A KE Tableau for a Logic of Formal Inconsistency

Adolfo Neto and Marcelo Finger

Computer Science Department Institute of Mathematics and Statistics University of São Paulo {adolfo,mfinger}@ime.usp.br

Abstract. In this paper we describe a **KE** tableau system for a Logic of Formal Inconsistency (**LFI**) called **mCi**. The family of **LFI**s is a family of paraconsistent logics with philosophical relevance and applications in computer science. We prove that the **KE** System for **mCi** is correct and complete, and describe the implementation of such a system, presenting the strategies designed for the actual implementation within the **KEMS** framework; **KEMS** is a multi-strategy theorem prover based on the **KE** refutation method for propositional logics. We conclude by presenting some problems we have developed to evaluate theorem provers for **mCi**, as well as the evaluation results obtained with **KEMS**.

1 Introduction

Paraconsistent logics can be used as the underlying logic to inconsistent but nontrivial theories [10]. Logics of Formal Inconsistency (LFIs) [3] are a family of paraconsistent logics that internalize the notions of consistency and inconsistency at the object-language level. This family of logics has some nice proof-theoretic features and have been used in some computer science applications such as the integration of inconsistent information in multiple databases [9]. They can also be used as a tool for knowledge engineering and as the base logic in rule-based decision-support systems.

We have designed and implemented **KEMS** [11,13], a multi-strategy theorem prover based on the **KE** method [8,7] for propositional logics. **KEMS** current version implements strategies for three logics: classical propositional logic (**CPL**) and two **LFIs**: **mbC** and **mCi**. In a previous paper [12], we presented the implementation of a **KE** system for **mbC** in **KEMS**. In this paper we present our investigations and results concerning **mCi**, a stronger logical system.

1.1 Outline

Here are the contributions of this paper. First, after presenting \mathbf{mCi} in Section 2 we give a new proof that \mathbf{mCi} 's decision problem is co-NP-complete. Second, we present a new **KE** tableau system for \mathbf{mCi} and prove that this system is correct

and complete (Section 3). Third, we describe the two new **KEMS** strategies we have designed that use the **mCi KE** system in Section 4. With these strategies, we have a theorem prover for **mCi**. Fourth, we present three problem families to evaluate theorem provers for **mCi** (Section 5). These problems are used in the evaluation of **mCi KEMS** strategies but can also be used in future theorem provers for **mCi**. Next, we show the first benchmark results for these families, comparing the performance of two **KEMS** strategies for **mCi**. Finally, we present concluding remarks and point to further work in Section 6.

2 mCi, a Logic of Formal Inconsistency

Logics of Formal Inconsistency are a class of paraconsistent logics, which internalize the notions of consistency and inconsistency at the object-language level [3]. A logic is *paraconsistent* if it can be used as the underlying logic to inconsistent but non-trivial theories, which we call paraconsistent theories [10].

A Logic of Formal Inconsistency is defined in [3] as any logic in which the 'Principle of Explosion' (ie $\forall \Gamma \forall A \forall B(\Gamma, A, \neg A \vdash B)$) does *not* hold, but the 'Principle of Gentle Explosion' does, that is $\forall A \forall B(\Gamma, \bigcirc(A), A, \neg A \vdash B)$ for some set of formulas $\bigcirc(A)$ depending on A. This family of logics has some nice prooftheoretic features and have been used in some computer science applications such as the integration of inconsistent information in multiple databases [9].

2.1 Axiomatization

The logic **mCi** is an **LFI** based on classical logic [3] and adds a new unary symbol \circ , where $\circ A$ means "A is consistent". Any **LFI** based on classical logic can be axiomatized starting from positive classical logic (**CPL**⁺), whose axiomatization is that of **CPL** without the (**exp**) axiom schema (see axiomatization below). We assume familiarity with the syntax and semantics of **CPL** and present below the **CPL** axiomatization used in [3] as a basis for defining **LFI** axiomatizations: **Axiom schemas:**

 $\begin{array}{l} (\mathbf{Ax1}) \quad A \to (B \to A) \\ (\mathbf{Ax2}) \quad (A \to B) \to ((A \to (B \to C)) \to (A \to C)) \\ (\mathbf{Ax3}) \quad A \to (B \to (A \land B)) \\ (\mathbf{Ax4}) \quad (A \land B) \to A \\ (\mathbf{Ax5}) \quad (A \land B) \to B \\ (\mathbf{Ax5}) \quad (A \land B) \to B \\ (\mathbf{Ax6}) \quad A \to (A \lor B) \\ (\mathbf{Ax7}) \quad B \to (A \lor B) \\ (\mathbf{Ax8}) \quad (A \to C) \to ((B \to C) \to ((A \lor B) \to C)) \\ (\mathbf{Ax9}) \quad A \lor (A \to B) \\ (\mathbf{Ax10}) \quad A \lor \neg A \\ (\mathbf{exp}) \quad A \to (\neg A \to B) \end{array}$

Inference rule schema:

(MP) $\frac{A, A \rightarrow B}{B}$

The axiomatization for \mathbf{mCi} is obtained from \mathbf{CPL}^+ axiomatization, by adding the following axiom schemas:

(bc1)
$$(\circ A) \to (A \to (\neg A \to B));$$

(ci) $(\neg \circ A) \to (A \land \neg A);$
(cc)_n $\circ \neg^n \circ A$ $(n \ge 0).$

where $(\mathbf{cc})_n$ is an infinite set of axiom schemas:

1.
$$(\mathbf{cc})_0: \circ \circ A;$$

2. $(\mathbf{cc})_1: \circ \neg \circ A;$
3. $(\mathbf{cc})_2: \circ \neg \neg \circ A;$
 \vdots

Notice that a new unary connective was introduced here: ' \circ ', the consistency connective. The intended reading of $\circ A$ is 'A is consistent'. In **mCi**, $\circ A$ is log-ically independent from $\neg(A \land \neg A)$, that is, \circ is a primitive unary connective, not an abbreviation depending on conjunction and negation, as it happens in da Costa's C_n hierarchy of paraconsistent logics [6].

An inconsistency connective '•' can be defined in **mCi** as $\bullet A \stackrel{\text{def}}{=} \neg \circ A$. It was shown in [3] that **mCi** can be defined either setting 'o' as primitive and '•' as defined, or the opposite, that is, by setting '•' as primitive and 'o' as defined.

2.2 Semantics

Let $\mathbf{2} \stackrel{\text{def}}{=} \{0, 1\}$ be the set of truth-values, where 1 denotes the 'true' value and 0 denotes the 'false' value. Let *For* be the set of **mCi** formulas. Below we inductively present a definition of a valuation for **mCi** [3]:

Definition 1. An mCi-valuation is any function $v : For \longrightarrow 2$ subject to the following clauses:

(v1) $v(A \land B) = 1$ iff v(A) = 1 and v(B) = 1; (v2) $v(A \lor B) = 1$ iff v(A) = 1 or v(B) = 1; (v3) $v(A \to B) = 1$ iff v(A) = 0 or v(B) = 1; (v4) $v(\neg A) = 0$ implies v(A) = 1; (v5) $v(\circ A) = 1$ implies v(A) = 0 or $v(\neg A) = 0$; (v6) $v(\neg \circ A) = 1$ implies v(A) = 1 and $v(\neg A) = 1$; (v7.n) $v(\circ \neg^n \circ A) = 1$ for n > 0.

A formula X is said to be *satisfiable* if truth-values can be assigned to its propositional variables in a way that makes the formula true, i.e. if there is at least one valuation such that v(X) = 1. A formula is a *tautology* if all possible valuations make the formula true. In **mCi**, for instance, $A \vee B$ is satisfiable, but it is not a tautology, while $\neg((A \land \neg A) \land (\circ A))$ is a tautology.

It is important to notice that the semantics in Definition 1 is non-deterministic due to clauses (v4)-(v6).

Let Γ be a set of formulas in *For*, and A a formula in *For*. We say that A is a semantical consequence of Γ (denoted by $\Gamma \models A$) if for any valuation v we have the following [2]:

if
$$v(B) = 1$$
 for all B in Γ , then $v(A) = 1$.

The **mCi** axiomatization we have presented above is sound and complete with respect to the semantical consequence relation presented in Definition 1 (see [3]). That is, for any Γ and A, $\Gamma \vdash_{\mathbf{CPL}} A$ implies $\Gamma \models_{\mathbf{CPL}} A$ (soundness [2]). And $\Gamma \models_{\mathbf{CPL}} A$ implies $\Gamma \vdash_{\mathbf{CPL}} A$ (strong completeness [2]).

It is easy to verify (and it was shown in [3]) that $A \wedge \neg A \dashv \neg \circ A$ in **mCi**. Therefore, $\neg \circ A$ and $(A \wedge \neg A)$ are equivalent in **mCi**. The $(\mathbf{cc})_n$ axioms were used in the definition of **mCi**(*i*) to make formulas of the form $\neg \circ A$ 'behave classically', and (*ii*) to obtain a logic that is controllably explosive in contact with formulas of the form $\neg^n \circ A$, where $\neg^0 A \stackrel{\text{def}}{=} A$ and $\neg^{n+1} A \stackrel{\text{def}}{=} \neg \neg^n A$. That is, with these axioms, any formula of the form $\neg^n \circ A$ 'behaves classically' such that $\{\neg^n \circ A, \neg^{n+1} \circ A\}$ is an explosive theory in **mCi**. Much more about **mCi** can be found in [3].

2.3 Complexity

The satisfiability problem for **CPL** formulas (SAT) was the first known NPcomplete problem. The class of NP-complete problems is a subclass of NP. While P is the class of decision problems that can be solved in polynomial time by a *deterministic* algorithm, NP is the class of decision problems that can be solved in polynomial time by a *nondeterministic* algorithm. Therefore $P \subseteq NP$. The problems in NP are such that positive solutions can be verified in polynomial time. NP-complete problems are the most difficult problems in NP, the ones most likely not to be in P. If we find a polynomial time algorithm for any NPcomplete problem, we can solve all problems in NP in polynomial time, because there is a polynomial time reduction from any NP problem into any NP-complete problem.

The *complement* of a decision problem is the decision problem resulting from reversing the 'yes' and 'no' answers. We can generalize this to the complement of a complexity class, called the complement class, which is the set of complements of every problem in the class. co-NP is the complement of the complexity class NP. It is the class of problems for which a 'no' answer can be verified in polynomial time. And co-NP-complete is the complement of the class of NP-complete problems.

The **CPL** decision problem (given a propositional formula, decide whether or not it is a *tautology*) is co-NP-complete, because a formula in **CPL** is a tautology if and only if its negation is unsatisfiable; and the **CPL** satisfiability problem is a well-known NP-complete problem [5]. Here we prove that the **mCi** decision problem is also co-NP-complete, as it was suggested in [3]. First we have to show that the **mCi** decision problem is co-NP-hard. This is true because it was shown in [3] that the decision problem for **mbC**, a **LFI** of which **mCi** is an extension, is co-NP-complete.

To complete the proof, we need a NP algorithm for the complement of **mCi** decision: the falsification of a formula. That is, we must show that given a formula A and an **mCi**-valuation v it is possible to verify if v(A) = 0 in polynomial time.

Let A be an **mCi** formula. We show below how to construct an **mCi**-valuation v for A. This is here only to show that it is more difficult to build an **mCi**-valuation than a **CPL**-valuation.

Let SSF(A) be the set of all strict subformulas of A. A *strict subformula* of A is any subformula of A except A itself. Then we construct a new set ESSF(A), such that for all $X \in SSF(A)$, X, $\circ X$ and $\neg X$ belong to ESSF(A).

If n is the size of A, then the size of ESSF(A) is at most 2n - 2. To build a valuation v for A we must, for any $X \in \text{ESSF}(A)$, set v(X) either to 0 or to 1, obeying the **mCi**-valuation clauses presented in Definition 1.

Up to now, we have only v(X) for all $X \in \text{ESSF}(A)$ (not necessarily a value for v(A)). The following algorithm allows us to find a value for v(A):

1. if, for some X, A is $\circ X$, then:

- (a) if $v(\neg X) = 0$, then v(X) = 1 and v(A) can be set either to 0 or to 1;
- (b) if v(X) = 1 and $v(\neg X) = 1$, then v(A) = 0;
- (c) if v(X) = 0 and $v(\neg X) = 1$, then v(A) can be set either to 0 or to 1;
- 2. if, for some X, A is $\neg X$, then:
 - (a) if v(X) = 1 then:
 - i. if $v(\circ X) = 1$ then v(A) = 0;
 - ii. if $v(\circ X) = 0$ then v(A) can be set either to 0 or to 1;
 - (b) if v(X) = 0 then v(A) = 1;
- 3. if, for some X, Y, A is X ∧ Y, then:
 (a) if v(X) = 1 and v(Y) = 1, then v(A) = 1;
 - (b) otherwise, v(A) = 0;
- 4. if, for some X, Y, A is $X \vee Y$, then:
 - (a) if v(X) = 0 and v(Y) = 0, then v(A) = 0;
 - (b) otherwise, v(A) = 1;
- 5. if, for some X, Y, A is $X \to Y$, then:
 - (a) if v(X) = 1 and v(Y) = 0, then v(A) = 0;
 - (b) otherwise, v(A) = 1.

Therefore, it is more difficult to build an **mCi**-valuation than a **CPL**-valuation¹. But, given a formula A, if we have a valuation for all formulas in ESSF(A), it is easy to verify that v(A) can be 0. The algorithm above is clearly polynomial in time (and also in space). As the NP class contains the problems that can be verified in polynomial time [5], the complement of the decision problem (falsification) for **mCi** is in NP. Therefore, the decision problem for **mCi** is co-NP-complete.

¹ A **CPL**-valuation can be built by setting values only to atomic formulas (see [3]).

3 A KE System for mCi

A sound and complete tableau system for \mathbf{mCi} obtained by using a general method for constructing tableau systems was presented in [3]. Let us call this system an $\mathbf{mCi} \ \mathbf{C}^{3}\mathbf{M}$ tableau system. The $\mathbf{C}^{3}\mathbf{M}$ tableau system rules for \mathbf{mCi} are shown in Figure 1. The rules for the binary connectives are the same as that from Smullyan's analytic tableaux (**AT**) [15].

$\frac{T A \to B}{F A \mid T B} \ (T \to)$	$\frac{\mathbf{F} A \to B}{\mathbf{T} A} (\mathbf{F} \to)$ $\mathbf{F} B$
$\frac{\mathbf{F} A \wedge B}{\mathbf{F} A \mid \mathbf{F} B} (\mathbf{F} \wedge)$	$\frac{\begin{array}{c} T A \land B \\ T A \\ T B \end{array}}{\operatorname{T} A} (T \land)$
$\frac{T A \lor B}{T A \mid T B} (T \lor)$	$\frac{\mathbf{F} A \lor B}{\mathbf{F} A} (\mathbf{F} \lor)$ $\mathbf{F} B$
$\frac{T \circ A}{F A F \neg A} (T \circ)$	$\frac{\mathbf{F} \neg A}{\mathbf{T} A} \ (\mathbf{F} \neg)$
$\frac{\operatorname{T}_{\neg}(\circ A)}{\operatorname{T}_{\neg} A} (\operatorname{T}_{\neg} \circ)$	$\overline{T \circ (\neg^{n}(\circ A))}$ for $(n \ge 0)$ $(T \circ \neg^{n} \circ)$
$TA \mid FA $ (PB)	

Fig. 1. mCi C^3M tableau rules.

In total, the **mCi** $C^{3}M$ system has 5 branching rules. As explained in [7], branching rules lead to inefficiency. To obtain a more efficient proof system, we used the $C^{3}M$ tableau system for **mCi** as a basis to devise an original **mCi KE** system. The rules of this system are presented in Figure 2.

Let us briefly explain how we have arrived at this set of rules. First, we have substituted all **mCi** $\mathbf{C}^{3}\mathbf{M}$ rules for binary connectives by **CPL KE** rules [7] for binary connectives. In this way we got rid of 3 branching rules. After that, we have substituted (T \circ) by (T \neg '), to get rid of another branching rule. Notice that (T \neg ') is a rule that can be derived in the **mCi** $\mathbf{C}^{3}\mathbf{M}$ system, using the definition of derived rule presented in [4]. For instance, (T \neg ') can be derived in **mCi** $\mathbf{C}^{3}\mathbf{M}$ because {T $A, T \circ A, FA$ } closes is **mCi** $\mathbf{C}^{3}\mathbf{M}$.

With only one branching rule left (PB) (the Principle of Bivalence), we can say that this is a **KE** system; the rule (PB) corresponds directly to the sequent

calculus *cut rule*. We went further and have substituted the infinite set of rules $(\mathbf{T} \circ \neg^n \circ)$ by the branch closing condition $(\mathbf{F} \circ \neg^n \circ)$, since the last set of rules is analytic and it is easier to implement in **KEMS** (see Section 4), because it has one premise.

$\frac{\begin{array}{c} T A \to B \\ T A \\ \hline T B \end{array}}{T B} (T \to 1)$	$\frac{\begin{array}{c} \mathbf{T} A \to B \\ \mathbf{F} B \\ \mathbf{F} A \end{array}}{\mathbf{F} A} (\mathbf{T} \to_2)$	$\frac{\mathbf{F} A \to B}{\mathbf{T} A} \mathbf{F} B (\mathbf{F} \to)$
	$\frac{\begin{array}{c} \mathbf{F} A \wedge B \\ \mathbf{T} B \\ \overline{\mathbf{F} A} \end{array} (\mathbf{F} \wedge_2)$	$\frac{\begin{array}{c} {\rm T}A \wedge B}{\begin{array}{c} {\rm T}A} & ({\rm T} \wedge) \end{array} \\ {\rm T}B \end{array}$
$ \begin{array}{c} T A \lor B \\ F A \\ T B \end{array} (T \lor_1) $	$ \begin{array}{c} T \ A \lor B \\ F \ B \\ \hline T \ A \end{array} (T \lor_2) $	$\frac{\mathbf{F} A \vee B}{\mathbf{F} A} (\mathbf{F} \vee)$ F B
$\frac{T \neg A}{T \circ A} (T \neg ')$	$\frac{\mathbf{F} \neg A}{\mathbf{T} A} (\mathbf{F} \neg)$	
$\frac{\mathbf{T} \neg (\circ A)}{\mathbf{T} A} (\mathbf{T} \neg \circ)$ $\mathbf{T} \neg A$	$\frac{F \circ (\neg^{n}(\circ A))}{\times} \text{ for }$	$(n \ge 0) (\mathbf{F} \circ \neg^n \circ)$
$\frac{TA \mid FA}{TA \mid FA}$ (PB)		

Fig. 2. mCi KE rules.

3.1 Correctness and Completeness Proof

Our intention here is to prove that the **mCi KE** system is sound and complete with respect to **mCi** valuation semantics. The proof will be as follows. First we will redefine the notion of downward saturatedness for **mCi**. Then we will prove that every downward saturated set is satisfiable. The **mCi KE** proof search procedure for a set of signed formulas S either provides one or more downward saturated sets that give a valuation satisfying S or finishes with no downward saturated set.

Therefore, if an **mCi KE** tableau for a set of formulas S closes, then there is no downward saturated set that includes it, so S is unsatisfiable. However, if the tableau is open and completed, then any of its open branches can be represented as a downward saturated set and be used to provide a valuation that satisfies S. We then conclude that the **mCi KE** system is sound and complete. **Definition 2.** A set of **mCi** signed formulas *DS* is *downward saturated*:

- 1. whenever a signed formula is in DS, its conjugate is *not* in DS;
- 2. when all premises of any **mCi KE** rule (except (PB) and ($\mathbf{F} \circ \neg^n \circ$), for $n \ge 0$) are in DS, its conclusions are also in DS;
- 3. when the major premise of a two-premise **mCi KE** rule is in DS, either its auxiliary premise or its conjugate is in DS. If $\mathbb{T} \neg X$ is in DS, either $\mathbb{T} \circ X$ or $\mathbb{F} \circ X$ can be in DS, but only if $\circ X$ is a subformula of some other formula in DS. If $\circ X$ is not a subformula of some other formula in DS, neither $\mathbb{T} \circ X$ nor $\mathbb{F} \circ X$ are in DS;
- 4. if a signed formula S X is in DS, then for any sign S, for any formula X, for all subformulas Y of X and for all $n \ge 0$, the signed formula $\mathbb{T} \circ \neg^n \circ Y$ is in DS.

We now prove a Hintikka's Lemma for **mCi** downward saturated sets:

Lemma 1. (*Hintikka's Lemma*) Every **mCi** downward saturated set is satisfiable.

Proof. For any downward saturated set DS, we can easily construct an **mCi** valuation v such that for every signed formula SX in the set, v(SX) = 1. How can we guarantee this is in fact a valuation? First, we know that there is no pair **T** X and **F** X in DS. Second, all **mCi KE** rules with one or more premises (except ($\mathbf{F} \circ \neg^n \circ$) rules) preserve valuations. Note that ($\mathbf{F} \circ \neg^n \circ$) rules are taken into account by the last clause in Definition 2. That is, if we have a set of signed formulas that contains $\mathbf{F} \circ \neg^n \circ X$, every downward saturated set that contains this set should also contain $\mathbf{T} \circ \neg^n \circ X$. Therefore it is not downward saturated. To be downward saturated a set DS must contain, for all its subformulas² X, $\mathbf{T} \circ \neg^n \circ X$ (and must not contain any $\mathbf{F} \circ \neg^n \circ X$). As we can see in clause (**v7**.*n*) of the **mCi** valuation definition (see Definition 1), $v(\mathbf{T} \circ \neg^n \circ X) = 1$ for all X. Therefore, DS is satisfiable.

Theorem 1. Let DS' be a set of signed formulas. DS' is satisfiable if and only if there exists a downward saturated set DS'' such that $DS' \subseteq DS''$.

Proof. (\Leftarrow) First, let us prove that if there exists a downward saturated set DS" such that $DS' \subseteq DS$ ", then DS' is satisfiable. This is obvious because from DS" we can obtain a valuation that satisfies all formulas in DS", and $DS' \subseteq DS$ ".

 (\Rightarrow) Now, let us prove that if DS' is satisfiable, there exists a downward saturated set DS'' such that $DS' \subseteq DS''$.

So, suppose that DS' is satisfiable and that there is no downward saturated set DS'' such that $DS'' \supseteq DS'$. Using items (2) and (3) of Definition 2, we can obtain a family of sets of signed formulas DS'_i $(i \ge 1)$ that include DS'. If none of them is downward saturated, it is because for all i, $\{TX, FX\} \in DS'_i$ for some X. But all rules are valuation-preserving, so this can only happen if DS is unsatisfiable, which is a contradiction.

² To be precise, by the subformulas of a set of signed formulas $\{S_iF_i\}$, where S_i is a sign and F_i is an unsigned formula, we mean the set of subformulas of $\{F_i\}$.

Corollary 1. DS' is a unsatisfiable set of formulas if and only if there is no downward saturated set DS'' such that $DS'' \subseteq DS'$.

Theorem 2. The mCi KE system is sound and complete.

Proof. The **mCi KE** proof search procedure for a set of signed formulas S either provides one or more downward saturated sets that give a valuation satisfying S or finishes with no downward saturated set. The **mCi KE** system is a refutation system. The **mCi KE** system is sound because if an **mCi KE** tableau for a set of formulas S closes, then there is no downward saturated set that includes it, so S is unsatisfiable. If the tableau is open and completed, then any of its open branches can be represented as a downward saturated set and be used to provide a valuation that satisfies S (in other words, S is satisfiable).

The **mCi KE** system is complete because if S is satisfiable, no **mCi KE** tableau for a set of formulas S closes. And if S is unsatisfiable, all completed **mCi KE** tableau for S close.

4 KEMS for mCi

KEMS is a multi-strategy theorem prover based on the **KE** method for propositional logics [11,13]. A multi-strategy theorem prover is a theorem prover where we can vary the strategy without modifying the core of the implementation. Everytime we want to solve a problem with **KEMS** we must choose a strategy and a sorter. A strategy is responsible, among other things, for: (*i*) choosing the next inference rule to be applied, (*ii*) choosing the formula on which to apply the Cut rule, and (*iii*) finding contradictions that close proof tree branches. A sorter is an object that tells a strategy how to sort a list of formulas before trying to apply rules. Further details about sorters can be found in [11]. We present below the actual **KE** systems for **mCi** we have implemented in **KEMS** as well as the two strategies for **mCi** implemented in **KEMS** current version.

4.1 Extended mCi KE System

In **KEMS** we work with an extended version of the **mCi KE** (see Section 3) system. This extended version, called **e-mCi-KE**, has two additional zeroary connectives: ' \top ' and ' \perp '. In Figure 3 we present the **KE** rules added to the original **mCi KE** system to obtain **e-mCi-KE**.



Fig. 3. 'Top' and 'bottom' KE rules.

Trying to achieve better performance with some problems, we have added some derived rules to **e-mCi-KE**. These rules can be used to shorten some proofs. In Figure 4 we show four rules that can be derived from the original rules and that can be added to the **e-mCi-KE**.

Fig. 4. Derived mCi KE rules.

As a matter of fact, the current **KEMS** version has **mCi** strategies that use only $(T \circ'')$ and $(T \neg'')$ as additional rules, but new **KEMS** strategies can be implemented to use the other two rules, or even other derived rules.

4.2 Strategies for mCi

When we decided to implement strategies for **mCi**, we had already implemented some strategies for **CPL** and **mbC**. Therefore, we could use their implementation as a basis for the implementation of **mCi** strategies, reusing much of the code. For **mCi**, we have implemented the two following strategies.

mCi Simple Strategy This is an extension of **CPL** Simple Strategy (see [11]) for **mCi**. It implements the **e-mCi-KE** system. This is the order of rule applications:

- 1. all mCi KE one-premise rules;
- 2. all **mCi KE** two-premise rules;
- 3. (PB) rule.

To apply a one-premise rule is easy. We iterate over a list of formulas. For each formula, we verify if the formula matches the pattern of the premise of an one-premise rule. If it does, we apply the rule and include its conclusion in the list.

It is a little bit more difficult to apply a two-premise rule. We also iterate over a list of formulas. For each formula, we verify if the formula matches the pattern of the main³ premise of a two-premise rule. If it does, we look for a

³ In our presentation, the main premise is always the first premise loooking top-down. The other premise is called auxiliary.

formula that matches the auxiliary premise of the same rule for the previously found main premise. If it finds the second premise, we apply the rule and include its conclusion in the list.

When there is no more one- or two-premise rule that can be applied, we apply the (PB) rule and branch the proof. We then put one of the branches in the top of a stack of open branches to be further analyzed and continue the proof procedure with the other branch.

mCi Extended Strategy This is an extension of the previous strategy. It implements the **e-mCi-KE** system extended with two derived rules: $(T \circ'')$ and $(T \neg'')$. This is the order of rule applications:

- 1. all **mCi KE** one-premise rules;
- 2. all **mCi KE** original two-premise rules;
- 3. all **mCi KE** derived two-premise rules;
- 4. (PB) rule.

The other features are equal to mCi Simple Strategy features.

5 Evaluation

Theorem provers are usually compared by using benchmarks [17]. SATLIB [14] (for **CPL**) and TPTP [16] (for first-order classical logic) are two web sites that contain benchmark problems to evaluate theorem provers. As there were no existing family of difficult problems for **LFI**s, we have developed new problem families to test **KEMS**. These families can be used to evaluate future theorem provers for **mCi** and other **LFI**s and are presented below.

5.1 Problem Families to Evaluate mCi Provers

In [11] we have presented nine families of difficult problems to evaluate **LFI** theorem provers. Here we present the seventh, eighth, and ninth families, which were developed specifically to evaluate **mCi** theorem provers. We had two objectives in mind. First, to obtain families of valid problems whose **KE** proofs were as complex as possible. And second, to devise problems which required the use of many, if not all, **mCi KE** rules. These families are not classically valid, since their formulas use the consistency and inconsistency connectives. However, if we define $\circ X \stackrel{\text{def}}{=} \top$ and $\bullet X \stackrel{\text{def}}{=} \bot$ in **CPL**⁴, then all families become **CPL**-valid and can be used for evaluating **CPL** provers.

The sequent to be proved (Φ_n^7) for the seventh family is:

$$\bigwedge_{i=1}^{n} (A_i), \bigwedge_{i=1}^{n} (B_i \to (\neg A_i)), \bigvee_{i=1}^{n} (\circ A_i) \vdash \bigwedge_{i=1}^{n} ((\bullet A_i) \lor (\neg B_i))$$

⁴ A natural way of extending **CPL** presented in [3].

In this sequent, the $\bigvee_{i=1}^{n} (\circ A_i)$ formula is actually not essential to arrive at the conclusion. Therefore, we can define a variant (called $\Phi^{7'}$) of this family where this formula does not appear. The Φ^{7} family is probably more difficult to prove because of the irrelevant premise.

For the eighth family, this is the sequent to be proved (Φ_n^8) :

$$\bigvee_{i=1}^{n} (\bullet A_i), \bigwedge_{i=1}^{n} (A_i \to (\neg B_i)), \bigwedge_{i=1}^{n} ((\neg A_i) \to C_{(n-i+1)}) \vdash \bigwedge_{i=1}^{n} ((\neg B_i) \land C_{(n-i+1)})$$

Finally, the sequent to be proved (Φ_n^9) for the ninth family is:

$$\bigvee_{i=1}^{n} (\circ A_i), \bigwedge_{i=1}^{n} (B_i \to (\bullet A_i)) \vdash \bigvee_{i=1}^{n} \neg ((\circ (\circ A_i)) \to B_i)$$

We can have several valid variations of the ninth family, for $m \ge 0$ and $p \ge 0$:

$$\bigvee_{i=1}^{n} (\neg^{m}(\circ A_{i})), \bigwedge_{i=1}^{n} (B_{i} \to (\neg^{m}(\bullet A_{i}))) \vdash \bigvee_{i=1}^{n} \neg((\circ(\neg^{p}(\circ A_{i}))) \to B_{i})$$

where $(\neg^1 A) \stackrel{\text{\tiny def}}{=} (\neg A)$ and $(\neg^n A) \stackrel{\text{\tiny def}}{=} (\neg (\neg^{n-1} A)).$

5.2 Results

In this section we exhibit some of the results obtained by **KEMS** on the problem families we just presented. All results were obtained on a Pentium IV machine with a 3.20G Hz processor and 3775MB memory running Linux version 2.6.15-26-386. The java -jar kems.jar command⁵ was issued with the -Xms200m -Xmx2048m options to set the initial and maximum heap sizes. This allows **KEMS** to use more of the computer's main memory than the default memory allocated by the java virtual machine. The time limit for the proof search procedure was set to three minutes.

The user can present to **KEMS** a problem and a prover configuration. The most important prover configuration parameters for our evaluation were the strategy and the sorter. The reason is that we have noticed through our experiments that these two are the parameters that most affect a prover configuration performance. For this reason, in the following we will refer to a prover configuration as a strategy-sorter pair, or simply a pair.

In the tables we display below each prover configuration will be represented by a binary tuple:

<strategyId,sorterId>

where strategyId and sorterId can vary.

These are the ids for strategies:

⁵ The command used to execute kems.jar, which is the java archive that contains KEMS executable version.

MCISS - mCi Simple Strategy; MCIES - mCi Extended Strategy.

And these are the ids for sorters:

ins - insertion order;
rev - reverse order;
and - 'and' connective;
or - 'or' connective;
imp - 'implication' connective;
T - 'true' sign;
F - 'false' sign;
inc - increasing complexity;
dec - decreasing complexity;
nfo - string order;
rfo - reverse string order.

In Tables 1, 2 and 3 we present the results (time spent in milliseconds and proof size) obtained by some selected strategy-sorter pairs for the bigger instance solved of each **mCi** family. And in Table 4 we present best **mCi** strategy-sorter pairs in time and proof size.

We conclude that **mCi** Simple Strategy and **mCi** Extended Strategy achieved comparable results. In all **LFI** tests the sorter in the best pairs varied according to the problem family. And it was interesting to notice that for some families the sorter choice was almost as important as the strategy choice.

The results obtained by **KEMS** with the **mCi** families are the first benchmark results for these families. These results can be compared with other provers for **mCi** and are available in **KEMS** site [13].

Pair	Time spent	Proof Size	Comments
<mcies, and=""></mcies,>	849	3524	best in time
<mciss,nfo></mciss,nfo>	883	3504	best in size
<mciss,f></mciss,f>	963	3504	best in size
<mciss, and=""></mciss,>	964	3504	best in size
<mcies.rev></mcies.rev>	40445	105872	worst in time and size

Table 1. mCi Φ_{20}^7 results table.

6 Conclusion

We have presented in this paper a **KE** tableau system for **mCi** and we proved that this system is correct and complete. We have also presented here the two **KEMS** strategies we have designed that use the **mCi KE** system. With these

Pair	Time spent	Proof Size	Comments
<mciss,or></mciss,or>	18430	25407	best in time and in size
<mciss,dec></mciss,dec>	18442	25407	second best in time
<mcies,or></mcies,or>	18635	25507	worst in size
<mcies,dec></mcies,dec>	18969	25507	worst in time and size
Table 2 m C : σ^8 modults table			

Table 2. mCi Φ_{50}^{s} results table.

Pair	Time spent	Proof Size	Comments
<mcies,inc></mcies,inc>	59644	47691	best in time and size
<mcies,imp></mcies,imp>	59818	47691	second best in time
<mciss,rev></mciss,rev>	100869	48142	worst in time and size

Table 3. mCi Φ_{75}^9 results table.

strategies, we have built a theorem prover for **mCi**. Besides that, we have created three problem families to evaluate theorem provers for **mCi**, and used these families to compare the performance of **KEMS** strategies for **mCi**. The results obtained are the first benchmark results for these families.

6.1 Further work

It would be useful to have a general procedure for automatically generating correct and complete **KE** systems for **LFI**s and other logical systems, similar to the procedure for generating tableau systems presented in [1]. This could help us to extend **KEMS** to other logical systems. Work in this direction are being studied by some authors of [1]. Having this method would facilitate one to extend **KEMS** to be able to deal with other logical systems.

Acknowledgements

This paper has been partially sponsored by FAPESP Thematic Project Grant ConsRel 2004/14107-2. Marcelo Finger is partly supported by CNPq grant PQ 301294/2004-6.

Bigger instance solved	Problem size	Best time pair	Best size pair
Φ_{20}^7	336	<mcies, and=""></mcies,>	<mciss,nfo></mciss,nfo>
Φ_{50}^8	946	<mciss,or></mciss,or>	<mciss,or></mciss,or>
Φ_{75}^{9}	1197	<mcies,inc></mcies,inc>	<mcies,inc></mcies,inc>

 Table 4. Best mCi strategy-sorter pairs.

References

- Carlos Caleiro, Walter Carnielli, Marcelo E. Coniglio, and Joao Marcos. Two's company: "The humbug of many logical values". In *Logica Universalis*, pages 169– 189. Birkhäuser Verlag, Basel, Switzerland, 2005. Pre-print available at http: //tinyurl.com/yb5qbz. Last accessed, November 2006.
- Walter Carnielli, Marcelo Coniglio, and Ricardo Bianconi. Logic and Applications: Mathematics, Computer Science and Philosophy (in Portuguese). Unpublished, 2005. Preliminary Version.
- Walter Carnielli, Marcelo E. Coniglio, and Joao Marcos. Logics of Formal Inconsistency. In *Handbook of Philosophical Logic*, volume 12. Kluwer Academic Publishers, 2007. To appear. Pre-print available at http://tinyurl.com/ybn4yw. Last accessed, November 2006.
- Walter Carnielli and Mamede Lima-Marques. Reasoning under Inconsistent Knowledge. Journal of Applied Non-Classical Logics, 2(1):49–79, 1992.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms - Second Edition. MIT Press, 2001.
- Newton C. A. da Costa, Décio Krause, and Otávio Bueno. Paraconsistent logics and paraconsistency: Technical and philosophical developments. *CLE e-prints (Section Logic)*, 4(3), 2004. Pre-print available at http://tinyurl.com/yxhon7. Last accessed, November 2006.
- Marcello D'Agostino. Tableau methods for classical propositional logic. In Marcello D'Agostino et al., editor, *Handbook of Tableau Methods*, chapter 1, pages 45–123. Kluwer Academic Press, 1999.
- 8. Marcello D'Agostino and Marco Mondadori. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, pages 285–319, 1994.
- Sandra de Amo, Walter Carnielli, and João Marcos. A Logical Framework for Integrating Inconsistent Information in Multiple Databases. In Thomas Eiter and Klaus-Dieter Schewe, editors, *Lecture Notes in Computer Science*, volume 2284, pages 67–84. Springer-Verlag, Berlim., 2002.
- 10. Itala M. Loffredo D'Ottaviano and Milton Augustinis de Castro. Analytical Tableaux for da Costa's Hierarchy of Paraconsistent Logics $C_n, 1 \le n \le \omega$. Journal of Applied Non-Classical Logics, 15(1):69–103, 2005.
- Adolfo Neto. A Multi-Strategy Tableau Prover. PhD thesis, University of São Paulo, 2007. http://kems.incubadora.fapesp.br/portal/documentos-1/tese. Last accessed, February 2007.
- Adolfo Neto and Marcelo Finger. Effective Prover for Minimal Inconsistency Logic. In Artificial Intelligence in Theory and Practice, IFIP International Federation for Information Processing, pages 465–474. Springer Verlag, 2006. Available at http: //www.springerlink.com/content/b80728w7m6885765. Last accessed, November 2006.
- Adolfo Neto and Marcelo Finger. KEMS A KE Multi-Strategy Tableau Prover, 2006. http://kems.iv.fapesp.br. Last accessed, November 2006.
- 14. Satisfiability library, 2003. http://www.satlib.org. Last accessed, March 22, 2005.
- 15. Raymond M. Smullyan. First-Order Logic. Springer-Verlag, 1968.
- Geoff Sutcliffe. Thousands of problems for theorem provers, 2001. http://www.cs.miami.edu/~tptp. Last accessed, March 2005.
- 17. Geoff Sutcliffe and Christian Suttner. The CADE ATP System Competition, 2003. http://www.cs.miami.edu/~tptp/CASC. Last accessed, March 2005.