Towards an efficient prover for the C_1 paraconsistent logic

Adolfo Neto[1,2]

Informatics Department (DAINF)
Federal University of Technology - Paraná (UTFPR)
Curitiba, Brazil

Celso A. A. Kaestner[3]

Informatics Department (DAINF)
Federal University of Technology - Paraná (UTFPR)
Curitiba, Brazil

Marcelo Finger[1,4]

Computer Science Department (DCC)
University of São Paulo (USP)
São Paulo, Brazil

Abstract

The **KE** inference system is a tableau method developed by Marco Mondadori which was presented as an improvement, in the computational efficiency sense, over Analytic Tableaux. In the literature, there is no description of a theorem prover based on the **KE** method for the C_1 paraconsistent logic. Paraconsistent logics have several applications, such as in robot control and medicine. These applications could benefit from the existence of such a prover. We present a sound and complete **KE** system for C_1, an informal specification of a strategy for the C_1 prover as well as problem families that can be used to evaluate provers for C_1. The C_1 **KE** system and the strategy described in this paper will be used to implement a **KE** based prover for C_1, which will be useful for those who study and apply paraconsistent logics.

Keywords: tableaux systems, **KE** system, C_1 logic, paraconsistent logics, problem families.

1 Introduction

Inconsistency is a phenomena that appears naturally in our world. Consider the following situation: two persons have different (contradictory) opinions about a spe-

[1] Adolfo Neto and Marcelo Finger acknowledge support from Fundaçcão de Amparo a Pesquisa do Estado de São Paulo (FAPESP), Brazil, through the Thematic Project ConsRel, grant number 2004/14107-2. Marcelo Finger also acknowledges an individual research grant from the Brazilian Research Council (CNPq).

[2] Email: adolfo@utfpr.edu.br

[3] Email: celsokaestner@utfpr.edu.br

[4] Email: mfinger@ime.usp.br

cific statement A: the first one considers A true, meanwhile the second one believes that $\neg A$ is true. This contradiction, however, should not prevent that common conclusions which do not involve A – directly or indirectly – can be deduced.

This situation is not adequately managed by classical logic, since it is not equipped to deal with inconsistency. The reason is the well known *"Ex contradictione sequitur quod libet"* principle: if a theory Γ is inconsistent, that is, if formulas A and $\neg A$ are theorems, then every formula B of the language is also a theorem in Γ; or, shortly, Γ becomes *trivial*.

Paraconsistent Logics were initially proposed by Da Costa [8] as logical systems that deal with contradictions in a discriminating way, avoiding the previous principle and managing inconsistent but non-trivial theories.

Presently automatic proof methods are widely used in several computer applications, such as in robot control [23], in medicine [14,16], and many others [10]. Most of the employed methods work on logical formalisms based on classical logic. In this paper we present the specification of an strategy for automatic theorem prover based on a **KE** system, an improvement of the well known tableaux deduction method, for a particular paraconsistent logic called C_1.

The rest of this paper is organized as follows: Section 2 introduces the axiomatization and valuation of the paraconsistent logic C_1; in Sections 3 and 4 we present the **KE** system for C_1 and its inference rules, and the KEMS strategy, respectively; Section 5 presents a set of problems constructed to evaluate the prover; in Section 6 we present a motivating example, showing that our proposal is adequate to deal with practical problems; in Section 7 we compare our work with similar ones; finally in Section 8 we draw some conclusions and propose future research.

We emphasize the main contributions of this paper: (a) a sound and complete **KE** system for C_1 (Section 3); (b) an informal specification of a KEMS [20] strategy for the C_1 prover (Section 4); and (c) problem families that can be used to evaluate provers for C_1 (Section 5).

1.1 Preliminaries

Let $\mathcal{P}$ be a countable set of propositional letters. We concentrate on the propositional language $\mathcal{L}$ formed by the usual boolean connectives $\rightarrow$ (implication), $\wedge$ (conjunction), $\vee$ (disjunction) and $\neg$ (negation). We call Σ this set of connectives: $\Sigma = \{\neg, \wedge, \vee, \rightarrow\}$ (Σ is called a signature in [6]). $\bigwedge_{i=1}^{n}$ and $\bigvee_{i=1}^{n}$ are, respectively, iterated conjuntion and iterated disjunction.

Throughout the paper, we use uppercase Latin or lowercase Greek letters to denote arbitrary formulas, and uppercase Greek letters to denote sets of formulas.

We also work here with signed formulas. A signed formula is an expression $\mathcal{S}\ A$ where $\mathcal{S}$ is called the sign and A is a propositional formula. The symbols $\mathtt{T}$ and $\mathtt{F}$, respectively representing the 'true' and 'false' truth-values, can be used as signs. The conjugate of a signed formula $\mathtt{T}\,A$ ($\mathtt{F}\,A$) is $\mathtt{F}\,A$ ($\mathtt{T}\,A$). The subformulas of a signed formula $\mathcal{S}\ A$ are all the formulas of the form $\mathtt{T}\,B$ or $\mathtt{F}\,B$ where B is a subformula of A.

The size of a signed formula $\mathcal{S}\ A$ is defined as the size of A. The size $s(A)$ of a formula A is defined as usual:

- $s(A) = 1$ if A is a propositional atom;
- $s(\oslash A) = 1 + s(A)$, where A is a formula and $\oslash$ is a unary connective;
- $s(A \oslash B) = 1 + s(A) + s(B)$, where $\oslash$ is a binary connective, and A and B are formulas.

A propositional valuation v is a function $v : \mathcal{P} \rightarrow \{0, 1\}$. We extend the definition of valuations to signed formulas in the following way: $v(\mathtt{T}\,A) = v(A)$ and $v(\mathtt{F}\,A) = 1 - v(A)$.

2 C_1, a paraconsistent logic

C_1 is a paraconsistent logic [8], "a logic of the early paraconsistent vintage" [6]. It is part of the hierarchy of logics C_n, $1 \leq n < \omega$ [10]. C_1 is of historical importance because it was one of the first paraconsistent logics to be presented.

Paraconsistent logics are logics in which theories can be inconsistent but non-trivial [10]. In classical logic, $A \wedge \neg A \vdash B$ for any formulas A and B. This is not true in paraconsistent logics.

In C_1, a consistency operator ($\circ$) is introduced. The intended meaning of $\circ A$ is "A is consistent" [6]. According to [6], "da Costa's intuition was that the 'consistency' (which he dubbed 'good behavior') of a given formula would not only be a sufficient requisite to guarantee its explosive character, but that it could also be represented as an *ordinary formula* of the underlying language."

In C_1, da Costa represented the consistency of a formula A by the formula $\neg(A \wedge \neg A)$. That is, the consistency connective "$\circ$" is not a primitive connective, but an abbreviation:

$$\circ A \stackrel{\text{def}}{=} \neg(A \wedge \neg A).$$

2.1 C_1's Axiomatization

Some axiomatizations for C_1 were presented in the literature [6,9,15]. The presentation below is based on [6] and [9].

Axiom schemas:

(Ax1) $\alpha \rightarrow (\beta \rightarrow \alpha)$

(Ax2) $(\alpha \rightarrow \beta) \rightarrow ((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow (\alpha \rightarrow \gamma))$

(Ax3) $\alpha \rightarrow (\beta \rightarrow (\alpha \wedge \beta))$

(Ax4) $(\alpha \wedge \beta) \rightarrow \alpha$

(Ax5) $(\alpha \wedge \beta) \rightarrow \beta$

(Ax6) $\alpha \rightarrow (\alpha \vee \beta)$

(Ax7) $\beta \rightarrow (\alpha \vee \beta)$

(Ax8) $(\alpha \rightarrow \gamma) \rightarrow ((\beta \rightarrow \gamma) \rightarrow ((\alpha \vee \beta) \rightarrow \gamma))$

(Ax10) $\alpha \vee \neg\alpha$

(Ax11) $\neg\neg\alpha \rightarrow \alpha$

(bc1) $\circ\alpha \rightarrow (\alpha \rightarrow (\neg\alpha \rightarrow \beta))$

(ca1) $(\circ\alpha \wedge \circ\beta) \to \circ(\alpha \wedge \beta)$

(ca2) $(\circ\alpha \wedge \circ\beta) \to \circ(\alpha \vee \beta)$

(ca3) $(\circ\alpha \wedge \circ\beta) \to \circ(\alpha \to \beta)$

Inference rule:

(MP) $\dfrac{\alpha, \alpha \to \beta}{\beta}$

The difference from classical propositional logic (**CPL**) axiomatization is that to obtain an axiomatization for **CPL** we must remove the schemas that deal with the consistency connective (**(bc1), (ca1), (ca2)** and **(ca3)**) and add the following axiom schema (called 'explosion law' in [6]):

(exp) $\alpha \to (\neg\alpha \to \beta)$

2.2 C_1's Valuation

C_1 received a bivaluation semantics in [9] (see also [6]). A set of clauses characterizing C_1-valuations (adapted from the one in [6]) is the following:

- $v(\alpha_1 \wedge \alpha_2) = 1$ if and only if $v(\alpha_1) = 1$ and $v(\alpha_2) = 1$;
- $v(\alpha_1 \vee \alpha_2) = 1$ if and only if $v(\alpha_1) = 1$ or $v(\alpha_2) = 1$;
- $v(\alpha_1 \to \alpha_2) = 1$ if and only if $v(\alpha_1) = 0$ or $v(\alpha_2) = 1$;
- $v(\neg\alpha) = 0$ implies $v(\alpha) = 1$;
- $v(\neg\neg\alpha) = 1$ implies $v(\alpha) = 1$;
- $v(\circ\alpha) = 1$ implies $v(\alpha) = 0$ or $v(\neg\alpha) = 0$.
- $v(\circ(\alpha \oslash \beta)) = 0$ implies $v(\circ\alpha) = 0$ or $v(\circ\beta) = 0$, for $\oslash \in \{\wedge, \vee, \to\}$;

Definition 2.1 Let Γ be $\{A_1, A_2, \ldots, A_n\}$ for $n \geq 0$. $\Gamma \vdash B$ is a valid sequent in C_1 if and only if, $v(B) = 1$ whenever $v(A_i) = 1$ for all i $(1 \leq i \leq n)$. "$\Gamma \vdash B$ is a valid sequent in C_1" can be abbreviated to $\Gamma \vdash_{C_1} B$.

3 The KE System for C_1

The **KE** inference system is a tableau method [13] developed by Marco Mondadori and discussed in detail in several works authored or co-authored by Marcello D'Agostino [2,11,12]. The **KE** system was presented as an improvement, in the computational efficiency sense, over Analytic Tableaux [22]. A **KE** System is a tableau system in which there is only one branching rule. As branching can lead to repetition of efforts (i.e. the same work being done in two or more branches), branching rules lead to less efficient proof systems (and implementations) [12].

We present here a sound and complete **KE** System we have devised for C_1. The rules in our system are presented in Figure 1. Note that in C_1, the set of connectives is $\Sigma = \{\neg, \wedge, \vee, \to\}$ but, to make the rules simpler, we have used the connectives in $\Sigma^\circ = \Sigma \cup \{\circ\}$, i.e. including the consistency connective, which is actually an abbreviation.

In [19,21] (and also in [18]) the first and third authors of this paper have pre-

$$\frac{\mathtt{F}\,A \to B}{\begin{array}{c}\mathtt{T}\,A\\ \mathtt{F}\,B\end{array}}\ (\mathtt{F}{\to}) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \to B\\ \mathtt{T}\,A\end{array}}{\mathtt{T}\,B}\ (\mathtt{T}{\to}_1) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \to B\\ \mathtt{F}\,B\end{array}}{\mathtt{F}\,A}\ (\mathtt{T}{\to}_2)$$

$$\frac{\mathtt{T}\,A \wedge B}{\begin{array}{c}\mathtt{T}\,A\\ \mathtt{T}\,B\end{array}}\ (\mathtt{T}\wedge) \qquad \frac{\begin{array}{c}\mathtt{F}\,A \wedge B\\ \mathtt{T}\,A\end{array}}{\mathtt{F}\,B}\ (\mathtt{F}\wedge_1) \qquad \frac{\begin{array}{c}\mathtt{F}\,A \wedge B\\ \mathtt{T}\,B\end{array}}{\mathtt{F}\,A}\ (\mathtt{F}\wedge_2)$$

$$\frac{\mathtt{F}\,A \vee B}{\begin{array}{c}\mathtt{F}\,A\\ \mathtt{F}\,B\end{array}}\ (\mathtt{F}\vee) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \vee B\\ \mathtt{F}\,A\end{array}}{\mathtt{T}\,B}\ (\mathtt{T}\vee_1) \qquad \frac{\begin{array}{c}\mathtt{T}\,A \vee B\\ \mathtt{F}\,B\end{array}}{\mathtt{T}\,A}\ (\mathtt{T}\vee_2)$$

$$\frac{\mathtt{F}\,\neg A}{\mathtt{T}\,A}\ (\mathtt{F}\neg) \qquad \frac{\mathtt{T}\,\neg\neg A}{\mathtt{T}\,A}\ (\mathtt{T}\neg\neg)$$

$$\frac{\begin{array}{c}\mathtt{T}\circ A\\ \mathtt{T}\,\neg A\end{array}}{\mathtt{F}\,A}\ (\mathtt{T}\circ\neg) \qquad \frac{\begin{array}{c}\mathtt{F}\circ(A \oslash B)\\ \mathtt{T}\circ A\end{array}}{\mathtt{F}\circ B}\ (\mathtt{F}\circ\oslash_1) \qquad \frac{\begin{array}{c}\mathtt{F}\circ(A \oslash B)\\ \mathtt{T}\circ B\end{array}}{\mathtt{F}\circ A}\ (\mathtt{F}\circ\oslash_2)$$

$$\begin{array}{c}\diagup\!\diagdown\\ \mathtt{T}\,A \quad \mathtt{F}\,A\end{array}\ (PB)$$

Fig. 1. C_1 **KE** rules.

sented **KE** Systems for two other paraconsistent logics: **mbC** and **mCi** (more about these two logics can be found in [6]). The **KE** System for C_1 has several rules in common with the **KE** Systems for these two logics. However, in these logics, consistency ($\circ$) is not a defined connective.

As in classical **KE** rules [12], rules with "1" (for instance, "$\mathtt{F}\wedge_1$") or "2" as subscript are interchangeable. Only one of each pair is actually essential. By using the **PB** rule (Figure 1), the other can be derived.

Note also that $\mathtt{F}\circ\oslash_1$ and $\mathtt{F}\circ\oslash_2$ are actually three rules each, because $\oslash$ can be any connective in $\{\wedge, \vee, \to\}$. In a 2-premiss rule, the *main premiss* is the first premiss. The second premiss is called *minor premiss*. The main premiss in $\mathtt{F}\circ\oslash_1$ (or in $\mathtt{F}\circ\oslash_2$) can be "$\mathtt{F}\circ(A \wedge B)$", "$\mathtt{F}\circ(A \vee B)$" or "$\mathtt{F}\circ(A \to B)$". And, as '$\circ$' is a defined connective, "$\mathtt{F}\circ(A \oslash B)$" is actually "$\mathtt{F}\,\neg((A \oslash B) \wedge \neg(A \oslash B)$". For instance, $\mathtt{F}\circ\wedge_1$ is:

$\mathtt{F}\ \neg(P \wedge (\neg P \wedge \circ P))$
$\mathtt{T}\ P \wedge (\neg P \wedge \circ P)$
$\mathtt{T}\ P$
$\mathtt{T}\ \neg P \wedge \circ P$
$\mathtt{T}\ \neg P$
$\mathtt{T}\ \circ P$
$\mathtt{F}\ P$
$\times$

Fig. 2. A proof of $\neg(P \wedge (\neg P \wedge \circ P))$ using the C_1 **KE** system.

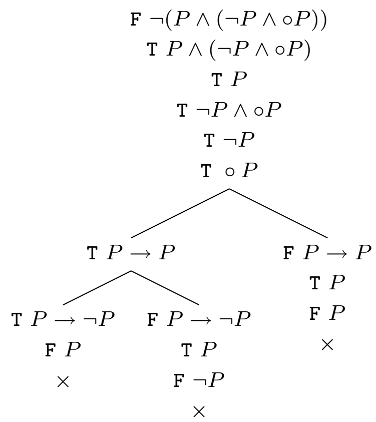

Fig. 3. A proof of $\neg(P \wedge (\neg P \wedge \circ P))$ [6].

$$\frac{\mathtt{F}\,\neg((A \wedge B) \wedge \neg(A \wedge B)) \qquad \mathtt{T}\,\neg(A \wedge \neg A)}{\mathtt{F}\,\neg(B \wedge \neg B)}\ (\mathtt{F}\circ\wedge_1)$$

It is easy to see that these rules ($\mathtt{F}\circ\oslash_1$ and $\mathtt{F}\circ\oslash_2$) are not analytic. In $\mathtt{F}\circ\wedge_1$, $\mathtt{F}\neg(B \wedge \neg B)$ is not a subformula of any premiss.

Therefore, in our system we have:

- 12 essential linear rules (5 of these rules are 1-premiss rules and 7 rules are 2-premiss rules);
- 6 derived linear 2-premiss rules;
- 1 (0-premiss) branching rule.

Of these rules, 6 of them ($\mathtt{F}\circ\oslash_1$ and $\mathtt{F}\circ\oslash_2$) are rather complex, far more complex than any **CPL KE** rule.

Example 3.1 The formula $\neg(P \wedge (\neg P \wedge \circ P))$ can be proved in C_1 **KE** system as depicted in Figure 2. The same formula was proved in [6] using the C_1 tableau system presented there (Figure 3). It is easy to see that the C_1 **KE** proof has less formula nodes and less branches than the C_1 tableau [6] proof.

3.1 Soundness and Completeness

Our intention here is to prove that the C_1 **KE** system is sound and complete. The proof is very similar to the **mCi KE** system's soundness and completeness proof presented in Section B.2.4 of [18]. We begin with some definitions.

Definition 3.2 [12] A branch of a **KE** tableau is closed when $\mathtt{T}\,A$ and $\mathtt{F}\,A$ appear in the branch.

Definition 3.3 [12] A **KE** tableau is closed if all its branches are closed.

Definition 3.4 $\Gamma \vdash_{C_1\mathbf{KE}} B$ if there is a closed **KE** tableau for $\Gamma \vdash B$.

Definition 3.5 The C_1 **KE** system is sound if, for any Γ and B, $\Gamma \vdash_{C_1\mathbf{KE}} B$ implies $\Gamma \vdash_{C_1} B$.

Definition 3.6 The C_1 **KE** system is complete if, for any Γ and B, $\Gamma \vdash_{C_1} B$ implies $\Gamma \vdash_{C_1\mathbf{KE}} B$.

Definition 3.7 A set of C_1 signed formulas DS is *downward saturated*:

(i) whenever a signed formula is in DS, its conjugate is *not* in DS;

(ii) when all premises of any C_1 **KE** rule (except PB) are in DS, its conclusions are also in DS;

(iii) when the major premiss of a 2-premiss C_1 **KE** rule is in DS, either its auxiliary premiss or its conjugate is in DS.

A Hintikka's Lemma holds for C_1 downward saturated sets:

Lemma 3.8 *(Hintikka's Lemma for C_1) Every C_1 downward saturated set is satisfiable.*

Proof. For any downward saturated set DS, we can easily construct a C_1 valuation v such that for every signed formula $\mathcal{S}X$ in the set, $v(\mathcal{S}X) = 1$. How can we guarantee this is in fact a valuation? First, we know that there is no pair $\mathtt{T}\,X$ and $\mathtt{F}\,X$ in DS. Second, all premised C_1 **KE** rules preserve valuations. That is, if $v(\mathcal{S}X_i) = 1$ for every premiss $\mathcal{S}X_i$, then $v(\mathcal{S}C_j) = 1$ for all conclusions C_j. And if $v(\mathcal{S}X_1) = 1$ and $v(\mathcal{S}X_2) = 0$, where X_1 and X_2 are, respectively, major and minor premises of a C_1 **KE** rule, then $v(\mathcal{S}'X_2) = 1$, where $\mathcal{S}'X_2$ is the conjugate of $\mathcal{S}X_2$. Therefore, DS is satisfiable. □

Theorem 3.9 *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' $\subseteq$ DS". Using items (ii) and (iii) of (3.7), we can obtain a family of sets of signed formulas DS'_i ($i \geq 1$) that include DS'. If none of them is downward saturated, it is because for all i, $\{\mathtt{T}\,X, \mathtt{F}\,X\} \in DS'_i$ for some X. But all rules are valuation-preserving, so this can only happen if DS is unsatisfiable, which is a contradiction. □

Corollary 3.10 *DS' is an unsatisfiable set of formulas if and only if there is no downward saturated set DS" such that DS' ⊆ DS".*

Theorem 3.11 *The C_1 **KE** system is sound and complete.*

Proof. The C_1 **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 C_1 **KE** system is a refutation system. The C_1 **KE** system is sound because if a C_1 **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 C_1 **KE** system is complete because if S is satisfiable, no C_1 **KE** tableau for a set of formulas S closes. And if S is unsatisfiable, all completed C_1 **KE** tableaux for S close. □

3.2 Decidability

We do not prove here that the C_1 **KE** system is decidable, i.e., that there is an algorithm for finding proofs in the C_1 **KE** system. We only present the sketch of such a proof that will be detailed in a future paper about the implementation of a C_1 prover.

The idea is to define a restriction of the C_1 **KE** system which imposes some conditions on the application the PB rule (Figure 1). In this *restricted* C_1 **KE** system, the PB rule can only be applied in a branch:

- when there is a non-atomic signed formula that can be the main premiss of a 2-premiss rule and that was not yet analysed (i.e. used as main premiss) in the branch; and
- when either $\mathtt{T}\,A$ or $\mathtt{F}\,A$ can be the minor premiss of a 2-premiss rule, where A is the PB formula (i.e. the A formula that appears as $\mathtt{T}\,A$ in the new left branch and $\mathtt{F}\,A$ in the new right branch after PB application).

For all the 2-premiss rules in Figure 1, the minor premiss's size is smaller than major premiss's size. This, alongside with the conditions above, guarantees the the proof search procedure eventually terminates.

4 A KEMS Strategy for C_1

KEMS [18] is a theorem prover that can be used to implement strategies for many different logical systems. For instance, the current version [20] has 6 strategies for **CPL**, 2 strategies for **mbC** and 2 strategies for **mCi**.

We have to follow some steps to implement a strategy for a logical system in KEMS. First, one has to know how KEMS implementation is structured (by reading

[18] and the source code available in [20]). Second, one has to implement the classes that will represent the logical system (such as **CPL** or C_1). Third, one has to implement the classes necessary to represent the rules of the **KE** system (such as C_1 **KE** system). Only after these three steps, one can implement one or more strategies for a given **KE** system.

KE systems (as well as many logical proof methods) are usually presented by showing their rules. The rules tells us only what we can do – they do not specify in which order to use the rules. A strategy is a deterministic algorithm for a given **KE** system, as well as a set of data structures used by the algorithm.

4.1 C_1 **KE** *Simple Strategy*

The C_1 **KE** Simple Strategy resembles **mbC** and **mCi** Simple Strategies (see Sections C.4.4 and C.4.5 of [18]). Let us informally describe the algorithm performed by this strategy:

(i) the strategy applies all possible linear rules in the current branch (in the beginning, the current branch is the branch containing the formulas obtained from the problem);

(ii) if the current branch closes (i.e. if a contradiction $\{\mathtt{T}\, A, \mathtt{F}\, A\}$ is found), then the strategy tries to remove a branch from its *stack of open branches*. If it succeeds, this branch becomes the current branch and the control goes back to the first step. If there is no remaining open branch, the procedure ends and the result is that the tableau is declared **closed**;

(iii) if the current branch is linearly saturated (i.e. no more linear rules can be applied), but not closed, the strategy tries to apply the PB rule. The PB rule can be applied when there is at least one non-atomic signed formula in the branch that can be the main premiss of a 2-premiss rule and this signed formula was not yet used as the main premiss in an application of a 2-premiss rule. If the strategy can apply the PB rule, then the (new) right branch is put in the stack of open branches and the left branch becomes the current branch. If the strategy cannot apply the PB rule, then the procedure finishes by declaring the tableau **open**.

The order of rule applications is:

(i) C_1 **KE** 1-premiss rules;

(ii) C_1 **KE** 2-premiss rules;

(iii) the PB rule.

See Sections C.2 and C.4 of [18] for more details on how rules are applied in KEMS.

4.1.1 Implementation Remarks

This strategy is a very straightforward strategy for a C_1 **KE** system. The idea is to use the PB rule only as a last resource (as shown in the canonical procedure for **KE** [12]). The difference is that in the C_1 **KE** system we cannot restrict the strategy to perform only analytic applications of PB. An analytic application of PB is an application of PB where the PB formula (i.e. the A formula that appears as

$\mathtt{T}\,A$ in the new left branch and $\mathtt{F}\,A$ in the new right branch after PB application) is a subformula of some formula in the branch.

Another difficulty in the implementation of this strategy (actually in the implementation of almost any proof system for C_1) is how to deal with the consistency connective.

We have two options:

(i) only accept problems using the connectives in Σ. Therefore, all rules presented in Figure 1 will have to be implemented using Σ connectives (which makes the rules and the associated pattern matching more complex). Note that the size of problems written in Σ° may grow exponentially (in the worst case) when translated to Σ;

(ii) accept problems written in Σ° and, whenever a $\neg(A \wedge \neg A)$ formula appears (for any A), treat it as if it was (also) $\circ A$ in the applications of rules that have formulas with $\circ$ as premisses. Although this option allows the prover to deal with smaller problems, it makes rule applications more difficult.

C_1 Simple Strategy will use option (i) above. Option (ii) will be used on a second strategy for C_1 **KE**.

5 Problem Families to Evaluate C_1 Provers

A problem family is a set of problems that we know, by construction, whether they are valid, satisfiable or unsatisfiable [18]. A problem is a sequent that can be given as input for a theorem prover. The i-th instance (for $i \geq 1$) of a problem family is a (valid, satisfiable or unsatisfiable) sequent.

In Section D.1.2 of [18], seven families of difficult problems that can be used to evaluate theorem provers for paraconsistent logics were presented. All these families were families of valid sequents. To the best of our knowledge, there are no other families of difficult problems designed with this purpose in mind. The families presented there can be used to evaluate provers for two logics: **mbC** and **mCi**, which are part of the class of logics of formal inconsistency (LFIs) [6].

In [6] it is shown that C_1 can also be classified as an LFI and that it extends **mbC**. Therefore, the first four families created to evaluate **mbC** provers [18] can also be used to evaluate provers for C_1.

However, these families do not test all C_1 **KE** rules. That is, to prove the problems in those families using the C_1 **KE** system, one does not need to use all its rules. Therefore, to extend what we could call "rule coverage", i.e. to test more rules, we present two more families of sequents. These sequents are valid in C_1 (but not in **mbC**, therefore they can also be used to test **mbC** provers) and, for proving them, we have to use rules that are not used in the first four families' proofs.

These families were not developed with any intuitive meaning in sight. As the objective was to test theorem provers, they were designed to be difficult to prove, by using as many rules as possible.

The motivation for developing and presenting these problem families *before* the actual C_1 prover was implemented was, inspired by the Test-Driven Development technique for software development [1], to use the tests as a guide for the design and implementation of the sofwtare.

Note: to make it easier to read the problems, we have used the connectives in Σ° and we sometimes use "[" and "]" in place of "(" and ")".

5.1 Fifth family

The sequents in this family (Φ^5) demand C_1's $\texttt{T}\neg\neg$ rule to be proven valid. Φ^5_n (the nth instance of Φ^5) is:

$$\circ A_1, \bigwedge_{i=1}^{n}(A_i), \bigwedge_{i=1}^{n}[A_{n+1} \to ((A_i \vee B_i) \to (\circ A_{i+1}))], (\bigwedge_{i=1}^{n} \circ A_i) \to \neg A_{n+1} \vdash \neg\neg\neg A_{n+1}$$

For instance, Φ^5_3 in signed tableau notation is:

$$\texttt{T}\ \circ A_1$$

$$\texttt{T}\ A_1 \wedge A_2 \wedge A_3$$

$$\begin{aligned}\texttt{T}\ & [A_4 \to ((A_1 \vee B_1) \to (\circ A_2))] \\ & \wedge [A_4 \to ((A_2 \vee B_2) \to (\circ A_3))] \\ & \wedge [A_4 \to ((A_3 \vee B_3) \to (\circ A_4))]\end{aligned}$$

$$\texttt{T}\ ((\circ A_1) \wedge (\circ A_2) \wedge (\circ A_3)) \to \neg A_4$$

$$\texttt{F}\ \neg\neg\neg A_4$$

5.2 Sixth family

In order to prove, using the C_1 **KE** system, that the sequents in this family (Φ^6) are valid, it is necessary to use the two C_1 **KE** rules where "$\circ$" is the main connective in the main premiss: $\texttt{F}\circ\oslash_1$ and $\texttt{F}\circ\oslash_2$. Φ^6_n (the n-th instance of Φ^6) is:

$$\bigwedge_{i=1}^{n}(B_i), \bigwedge_{i=1}^{n}(\circ C_i), \bigwedge_{i=1}^{n}((A_i \vee B_i) \to (\circ A_{i+1})), (\bigwedge_{i=1}^{n} C_i) \to (D \wedge \neg C_1) \vdash [\bigvee_{i=1}^{n}(\circ(A_{i+1} \to C_i))] \vee D$$

For instance, Φ^6_3 is:

$$\texttt{T}\ B_1 \wedge B_2 \wedge B_3$$

$$\texttt{T}\ \circ C_1 \wedge \circ C_2 \wedge \circ C_3$$

$$\texttt{T}\ ((A_1 \vee B_1) \to (\circ A_2)) \wedge ((A_2 \vee B_2) \to (\circ A_3)) \wedge ((A_3 \vee B_3) \to (\circ A_4))$$

$$\texttt{T}\ (C_1 \wedge C_2 \wedge C_3) \to (D \wedge \neg C_1)$$

$$\texttt{F}\ [\circ(A_2 \to C_1)] \vee [\circ(A_3 \to C_2)] \vee [\circ(A_4 \to C_3)] \vee D$$

6 A Motivating Example

We present here an example almost completely based on the example shown in [17]:

Consider the construction of a simple medical system aimed at diagnosing three diseases K, L and M. There are two different symptoms, denoted by N and O. The intended usage of this system is as follows:

- The core part of the system is the knowledge provided by a doctor (DOC_1).
- When we intend to apply this knowledge to a specific patient, other professionals conduct medical tests on this patient add the results of these tests to the knowledge base.
- In order to use the system, we submit a goal to the program in a similar way as it is done in Prolog.

We assume that the system is written in the form of a finite set of formulas over C_1. Suppose that DOC_1 provided us the following five rules (formulas):

(F_1) $K \to \neg L$
(F_2) $L \to \neg K$
(F_3) $K \to M$
(F_4) $N \to K$
(F_5) $O \to L$

Intuitively, the doctor is telling that:

- An individual cannot have both diseases K and L (F_1 and F_2).
- If an individual has the disease K, them he has the disease M (F_3)
- If an individual has the symptom N, them he has the disease K (F_4)
- If an individual has the symptom O, them he has the disease L (F_5)

To exemplify the use of this knowledge base, we describe four situations. The first one is similar to a query to a Prolog program, while the other three explore the capacity of handling inconsistencies:

Case 1: Suppose that the patient has symptom N and we want to know if he has the disease K but not L.

To answer this query we must verify if

$$F_1, F_2, F_3, F_4, F_5, N \vdash_{C_1} K \wedge \neg L$$

is valid. As the **KE** proof for this sequent is a closed tableau, this sequent is valid. It is also valid in classical logic.

Case 2: Now suppose that the patient tested positive for symptoms N and O, and we want to know if he has both diseases K and L.

To answer this query we must verify if

$$F_1, F_2, F_3, F_4, F_5, N, O \vdash K \wedge L \tag{1}$$

is valid. (1) is valid in C_1. In classical propositional logic, (1) is also valid. Actually, in classical propositional logic:

$$F_1, F_2, F_3, F_4, F_5, N, O \vdash B \tag{2}$$

is valid for any formula B. However, (2) is not valid in C_1 for any formula B .

Case 3: Now suppose that the patient tested positive for symptoms N and O, and we want to know if he has *not* the disease M.

To answer this query we must verify if

$$F_1, F_2, F_3, F_4, F_5, N, O \vdash_{C_1} \neg M \quad (3)$$

is valid. The **KE** proof for this sequent (an open tableau, which shows that the sequent is NOT valid) is the following:

$$\begin{array}{c} \mathtt{T}\, K \to \neg L \\ \mathtt{T}\, L \to \neg K \\ \mathtt{T}\, K \to M \\ \mathtt{T}\, N \to K \\ \mathtt{T}\, O \to L \\ \mathtt{T}\, N \\ \mathtt{T}\, O \\ \mathtt{F}\, \neg M \\ \hline \mathtt{T}\, K \\ \mathtt{T}\, L \\ \mathtt{T}\, M \\ \mathtt{T}\, \neg K \\ \mathtt{T}\, \neg L \end{array}$$

However, this sequent is valid in classical logic, because a classical contradiction is found ($\mathtt{T}\, K$ and $\mathtt{T}\, \neg K$).

Case 4: Now suppose again that the patient tested positive for symptoms N and O, but now we want to know if he has the disease K and if this conclusion is not consistent ($\neg \circ K$).

To answer this query we must verify if

$$F_1, F_2, F_3, F_4, F_5, N, O \vdash_{C_1} K \wedge \neg \circ K \quad (4)$$

is valid. The **KE** proof for this sequent (a closed tableau) is the following:

$$\begin{array}{c} \mathtt{T}\, K \to \neg L \\ \mathtt{T}\, L \to \neg K \\ \mathtt{T}\, K \to M \\ \mathtt{T}\, N \to K \\ \mathtt{T}\, O \to L \\ \mathtt{T}\, N \\ \mathtt{T}\, O \\ \mathtt{F}\, K \wedge \neg \circ K \\ \hline \mathtt{T}\, K \\ \mathtt{T}\, L \\ \mathtt{T}\, \neg L \\ \mathtt{T}\, \neg K \\ \mathtt{T}\, M \\ \mathtt{F}\, \neg \circ K \\ \mathtt{T}\, \circ K \\ \mathtt{F}\, K \\ \times \end{array}$$

This query shows that, besides "dealing with inconsistencies in the knowledge base without every formula becoming derivable" [17], a common feature of paraconsistent logics, C_1 allows us to express propositions about the (in)consistency of formulas.

The sequent (4) is valid in classical logic. Note that, for any formula B, "$\neg \circ B$" is a theorem in classical logic. Therefore, in classical logic, $\Gamma \vdash K \wedge \neg \circ K$ if and only $\Gamma \vdash K$.

7 Related Work

A tableau system for C_1 was presented in [7]. As this system is based on analytic tableaux (AT) [22], it has four branching rules: the three ones from AT plus a $\mathtt{T}\,\neg$ branching rule. Due to this $\mathtt{T}\,\neg$ rule, infinite loops may occur during the proof search, postponing indefinitely the analysis of formulas that involve the negation and consistency operators. Notwithstanding, this system is decidable. This system has been implemented but the source code is not available.

In [4] two tableau systems for C_1 were presented, the second one being a version of the first one considered by the authors more adequate to be implemented. The first system has 12 rules (8 of them are branching rules) while the second has 20 rules (12 of them are branching rules). The rules are rather complex, involving much more formulas and connectives than C_1 **KE** rules. The second system was elegantly implemented in LISP (the source code is available in [3]). However it was written in a LISP dialect (muLISP) which cannot be compiled in modern LISP compilers.

We have experimented using Buchsbaum's system with the formulae described in Section 5. For example, it was not able to prove instance Φ^5_{27} due to lack of memory. This confirms that the family Φ^5 is a family of difficult problems. We are translating this prover to a different LISP dialect to make it more robust.

Another C_1 tableau system appears in [6]. It was obtained by using a general method for constructing tableau systems [5]. Although this system has a PB branching rule, a feature of **KE** systems, is not a **KE** system. To be a **KE** system it should have only one branching rule, but it has 8 branching rules. Just like the system in [7], it is based on AT. However, it does not have rules that lead to infinite loops. We do not know of any implementation of this method.

In [15], tableau systems for several logics of the C_n hierarchy were presented. The C_1 tableau system presented there is also based on AT. While in the previous systems $\circ A \stackrel{\text{def}}{=} \neg(A \wedge \neg A)$ was applied whenever necessary to generate the branches of the tableau, this system has specific rules to directly deal with all operators, including "$\circ$". However, as it is based on the analytic tableau method, it also has too many (six) branching rules. We also do not know of any implementation of this method.

Therefore, the distinctive feature of our C_1 **KE** system is that it has 13 essential rules and only of them is a branching rule. This feature will allow us to implement efficient strategies for this system in KEMS [20].

8 Conclusion

In this paper, we have presented a sound and complete **KE** system for Da Costa's C_1 calculus for paraconsistent logic. We have shown that our system has less branching rules than other tableau systems for C_1 described in the literature [3,4,6,7,15].

Therefore, it is probably more efficient than those systems (see [12] for a discussion on why branching leads to inefficiency).

We have also described a strategy for this **KE** system that can be implemented in KEMS. Future work includes implementing this strategy, as well as designing and implementing other strategies for the C_1 **KE** system.

In order to evaluate C_1 **KE** strategies, we have developed two problem families. These families and the first four problem families described in section D.1.2 of [18] can also be used to evaluate other theorem provers for C_1, such as Arthur Buchsbaum's prover for C_1 [3].

As further work, we intend to compare the results obtained by our strategies (in the style of section D.2 of [18]) among themselves as well as with Arthur Buchsbaum's prover.

References

[1] Kent Beck. *Test Driven Development: By Example.* Addison-Wesley Professional, November 2002.

[2] Krysia Broda, Marcello D'Agostino, and Marco Mondadori. A Solution to a Problem of Popper. In *Proceedings of the conference Karl Popper Philosopher of Science*, 1995. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.7542. Last accessed, June 2009.

[3] Arthur Buchsbaum. An automatic proof method for paraconsistent logic (in portuguese), 1988. Available at http://migre.me/gQD. Last accessed, Mar 2009.

[4] Arthur Buchsbaum and Tarcisio Pequeno. A reasoning method for a paraconsistent logic. *Studia Logica*, 52(2):281–289, June 1993.

[5] Carlos Caleiro, Walter Carnielli, Marcelo Coniglio, and Joao Marcos. Two's company: The humbug of many logical values. In *Logica Universalis*, pages 169–189. Birkhauser Basel, 2005.

[6] Walter Carnielli, Marcelo E. Coniglio, and Joao Marcos. *Handbook of the Philosophical Logic*, volume 14, chapter Logics of Formal Inconsistency, pages 15–107. Springer-Verlag, second edition, 2007.

[7] Walter Alexandre Carnielli and Mamede Lima-Marques. Reasoning under inconsistent knowledge. *Journal of Applied Non-Classical Logics*, 2(1), 1992.

[8] Newton C. A. da Costa. *Sistemas Formais Inconsistentes.* Rio de Janeiro, NEPE, 1963. Reprinted by Editora da UFPR, Curitiba, 1993.

[9] Newton C. A. da Costa and E. H. Alves. A semantical analysis of the calculi C_n. *Notre Dame Journal of Formal Logic*, 18(4):621–630, 1977. Available at http://migre.me/gMA. Last accessed, Mar 2009.

[10] Newton C. A. da Costa, Decio Krause, and Otavio Bueno. *Handbook of the Philosophy of Science. Philosophy of Logic*, chapter Paraconsistent Logics and Paraconsistency, pages 791–911. Elsevier, 2007.

[11] Marcello D'Agostino. Are Tableaux an Improvement on Truth-Tables? Cut-Free proofs and Bivalence. *Journal of Logic, Language and Information*, pages 235–252, 1992. Available at http://citeseer.nj.nec.com/140346.html. Last accessed, May 2005.

[12] 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.

[13] 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.

[14] Fabio Romeu de Carvalho, Israel Brunstein, and Jair Minoro Abe. Prevision of Medical Diagnosis Based on Paraconsistent Annotated Logic. *International Journal of Computing Anticipatory Systems*, 18:288–297, 2005.

[15] Itala M. Loffredo D'Ottaviano and Milton Augustinis de Castro. Analytical tableaux for da costa's hierarchy of paraconsistent logics. *Electronic Notes in Theoretical Computer Science*, 143:27 – 44, 2006. Proceedings of the 12th Workshop on Logic, Language, Information and Computation (WoLLIC 2005).

[16] Fahim T. Imam, Wendy MacCaull, and Margaret Ann Kennedy. Merging healthcare ontologies: Inconsistency tolerance and implementation issues. *Proceedings of the Twentieth IEEE International Symposium on Computer-Based Medical Systems*, pages 530–535, 2007.

[17] Decio Krause, Emerson Faria Nobre, and Martin Musicante. Bibel's matrix connection method in paraconsistent logic: general concepts and implementation. In *Proceedings of the XXI International Conference of the Chilean Computer Science Society*, pages 161–167, 2001.

[18] Adolfo Neto. *A Multi-Strategy Tableau Prover*. PhD thesis, University of Sao Paulo, 2007. Available at http://www.dainf.ct.utfpr.edu.br/~adolfo/Thesis/. Last accessed, Mar 2009.

[19] Adolfo Neto and Marcelo Finger. Effective Prover for Minimal Inconsistency Logic. In *Artificial Intelligence in Theory and Practice*, IFIP, pages 465–474. Springer Verlag, 2006. Available at http://www.springerlink.com/content/b80728w7m6885765. Last accessed, November 2006.

[20] Adolfo Neto and Marcelo Finger. *KEMS - A* **KE**-*based Multi-Strategy Tableau Prover*, 2006. http://www.dainf.ct.utfpr.edu.br/~adolfo/KEMS. Last accessed, April 2009.

[21] Adolfo Neto and Marcelo Finger. A KE tableau for a logic of formal inconsistency. In *Proceedings of TABLEAUX'07 position papers and Workshop on Agents, Logic and Theorem Proving. Technical Report (LSIS.RR.2007.002) of the LSIS/Université Paul Cézanne*, Marseille, France, 2007.

[22] Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.

[23] Cláudio Rodrigo Torres, Germano Lambert-Torres, Luiz Eduardo Borges da Silva, and Jair Minoro Abe. Intelligent system of paraconsistent logic to control autonomous moving robots. In *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on*, pages 4009–4013, Nov. 2006.