An Implementation of a Free-variable Tableaux for KLM Preferential Logic P of Nonmonotonic Reasoning: the Theorem Prover FREEP 1.0

Laura Giordano[1], Valentina Gliozzi[2], Nicola Olivetti[3], and Gian Luca Pozzato[2]

[1] Dip. di Informatica - Univ. Piemonte O. "A. Avogadro" - laura@mfn.unipmn.it
[2] Dip. di Informatica - Università di Torino - {gliozzi,pozzato}@di.unito.it
[3] LSIS-UMR CNRS 6168 Univ. "P. Cézanne" - nicola.olivetti@univ-cezanne.fr

Abstract. In this work we present FREEP 1.0, a theorem prover for the KLM preferential logic **P** of nonmonotonic reasoning. FREEP 1.0 is a SICStus Prolog implementation of a *free-variables*, labelled tableau calculus for **P**, obtained by introducing suitable modalities to interpret conditional assertions. The performances of FREEP 1.0 are promising. FREEP 1.0 can be downloaded at http://www.di.unito.it/~pozzato/FREEP 1.0.

1 Introduction

The work of Kraus, Lehmann and Magidor (from now on KLM) at the beginning of the 90s [1] is a milestone in the area of nonmonotonic reasoning. The postulates outlined by KLM have been widely accepted as the "conservative core" of nonmonotonic reasoning: they correspond to properties that any concrete reasoning mechanism should satisfy. As shown by Halpern and Friedman [2], many different approaches to nonmonotonic reasoning are characterized by the properties of two of the KLM logics, namely preferential logic **P** and rational logic **R**.

According to the KLM framework, a defeasible knowledge base is represented by a (finite) set of nonmonotonic conditionals or assertions of the form $A \mathrel{|\!\sim} B$, whose reading is *normally (or typically) the A's are B's*. The operator "$\mathrel{|\!\sim}$" is nonmonotonic, in the sense that $A \mathrel{|\!\sim} B$ does not imply $A \wedge C \mathrel{|\!\sim} B$. For instance, a knowledge base K may contain $sumo\_wrestler \mathrel{|\!\sim} fat$, $sumo\_wrestler \mathrel{|\!\sim} sumo\_lover$, $sumo\_lover \mathrel{|\!\sim} \neg fat$, whose meaning is that sumo wrestlers are typically fat, sumo wrestlers typically love sumo, but people loving sumo typically are not fat. If $\mathrel{|\!\sim}$ were interpreted as classical implication, one would get $sumo\_wrestler \mathrel{|\!\sim} \bot$, i.e. typically there are not sumo wrestlers, thereby obtaining a trivial knowledge base. One can derive new conditional assertions from the knowledge base by means of a set of inference rules, without incurring the trivializing conclusions of classical logic. In KLM framework, the set of inference rules defines some fundamental types of inference systems, namely, from the strongest to the weakest: Rational (**R**), Preferential (**P**), Loop-Cumulative (**CL**), and Cumulative (**C**) logic.

In this paper we focus on **Preferential logic P**. Concerning the above example, in preferential logic **P** one can infer $sumo\_lover \wedge sumo\_wrestler \mathrel{|\!\sim} fat$, giving preference to more specific information, and that $sumo\_lover \mathrel{|\!\sim} \neg sumo\_wrestler$. From a semantic point of view, the models of preferential logic **P** are possible-world structures equipped with a preference relation $<$ among worlds. The meaning of a conditional assertion $A \mathrel{|\!\sim} B$ is that B holds in the *most preferred* worlds where A holds.

In [3] an analytic tableau calculus for propositional **P** is introduced. The basic idea of this calculus is to interpret the preference relation $<$ as an accessibility relation: a conditional $A \mathrel{|\!\sim} B$ holds in a model if B is true in all worlds satisfying A that are minimal wrt $<$. Following this idea, the calculus provides a sort of run-time translation of **P** into Gödel-Löb modal logic of provability G. The relation with modal logic G is motivated by the fact that we assume, following KLM, the so-called *smoothness condition*, which is related to the well-known *limit assumption*. This condition ensures that minimal A-worlds exist whenever there are A-worlds, by preventing infinitely descending chains of worlds. This condition is therefore ensured by the finite-chain condition on the accessibility relation (as in modal logic G). The resulting calculus, called $\mathcal{T}\mathbf{P}^{\mathbf{T}}$, is a *non*-labelled, sound, complete and terminating proof method for the logic **P**. In [4] a SICStus Prolog implementation of $\mathcal{T}\mathbf{P}^{\mathbf{T}}$ is presented. The program, called KLMLean 2.0, is inspired to the "lean" methodology, and, as far as we know, it is the first theorem prover for the preferential logic **P**.

In this work we extend our investigation on efficient theorem proving for KLM logics by introducing FREEP 1.0, a theorem prover for preferential logic **P** that implements a *free-variable* tableau calculus for this logic. Free-variable tableaux are a well-known and well-established technique for first order theorem proving [5], whose basic idea is to adopt free-variables as a meta-linguistic device for representing all labels/worlds that can be used in a proof search. In order to reduce the search space, the instantiation of these free-variables is postponed as much as possible, until more information is available (for instance to close a branch). In recent years, Beckert and Goré have presented the theoretical foundations to extend the free-variable tableaux technique to all 15 basic propositional modal logics [6].
In our paper, we first introduce a *labelled* tableau calculus for the preferential logic **P**, based on the same translation into G adopted by $\mathcal{T}\mathbf{P}^{\mathbf{T}}$. This calculus, called $\mathcal{LAB}\mathbf{P}$, can be straightforwardly derived from the labelled calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ for the rational logic **R** introduced in [7]. We then show that $\mathcal{LAB}\mathbf{P}$ can be easily turned into a terminating calculus. Last, we present FREEP 1.0, an implementation of a free-variables version of $\mathcal{LAB}\mathbf{P}$ in SICStus Prolog. This last point is the main contribution of the paper.

2 KLM Preferential Logic P

In this section we briefly recall the axiomatization and semantics of the preferential logic **P**. For a complete description of KLM systems, see [1] and [8]. The language $\mathcal{L}$ we consider is defined from a set of propositional variables ATM, the boolean connectives and the conditional operator $\mathrel{|\!\sim}$. We use $A, B, C, ...$ to denote propositional formulas, whereas $F, G, ...$ are used to denote all formulas (even conditionals). The formulas of $\mathcal{L}$ are defined as follows: if A is a propositional formula, $A \in \mathcal{L}$; if A and B are propositional formulas, $A \mathrel{|\!\sim} B \in \mathcal{L}$; if F is a boolean combination of formulas of $\mathcal{L}$, $F \in \mathcal{L}$. $\mathcal{L}$ corresponds to the fragment of the language of conditional logics without nested occurrences of the conditional operator $\mathrel{|\!\sim}$.

The axiomatization of **P** consists of all axioms and rules of propositional calculus together with the following axioms and rules ($\vdash_{PC}$ denotes provability in the propositional calculus, whereas $\vdash$ denotes provability in **P**):

- REF. $A \mathrel{|\!\sim} A$ (reflexivity)
- LLE. If $\vdash_{PC} A \leftrightarrow B$, then $\vdash (A \mathrel{|\!\sim} C) \rightarrow (B \mathrel{|\!\sim} C)$ (left logical equivalence)
- RW. If $\vdash_{PC} A \rightarrow B$, then $\vdash (C \mathrel{|\!\sim} A) \rightarrow (C \mathrel{|\!\sim} B)$ (right weakening)
- AND. $((A \mathrel{|\!\sim} B) \wedge (A \mathrel{|\!\sim} C)) \rightarrow (A \mathrel{|\!\sim} B \wedge C)$
- CM. $((A \mathrel{|\!\sim} B) \wedge (A \mathrel{|\!\sim} C)) \rightarrow (A \wedge B \mathrel{|\!\sim} C)$ (cautious monotonicity)
- OR. $((A \mathrel{|\!\sim} C) \wedge (B \mathrel{|\!\sim} C)) \rightarrow (A \vee B \mathrel{|\!\sim} C)$

The semantics of **P** is defined by considering possible world structures with a preference relation (a strict partial order) $w < w'$ whose meaning is that w is preferred to w'. We have that $A \mathrel{|\!\sim} B$ holds in a model $\mathcal{M}$ if B holds in all *minimal A-worlds* (w.r.t. $<$). This definition makes sense provided minimal A-worlds exist (whenever there are A-worlds). This is ensured by the *smoothness condition* in the next definition.

Definition 1 (Semantics of P, Definition 16 in [1]). *A preferential model is a triple $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ where: $\mathcal{W}$ is a non-empty set of items called worlds; $<$ is an irreflexive and transitive relation on $\mathcal{W}$; V is a function $V : \mathcal{W} \longmapsto pow(ATM)$, which assigns to every world w the set of atoms holding in that world. We define the truth conditions for a formula F as follows: (i) if F is a boolean combination of formulas, $\mathcal{M}, w \models F$ is defined as for propositional logic; (ii) let A be a propositional formula; we define $Min_<(A) = \{w \in \mathcal{W} \mid \mathcal{M}, w \models A$ and $\forall w'$, $w' < w$ implies $\mathcal{M}, w' \not\models A\}$; (iii) $\mathcal{M}, w \models A \mathrel{|\!\sim} B$ if for all $w' \in \mathcal{W}$, if $w' \in Min_<(A)$ then $\mathcal{M}, w' \models B$.*
The relation $<$ satisfies the following smoothness condition*: if $\mathcal{M}, w \models A$ then $w \in Min_<(A)$ or $\exists w' \in Min_<(A)$ such that $w' < w$.*

We say that a formula F is valid in a model $\mathcal{M}$ *($\mathcal{M} \models F$), if $\mathcal{M}, w \models F$ for every $w \in \mathcal{W}$. A formula is* valid *if it is valid in every model $\mathcal{M}$.*

3 A Labelled Tableau Calculus for Preferential Logic P

In this section we present a labelled tableau calculus for preferential logic **P**. This calculus, called $\mathcal{LAB}\mathbf{P}$ from $\mathcal{LAB}$elled **P**, follows the same intuition as the calculus $\mathcal{T}\mathbf{P}^{\mathbf{T}}$ introduced in [3], i.e. to interpret the preference relation as an accessibility relation, thus implementing a sort of run-time translation into modal logic G. $\mathcal{LAB}\mathbf{P}$ is straightforwardly derived from the labelled calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ for rational logic **R** [7], by removing rule $(<)$, corresponding to modularity that differentiates **P** from **R**[1].

The calculus $\mathcal{LAB}\mathbf{P}$ makes use of labels to represent possible worlds. We consider a language $\mathcal{L}_P$ and a denumerable alphabet of labels $\mathcal{A}$, whose elements are denoted by *x, y, z, ...* $\mathcal{L}_P$ extends $\mathcal{L}$ by formulas of the form $\Box A$, where A is propositional, whose intuitive meaning is as follows: $\Box A$ holds in a world w if A holds in all the worlds w' such that $w' < w$, that is to say:

Definition 2 (Truth condition of $\Box$). *$\mathcal{M}, w \models \Box A$ if for every $w' \in \mathcal{W}$ if $w' < w$ then $\mathcal{M}, w' \models A$.*

[1] The property of modularity characterizing **R** is the following: for each $u, v, w \in \mathcal{W}$, if $u < v$ then either $u < w$ or $w < v$. This property is captured by the rule $(<)$ of $\mathcal{T}\mathbf{R}^{\mathbf{T}}$.

It can be observed that $\Box$ has (among others) the properties of the modal system G, whose characterizing axiom is $\Box(\Box A \rightarrow A) \rightarrow \Box A$. This axiom guarantees that the accessibility relation (defined as xRy if $y < x$) is transitive and does not have infinite ascending chains. From definition of $Min_<(A)$ in Definition 1, it follows that, for any formula A, $w \in Min_<(A)$ iff $\mathcal{M}, w \models A \wedge \Box\neg A$. As we will see, $\mathcal{LAB}\mathbf{P}$ only makes use of boxed formulas with a negated argument, i.e. formulas of the form $x : \Box\neg A$.

Our tableau calculus comprises two kinds of labelled formulas: (i) *world formulas* $x : F$, whose meaning is that F holds in the possible world represented by x; (ii) *relation formulas* of the form $x < y$, where $x, y \in \mathcal{A}$, used to represent the relation $<$.

We define $\Gamma^M_{x \rightarrow y} = \{y : \neg A, y : \Box\neg A \mid x : \Box\neg A \in \Gamma\}$. The calculus $\mathcal{LAB}\mathbf{P}$ is presented in Figure 1. We call *dynamic* the rules $(\mathrel{|\!\sim}^-)$ and $(\Box^-)$ that introduce new labels in their conclusion; all the other rules are called *static*.

$$(\mathbf{AX})\ \Gamma, x : P, x : \neg P \quad \text{with } P \in ATM \qquad (\neg)\frac{\Gamma, x : \neg\neg F}{\Gamma, x : F} \qquad (\mathrel{|\!\sim}^-)\frac{\Gamma, u : \neg(A \mathrel{|\!\sim} B)}{\Gamma, x : A, x : \Box\neg A, x : \neg B}\ x \text{ new label}$$

$$(\mathrel{|\!\sim}^+)\frac{\Gamma, u : A \mathrel{|\!\sim} B}{\Gamma, u : A \mathrel{|\!\sim} B, x : \neg A \qquad \Gamma, u : A \mathrel{|\!\sim} B, x : \neg\Box\neg A \qquad \Gamma, u : A \mathrel{|\!\sim} B, x : B}\ x \text{ occurs in } \Gamma$$

$$(\Box^-)\frac{\Gamma, x : \neg\Box\neg A}{\Gamma, y < x, y : A, y : \Box\neg A, \Gamma^M_{x \rightarrow y}}\ y \text{ new label} \qquad (\wedge^+)\frac{\Gamma, x : F \wedge G}{\Gamma, x : F, x : G} \qquad (\wedge^-)\frac{\Gamma, x : \neg(F \wedge G)}{\Gamma, x : \neg F \qquad \Gamma, x : \neg G}$$

Fig. 1. The calculus $\mathcal{LAB}\mathbf{P}$. To save space, rules for $\rightarrow$ and $\vee$ are omitted.

Notice that there is not a rule for positive boxed formulas, as the propagation of positive boxed formulas is performed by the set $\Gamma^M_{x \rightarrow y}$, when a new world is created by $(\Box^-)$.

Definition 3 (Truth conditions of formulas of $\mathcal{LAB}\mathbf{P}$). *Given a model $\mathcal{M} = \langle \mathcal{W}, < , V\rangle$ and a label alphabet $\mathcal{A}$, we consider a mapping $I : \mathcal{A} \mapsto \mathcal{W}$. Given a formula α of the calculus $\mathcal{LAB}\mathbf{P}$, we define $\mathcal{M} \models_I \alpha$ as follows: $\mathcal{M} \models_I x : F$ iff $\mathcal{M}, I(x) \models F$; $\mathcal{M} \models_I x < y$ iff $I(x) < I(y)$. We say that a set Γ of formulas of $\mathcal{LAB}\mathbf{P}$ is satisfiable if there exist $\mathcal{M}$ and I s.t., for all formulas $\alpha \in \Gamma$, we have that $\mathcal{M} \models_I \alpha$.*

A tableau is a tree whose nodes are sets of labelled formulas Γ. Therefore, a branch is a sequence of sets of labelled formulas $\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$ Each node Γ_i is obtained by its immediate predecessor Γ_{i-1} by applying a rule of $\mathcal{LAB}\mathbf{P}$, having Γ_{i-1} as the premise and Γ_i as one of its conclusions. A branch is closed if one of its nodes is an instance of $(\mathbf{AX})$, otherwise it is open. We say that a tableau is closed if all its branches are closed. In order to verify that a set of formulas Γ is unsatisfiable, we label all the formulas in Γ with a new label x, and verify that the resulting set of labelled formulas has a closed tableau. For instance, in order to verify that $sumo\_lover \mathrel{|\!\sim} \neg sumo\_wrestler$ can be inferred from the knowledge base of the example in the Introduction, one needs to check if there is a closed tableau for the set $\{x : sumo\_lover \mathrel{|\!\sim} \neg fat, x : sumo\_wrestler \mathrel{|\!\sim} sumo\_lover, x : sumo\_wrestler \mathrel{|\!\sim} fat, x : \neg(sumo\_lover \mathrel{|\!\sim} \neg sumo\_wrestler)\}$. A derivation is presented in Figure 2.

The calculus $\mathcal{LAB}\mathbf{P}$ is sound and complete w.r.t. the semantics. Due to space limitations, we omit the proofs, which are very similar to the ones for $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ in [7].

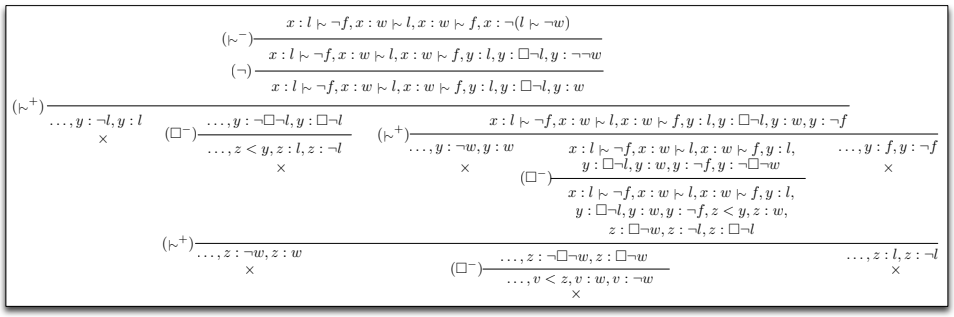

Fig. 2. A derivation in $\mathcal{LAB}\mathbf{P}$. We use l for $sumo\_lover$, w for $sumo\_wrestler$, and f for fat.

Theorem 1 (Soundness and Completeness of $\mathcal{LAB}\mathbf{P}$). *Given a set of formulas Γ, it is unsatisfiable if and only if there is a closed tableau in $\mathcal{LAB}\mathbf{P}$ starting with Γ.*

Let us now refine the calculus $\mathcal{LAB}\mathbf{P}$ in order to ensure termination. In general, non-termination in labelled tableau calculi can be caused by two different reasons: 1. dynamic rules can generate infinitely-many worlds, thus creating infinite branches; 2. some rules copy their principal formula in their conclusion(s), therefore they can be reapplied over the same formula without any control.
As far as $\mathcal{LAB}\mathbf{P}$ is concerned, we can show that the first source of non termination (point 1) cannot occur. Indeed, similarly to $\mathcal{TR}^{\mathbf{T}}$, we have that the only rules that can introduce new labels in the tableau are $(\nvdash^-)$ and $(\Box^-)$. It can be proven that there can be only finitely many applications of these rules in a proof search. Intuitively, the rule $(\nvdash^-)$ can be applied only once for each negated conditional occurring in the initial set Γ (hence it introduces only a finite number of labels). Furthermore, the generation of infinite branches due to the interplay between rules $(\nvdash^+)$ and $(\Box^-)$ cannot occur. Indeed, each application of $(\Box^-)$ to a formula $x : \neg\Box\neg A$ (introduced by $(\nvdash^+)$) adds the formula $y : \Box\neg A$ to the conclusion, so that $(\nvdash^+)$ can no longer consistently introduce $y : \neg\Box\neg A$. This is due to the properties of $\Box$, that are similar to the properties of the corresponding modality of modal system G.
Concerning point 2, the calculus $\mathcal{LAB}\mathbf{P}$ does not ensure a terminating proof search due to the $(\nvdash^+)$ rule, which can be applied without any control. It is easy to observe that it is useless to apply the rule on the same conditional formula more than once by using the same label x. Indeed, all formulas in the premise of $(\nvdash^+)$ are kept in the conclusions, then we can assume, without loss of generality, that two applications of $(\nvdash^+)$ on x are consecutive. We observe that the second application is useless, since each of the conclusions has already been obtained after the first application, and can be removed. We prevent redundant applications of $(\nvdash^+)$ by keeping track of labels in which a conditional $u : A \nvdash B$ has already been applied in the current branch. To this purpose, we add to each positive conditional a list L of *used* labels; we restrict the application of $(\nvdash^+)$ only to labels not occurring in the corresponding list. The resulting terminating calculus $\mathcal{LAB}\mathbf{P}$ is obtained by replacing rule $(\nvdash^+)$ in Figure 1 with the one presented in Figure 3. The proof of termination is similar to the one in [7] (Theorem 6).

Theorem 2 (Termination of $\mathcal{LAB}\mathbf{P}$). *Let Γ be a finite set of formulas, then any tableau generated by $\mathcal{LAB}\mathbf{P}$ starting with Γ is finite.*

$$(\mid\!\sim^{+})\ \frac{\Gamma, u : A \mid\!\sim B^{L}}{\Gamma, x : \neg A, u : A \mid\!\sim B^{L,x} \qquad \Gamma, x : \neg\Box\neg A, u : A \mid\!\sim B^{L,x} \qquad \Gamma, x : B, u : A \mid\!\sim B^{L,x}}$$

with $x \notin L$

Fig. 3. The rule $(\mid\!\sim^{+})$ in the terminating tableau calculus $\mathcal{LAB}\mathbf{P}$.

4 Design of FREEP 1.0

Here we describe FREEP 1.0, an implementation of a free-variable extension of $\mathcal{LAB}\mathbf{P}$ calculus in SICStus Prolog. For expository reasons, we proceed in two steps. First, in section 4.1 we present a simple implementation of $\mathcal{LAB}\mathbf{P}$ that does not use free-variables, yet. Second, in section 4.2 we refine it by introducing free-variables in order to increase its performances. Intuitively, this refinement consists in the fact that the $(\mid\!\sim^{+})$ rule introduces a *free-variable* in all its conclusions, rather than immediately choosing the label to use. The free-variables will then be instantiated to close a branch with an axiom, thus the choice of the label is postponed as much as possible.

4.1 A Simple Implementation of $\mathcal{LAB}\mathbf{P}$ (without free-variables)

The Prolog program consists of a set of clauses, each representing a tableau 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. We represent each node of a proof tree (i.e. set of formulas) by a Prolog list. A world formula $x : A$ is represented by a *pair* `[x,a]`, whereas a relation formula $x < y$ is represented by a triple `[x,<,y]`. The tableau calculus is implemented by the predicate

prove(Gamma,Cond,Labels,Tree).

which succeeds if and only if the set of formulas Γ, represented by the list `Gamma`, is unsatisfiable. `Cond` is a list representing the set of *used conditionals*, and it is needed in order to control the application of the rule $(\mid\!\sim^{+})$, as described in the previous section. More in detail, the elements of `Cond` are pairs of the form `[A=>B,Used]`, where `Used` is a Prolog list containing all the labels that have already been used to apply $(\mid\!\sim^{+})$ on $x : A \mid\!\sim B$ in the current branch. As we will discuss later in this section, this allows to apply this rule in a controlled way, ensuring termination of the proof search. `Labels` is the list of labels introduced in the current branch. When `prove` succeeds, `Tree` contains a representation of a closed tableau. For instance, to prove that $A \mid\!\sim B \wedge C, \neg(A \mid\!\sim C)$ is unsatisfiable in **P**, one queries FREEP 1.0 with the goal `prove([[x,a => (b and c)], [x,neg (a => c)]],[ ], [x], Tree).` The string "`=>`" is used to represent the conditional operator $\mid\!\sim$, "`and`" is used to denote $\wedge$, and so on. Each clause of `prove` implements one axiom or rule of the tableau calculus; for example, the clauses implementing $(\mathbf{AX})$ and $(\mid\!\sim^{-})$ are as follows:

```
prove(Gamma,_,_,tree(...)):-
   member([X,F],Gamma),member([X,neg F],Gamma),!.
prove(Gamma,Cond,Labels,tree(...)):-
   select([X,neg (A => B)],Gamma,NewG),!,newLabel(Labels,Y),
```

```
   gammaM(NewG,X,Y,ResGM),append(ResGM,NewG,DefG),
   prove([[Y,neg B]|[[Y,box neg A]|[[Y,A]|DefG]]],Cond,[Y|Labels],...).
```

The clause for $(\mid\!\sim^{-})$ is applied when a formula $x : \neg(A \mid\!\sim B)$ belongs to Γ. The predicate `select` removes $x : \neg(A \mid\!\sim B)$ from `Gamma`, then the auxiliary predicate `gammaM` is invoked to compute the set $\Gamma^{M}_{x \to y}$ defined in the previous section on the resulting list `NewG`; finally, the predicate `prove` is recursively invoked on the only conclusion of the rule. The predicate `newLabel` is used to generate a new label `Y` not occurring in the current branch.

To search a derivation of a set of formulas Γ, FREEP 1.0 proceeds as follows: first of all, if Γ is an instance of (**AX**), the goal will succeed immediately by using the clause for the axioms. If it is not, then the first applicable rule will be chosen, e.g. if `Gamma` contains a formula `[X,neg(neg F)]`, then the clause for $(\neg)$ rule will be used, invoking `prove` on its unique conclusion. FREEP 1.0 proceeds in a similar way for the other rules. The ordering of the clauses is such that the boolean rules are applied before the other ones. Let us now analyze the crucial rule $(\mid\!\sim^{+})$, which is implemented by the two following clauses:

```
prove(Gamma,Cond,Labels,tree(...)):-
   member([_,A=>B],Gamma),\+member([A=>B,_],Cond),!,
   member(X,Labels),!,
   prove([[X,A->B]|Gamma],[[A=>B,[X]]|Cond],Labels,...),
   prove([[X,neg box neg A]|Gamma],[[A=>B,[X]]|Cond],...).
prove(Gamma,Cond,Labels,tree(...)):-
   member([_,A=>B],Gamma),select([A=>B,Used],Cond,NewCond),
   member(X,Labels),\+member(X,Used),!,
   prove([[X,A->B]|Gamma],[[A=>B,[X|Used]]|NewCond],Labels,...),
   prove([[X,neg box neg A]|Gamma],[[A=>B,[X|Used]]|NewCond],...).
```

The above clauses are applied when a conditional formula $u : A \mid\!\sim B$ belongs to `Gamma`. If it is the first time the rule is applied on $A \mid\!\sim B$ in the current branch, i.e. there is no `[A=>B,Used]` in the list `Cond`, then the first clause is applied. It chooses a label `X` to use (predicate `member(X,Labels)`), then the predicate `prove` is recursively invoked on the conclusions of the rule. Otherwise, i.e. if $(\mid\!\sim^{+})$ has already been applied on $A \mid\!\sim B$ in the current branch, the second clause is invoked: the theorem prover chooses a label `X` in the `Labels` list, then it checks if `X` belongs to the list `Used` of labels already used to apply $(\mid\!\sim^{+})$ on $A \mid\!\sim B$ in the current branch. If not, the predicate `prove` is recursively invoked on the conclusions of the rule. Notice that the above clauses implement an alternative version of the $(\mid\!\sim^{+})$ rule, equivalent to the one presented in Figure 3. In order to build a binary tableau, the rule has two conclusions rather than three, namely the conclusions $\Gamma, u : A \mid\!\sim B, x : \neg A$ and $\Gamma, u : A \mid\!\sim B, x : B$ are replaced by the single conclusion $\Gamma, u : A \mid\!\sim B, x : A \to B$.

Even if $(\mid\!\sim^{+})$ is invertible, and it does not introduce any backtracking point in a proof search, choosing the right label in the application of $(\mid\!\sim^{+})$ is highly critical for the performances of the theorem prover. Indeed, if the current set of formulas contains n different labels, say $x_1, x_2, \ldots, x_n$, it could be the case that a single application of $(\mid\!\sim^{+})$ on $u : A \mid\!\sim B$ by using x_n leads to an immediate closure of both its conclusions, thus ending the proof. However, if the deductive mechanism chooses the labels to use

in the above order, then the right label x_n will be chosen only after $n-1$ applications of $(\sim^+)$ on each branch, thus creating a bigger tableau and, as a consequence, decreasing the performances of the theorem prover. In order to postpone this choice as much as possible, we have defined a **free-variables** version of the prover, whose basic idea is to use *Prolog variables* as *jolly* (or *wildcards*) in each application of the $(\sim^+)$ rule. The refined version of the prover is described in the following section.

4.2 Free-variables Implementation

A free-variable can be seen as a jolly representing *all the labels* that can be used in an application of $(\sim^+)$, that is to say the rule $(\sim^+)$ allows to step from the premise $\Gamma, u : A \sim B$ to the following conclusions: $\Gamma, u : A \sim B, X : A \rightarrow B$ and $\Gamma, u : A \sim B, X : \neg\Box\neg A$, where X is a free-variable. X will be then instantiated when further information is available on the tableau, that is to say when such an instantiation will lead to close the branch with an axiom.

In the following we will use another kind of free-variables, called *conditional free-variables*, in order to deal with boxed formulas. Intuitively, a boxed formula $x : \Box\neg A$ will be represented by $[V, x] : \neg A$, where V is a conditional free-variable, representing all the successors of x (V is called "conditional" since x might have no successors). The two kinds of free-variables should not be confused; we will denote "jolly" free-variables with $X, X_1, X_2, \ldots$, and "conditional" free-variables with $V, V_1, V_2, \ldots$.

Similarly to Beckert and Goré's MODLEANTAP [6], FREEP 1.0 also adopts the following refinements:

- it makes use of integers starting from 1 to represent a single world; in this way, if `Max` is the maximal integer occurring in a branch, then we have that: a) `Max` is also the *number* of different labels occurring in that branch; b) the next new label to be generated is simply `Max+1`;
- the labels are *Prolog lists* representing *paths of worlds*; as a consequence, relation formulas, used in $\mathcal{LAB}\mathbf{P}$ to represent the preference relation, are replaced by *implicit relations* in the labels. Intuitively, when $(\Box^-)$ is applied to $[1] : \neg\Box\neg A$, a formula $[2, 1] : A$ is introduced rather than $[2] : A$ plus an explicit relation formula $[2, <, 1]$. In general, a label is a list $[n_k, n_{k-1}, \ldots, n_2, n_1]$, representing that $n_k < n_{k-1}, \ldots, n_2 < n_1$. From now on, we denote with $\sigma, \gamma, \ldots$, such "path labels";
- it replaces a boxed formula $\sigma : \Box\neg A$, where σ is a "path label", with a formula $[V, \sigma] : \neg A$, where V is a *conditional* free-variable, representing *all the paths descending from* σ, if any. In this case, the free-variable is conditional in the sense that it could be the case that σ has no sons in the tableau;
- in order to distinguish "jolly" free-variables from "conditional" free-variables, we partition each node of a tableau in two different components $\langle \Gamma \mid \Sigma \rangle$. Σ contains formulas whose labels contain conditional free-variables, of the form $[V, \sigma] : \neg A$, corresponding to boxed formulas $\sigma : \Box\neg A$. Γ contains the other formulas. This distinction is needed in order to avoid unwanted unification between elements of Σ. For instance, we do not want that a branch closes because it contains two formulas such as $[V, \sigma] : A$ and $[V_1, \sigma] : \neg A$; indeed, such a branch would be satisfiable by a model in which σ has no successors.

We use a Prolog-like notation to denote a path label. In this respect, $[n_1, \ldots, n_k]$ denotes a label whose elements are $n_1, \ldots, n_k$, $[\,]$ denotes the empty list, and $[n_1, \ldots, n_k, \sigma]$ denotes a list whose first k elements are $n_1, \ldots, n_k$ followed by the list σ. We also denote with $[\sigma_1, \sigma_2]$ the list obtained by appending σ_2 to σ_1. The basic ideas of the free-variables tableau implemented by FREEP 1.0 can be summarized as follows:

- when $(\mid\!\sim^{+})$ is applied, a jolly free-variable is introduced in the two conclusions; the free-variable represents all the labels occurring in the current branch, and will be instantiated later, when such an instantiation leads to close a branch with an axiom;
- when the $(\Box^{-})$ rule is applied to $\langle \Gamma, \sigma : \neg\Box\neg A \mid \Sigma\rangle$, then the formula $[n, \sigma] : A$ is added to Γ, where n is `Max`+1 and `Max` is the maximal integer occurring in the branch, whereas $[V, n, \sigma] : \neg A$ is added to Σ. The conditional free-variable V will then be instantiated to close a branch with an axiom. As an example, if another formula $[m, n, \sigma] : A$ (or $[m_2, m_1, n, \sigma] : A$) will be introduced in the branch, then the branch will close since V can be instantiated with $[m]$ (resp. $[m_2, m_1]$);
- given a tableau node $\langle \Gamma \mid \Sigma, [V, \sigma] : \neg A\rangle$, the branch is closed when there is a formula of the form $[\gamma, \sigma] : A$ belonging to Γ, where γ is a list of numbers. This machinery is used to perform the propagation of boxed formulas, rather than explicitly computing, step by step (when a new world is generated), the set $\Gamma^{M}_{x\to y}$. In fact, at the world $[\gamma, \sigma]$ reachable from σ, the formula $\neg A$ holds as the labelled formula $[V, \sigma] : \neg A$ says that $\neg A$ must hold in all the worlds reachable from σ (and V stands for any path from σ).

Observe that the improved implementation of the tableau for **R** given in [4] already makes use of jolly free-variables. However, it does not introduce neither path labels nor conditional free-variables, thus requiring to explicitly compute $\Gamma^{M}_{x\to y}$ step by step.

Coming back to FREEP 1.0, by looking carefully at how free-variables are introduced, it can be shown that labels containing jolly free-variables have the form $[n_1, n_2, \ldots, n_k, X]$, where $n_1, \ldots, n_k$ are integers and X is a jolly free-variable. On the contrary, labels containing conditional free-variables have the form $[V, \sigma]$, where V is a conditional free-variable and σ is a path label that might contain a jolly free-variable. In Figure 4 two derivations of $\{A \mid\!\sim B \wedge C, \neg(A \mid\!\sim B), \neg(D_1 \mid\!\sim E_1), \neg(D_2 \mid\!\sim E_2)\}$ are presented. The upper tableau is obtained by applying the simple implementation of $\mathcal{LAB}\mathbf{P}$ introduced in section 4.1, has height 9 and contains 59 nodes. The lower one, obtained by applying FREEP 1.0, is a tableau of height 6 and contains only 9 nodes.

Here again, the free-variables tableau calculus is implemented by the predicate **prove(Gamma,Sigma,Max,Cond,Tree)**, where `Gamma` and `Sigma` are Prolog lists representing a node $\langle \Gamma \mid \Sigma\rangle$ as described above and `Max` is the maximal integer in the current branch. `Cond` is used to apply $(\mid\!\sim^{+})$ in a controlled way; it is a list whose elements are triples of the form `[A=>B,N,Used]`, representing that $(\mid\!\sim^{+})$ has been applied `N` times on $A \mid\!\sim B$ in the current branch, by introducing the free-variables of the list `Used`. In general, `Used` is a list of paths partially instantiated. This machinery is used to control the application of $(\mid\!\sim^{+})$ as described in the previous section, in order to ensure termination by preventing that the rule is applied more than once by using the same label. Indeed, as we will see later, the rule is only applied to a conditional $A \mid\!\sim B$ if it has already been applied less than `Max` times in the current branch, i.e. `N<Max`. Furthermore, it must be ensured that all labels in `Used` are different. When the predicate `prove` succeeds, the argument `Tree` is instantiated by a Prolog functor representing the closed tableau found

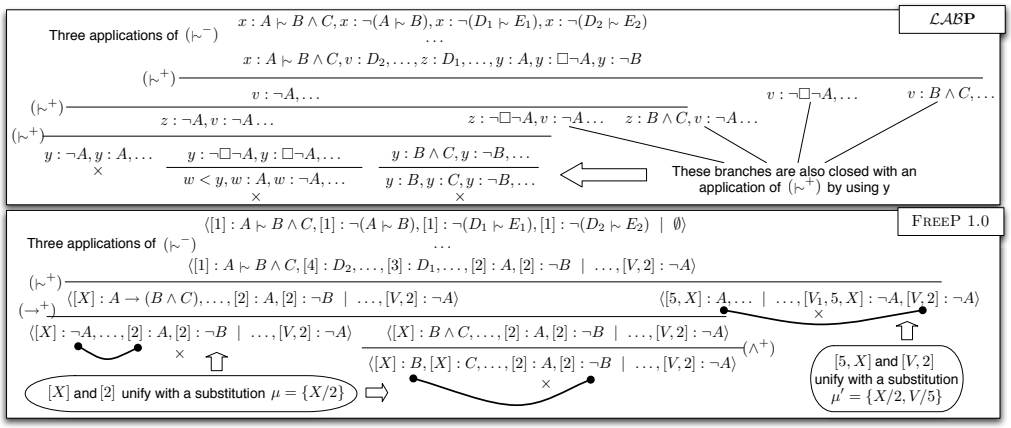

Fig. 4. A derivation of $\{A \sim B \wedge C, \neg(A \sim B), \neg(D_1 \sim E_1), \neg(D_2 \sim E_2)\}$ in $\mathcal{LAB}\mathbf{P}$ and in FREEP 1.0.

by the theorem prover. For instance, to prove that $A \sim B \wedge C, \neg(A \sim B)$ is unsatisfiable in **P**, one queries FREEP 1.0 with the following goal: `prove([[[1],a=>b and c],[[1],neg(a=>b)]],[ ],1,[ ], Tree)`.

Before introducing some examples of the clauses of FREEP 1.0, we introduce the following two definitions of the *unification* of path labels. A branch of a tableau is closed if it contains two formulas $\sigma : F$ and $\gamma : \neg F$ such that the two labels (paths) σ and γ unify, i.e. there exists a substitution of the free-variables occurring in them such that the two labels represent the same world. We distinguish the case where the two formulas involved in a branch closure are both in Γ, i.e. their labels only contain jolly free-variables, from the case where one of them belongs to Σ, i.e. its label also contains a conditional free-variable. The first case is addressed in Definition 4, whereas the second case is addressed in Definition 5.

Definition 4 (Unification of labels in Γ). *Given two formulas $\sigma : F \in \Gamma$ and $\gamma : G \in \Gamma$, the labels σ and γ* unify *with substitution μ if the following conditions hold:*

(A)
- *if $\sigma = \gamma = [\,]$, then $\mu = \epsilon$;*
- *if $\sigma = [n_1, \sigma']$ and $\gamma = [n_1, \gamma']$, where n_1 is an integer (Prolog constant), and σ' and γ' unify with a substitution μ', then $\mu = \mu'$;*
- *if $\sigma = [n_1, \sigma']$ and $\gamma = X$, where X is a jolly free-variable, and X does not occur in σ', then $\mu = X/[n_1, \sigma']$;*
- *if $\sigma = X_1$ and $\gamma = X_2$, where X_1 and X_2 are jolly free-variables, then $\mu = X_1/X_2$.*

(B) the substitution μ is such that, given two jolly free-variables X_1 and X_2 introduced on a branch by applying $(\sim^+)$ on the same conditional $A \sim B$, X_1 and X_2 are not instantiated with the same path by μ.

Definition 5 (Unification of labels in Γ and Σ). *Given two formulas $\sigma : F \in \Sigma$ and $\gamma : G \in \Gamma$, where $\sigma = [V, n_1, \ldots, n_k]$, the labels σ and γ* unify *with substitution $\mu = \mu' \circ \{V/\gamma_1\}$ if the following conditions hold:*

– *one can split γ in two parts, i.e. $\gamma = [\gamma_1, \gamma_2]$, such that $\gamma_1 \neq \epsilon$ and $\gamma_2 \neq \epsilon$*[2]*;*

[2] Given the form of labels in Γ, these conditions ensure that γ_1 does not contain free-variables.

– γ_2 *and* $[n_1, \ldots, n_k]$ *unify w.r.t. Definition 4 with a substitution* μ'.

Here below are the clauses implementing axioms:

```
prove(Gamma,_,_,Cond,...):-
   member([X,A],Gamma),member([Y,neg A],Gamma),
   matchList(X,Y),constr(Cond),!.
prove(Gamma,Sigma,_,Cond,...):-
   member([[V|X],A],Sigma),member([Y,neg A],Gamma),
   copyFormula([[V|X],A],Fml),split(Y,PrefY,SuffY),V=PrefY,
   matchList(SuffY,X),constr(Cond),!.
```

The first clause is applied when two formulas `[X,A]` and `[Y,neg A]` belong to `Gamma`. The predicate `matchList(X,Y)` implements the point (A) of Definition 4, i.e. it checks if there exists a substitution of the jolly free-variables belonging to `X` and `Y` such that these two labels unify. The predicate `constr(Cond)` implements point (B) in Definition 4. This ensures the condition of termination of the proof search; indeed, as an effect of the matching between labels, the instantiation of some free-variables could lead to duplicate an item in `Used`, resulting in a redundant application of $(\mid\!\sim^+)$ on $A \mid\!\sim B$ in the same world/label. The predicate `constr` avoids this situation.
The second clause checks if a branch can be closed by applying a free-variable substitution involving a formula in `Sigma`, i.e. a formula whose label has the form `[V|X]`, where `V` is a conditional free-variable. Given a formula `[Y,neg A]` in `Gamma` and a formula `[[V|X],A]` in `Sigma`, if the two path labels unify, then the branch can be closed. The clause implements Definition 5 by means of the auxiliary predicates `split,V=PrefY`, and `matchList`. In order to allow further instantiations of the conditional free-variable `V` in other branches, the predicate `copyFormula` makes a new copy the labelled formula `[[V|X],A]`, generating a formula `Fml` of the form `[[V'|X'],A]`, where `V'` is a conditional free-variable whose instantiation is *independent* from the instantiation of `V`. For details, see the implementation.

To search a derivation for a set of formulas, FREEP 1.0 proceeds as follows: first, it checks if there exists an instantiation of the free-variables such that the branch can be closed, by applying the clauses implementing axioms. Otherwise, the first applicable rule will be chosen. The ordering of the clauses is such that boolean rules are applied in advance. As another example of clause, here below is one of the clauses implementing $(\mid\!\sim^+)$. It is worth noticing that the following clause implements a further refinement of the calculus $\mathcal{LAB}\mathbf{P}$, namely the $(\Box^-)$ rule is "directly" applied to the conclusion of $(\mid\!\sim^+)$, which is the only rule that introduces a negated boxed formula $X : \neg\Box\neg A$ in the branch. In this way, an application of $(\mid\!\sim^+)$ to $\langle \Gamma, \sigma : A \mid\!\sim B \mid \Sigma\rangle$ leads to the following conclusions: $\langle \Gamma, \sigma : A \mid\!\sim B, X : A \rightarrow B \mid \Sigma\rangle$ and $\langle \Gamma, \sigma : A \mid\!\sim B, [n, X] : A \mid [X', n, X] : \neg A, \Sigma\rangle$. The rule $(\Box^-)$ can then be removed from the calculus.

```
prove(Gamma,Sigma,Max,Cond,...):-
   member([Label,A=>B],Gamma),
   select([A=>B,NumOfWildCards,PrevWildCards],Cond,NewCond),
   NumOfWildCards<Max,!,NewWildCards#=NumOfWildCards+1,
   prove([[[X],A->B]|Gamma],Sigma,Max,[[A=>B,NewWildCards,
           [[X]|PrevWildCards]]|NewCond],...),!,N#=Max+1,
   prove([[[N|[X]],A]|Gamma],[[[_|[N|[X]]],neg A]|DefSigma],N,
           [[A=>B,NewWildCards,[[X]|PrevWildCards]]|NewCond],...).
```

If a conditional formula `[Label,A=>B]` belongs to `Gamma`, then the above clause is invoked. If $(\mid\!\sim^{+})$ has already been applied to $A \mid\!\sim B$ in the current branch, then an element `[A=>B,NumOfWildCards,PrevWildCards]` belongs to `Cond`: the predicate `prove` is recursively invoked on the two conclusions of the rule only if the predicate `NumOfWildCards<Max` succeeds, by preventing that the rule is applied a redundant number of times. As mentioned above, this machinery, together with the predicate `constr` described above, ensure a terminating proof search.

4.3 Performances of FREEP 1.0

The performances of FREEP 1.0 are promising. We have tested FREEP 1.0 over 300 sets of formulas randomly generated, each one containing 100 conditional formulas, and we have compared its performances with KLMLean 2.0, a theorem prover implementing the non-labelled calculus $\mathcal{T}\mathbf{P}^{\mathbf{T}}$. The statistics obtained are shown in Table 1 and present the number of successes within a fixed time limit:

Implementation	1 ms	10 ms	100 ms	1 s	2.5 s	5 s
KLMLean 2.0	113	113	177	240	255	267
FREEP 1.0	157	158	207	255	269	276

Table 1. Some statistics comparing FREEP 1.0's performances with KLMLean 2.0's.

On the unsatisfiable set of formulas $\{\neg(A \vee D \mid\!\sim F \vee \neg C \vee \neg B \vee A), (A \wedge B) \vee (C \wedge D) \mid\!\sim E \wedge F, \neg(P_1 \mid\!\sim E \wedge F), \neg(P_2 \mid\!\sim E \wedge F), \neg(P_3 \mid\!\sim E \wedge F), \neg(P_4 \mid\!\sim E \wedge F), \neg(P_4 \mid\!\sim E \wedge F), \neg((A \wedge B) \vee (C \wedge D) \mid\!\sim G_1), \neg((A \wedge B) \vee (C \wedge D) \mid\!\sim G_2)\}$, FREEP 1.0 succeeds in less than 2 s, whereas KLMLean 2.0 requires 1 minute. In future research we intend to improve the performances of FREEP 1.0 by experimenting standard refinements and heuristics. For instance, we intend to investigate how to increase the efficiency of the machinery adopted in order to ensure a terminating proof search by means of the constraint logic programming. Moreover, we aim to extend the free-variables technique to the case of the other KLM logics.

References

1. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence **44**(1-2) (1990) 167–207
2. Friedman, N., Halpern, J.Y.: Plausibility measures and default reasoning. Journal of the ACM **48**(4) (2001) 648–685
3. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux for KLM Preferential and Cumulative Logics. In: Proc. of LPAR 2005, LNAI 3835, Springer (2005) 666–681
4. Giordano, L., Gliozzi, V., Pozzato, G.L.: Klmlean 2.0: a theorem prover for klm logics of nonmonotonic reasoning. In: Proc. of TABLEAUX 2007, to appear (2007)
5. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer (2nd ed.) (1996)
6. Beckert, B., Goré, R.: Free-variable tableaux for propositional modal logics. Studia Logica **69**(1) (2001) 59–96
7. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux Calculi for KLM Rational Logic R. In: Proc. of JELIA 2006. Volume 4160 of LNAI., Springer (2006) 190–202
8. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence **55**(1) (1992) 1–60