, Blackwell, Oxford, pp. 98-112, 1968.

Viganò L., *Labelled Non-classical Logics*, Kluwer Academic Publishers, Dordrecht, 2000.

A. CondLean's graphical user interface

The program CondLean has also a graphical interface (GUI) implemented in Java. The GUI interacts with the SICStus Prolog implementation by means of the package

`se.sics.jasper`. Thanks to the GUI, one does not need to know how to call the predicate `prove`: one just introduces a sequent in a text box and searches a derivation by clicking a button. Moreover, one can choose the intended system of conditional logic. Some pictures of CondLean are presented in Figures 8, 9, and 10.

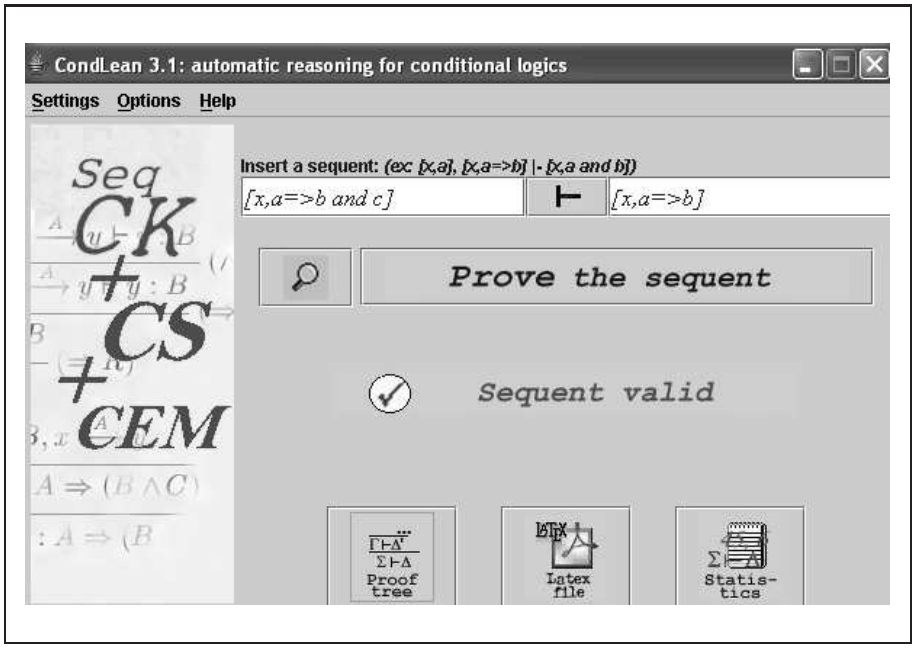

Figure 8. *The control window of CondLean*

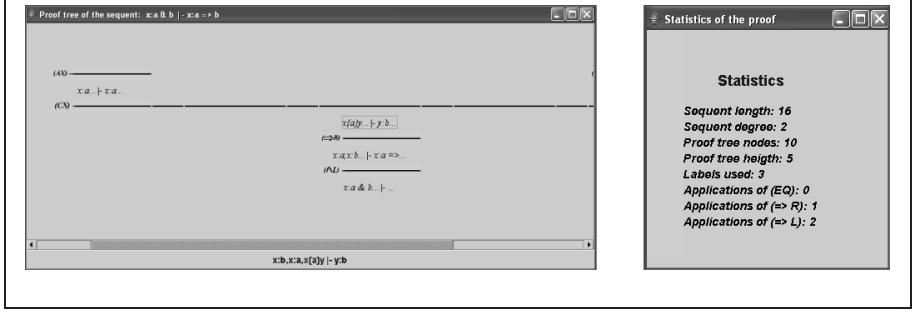

Figure 9. *When the sequent is valid, the derivation found by the SICStus Prolog engine can be displayed in a special window. Users can also view some statistics of the proof*

The predicate `prove`, implementing SeqS calculi, is equipped by an additional argument, called `ProofTree`, which is used by the SICStus Prolog implementation in order to *return* the proof tree found back to the Java interface. When the user

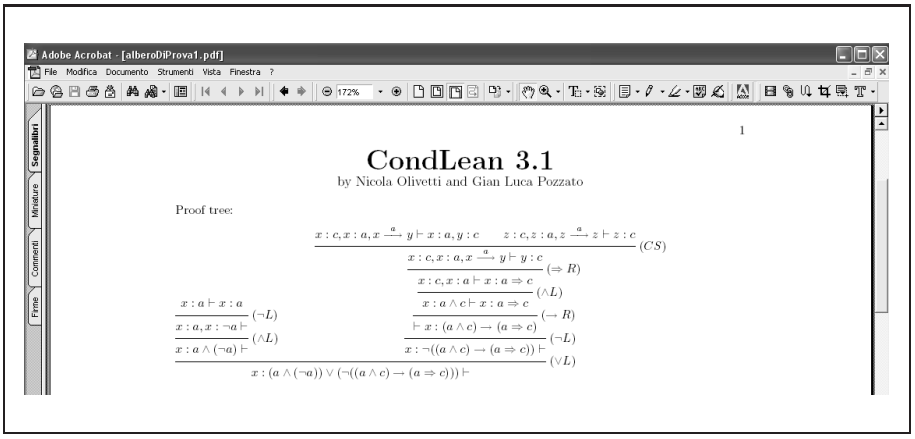

Figure 10. *CondLean also builds a* LaTeX *source file containing the derivation found*

asks CondLean to prove a sequent by clicking the button, then the Prolog engine starts its work and, if the sequent is valid, it returns a functor `tree`, matching with the argument `ProofTree` and representing the derivation found. The functor `tree` is then manipulated by the Java interface in order to display the derivation to the user.

When the submitted sequent is valid, CondLean offers these options:

– display a proof tree of the sequent in a special window;
– build a LaTeX source file containing the same proof tree: one can then compile it with a LaTeX engine and obtain a printable version of the tree (in a .dvi or .ps or .pdf file);
– view some statistics of the proof.