Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 1. pp. 65–74.

Redundancy for rigid clausal tableaux

Gergely Kovásznai

Faculty of Informatics, University of Debrecen, Hungary
e-mail: kovasz@inf.unideb.hu

Abstract

Redundancy is a substantial concept when avoiding endless computation in theorem proving methods. In this paper, we are targeting a special class of tableau calculi, namely *rigid clausal tableaux.* These tableaux are (1) clausal in the sense that they consist solely of literals, and are (2) rigid in the sense that they employ rigid variables. Such calculi are e.g. rigid hyper tableaux [7] and multi-hyper (hyperS) tableaux [6]. In this paper, we construct a general redundancy concept for rigid clausal tableaux. As a warm-up, we deduce the so-called Sufficient Redundancy Criterion for hyper tableaux [1], in order to demonstrate how rigid variables complicate the redundancy check. Finally, we define what a *redundant clause* w.r.t. a rigid clausal tableau means, and we prove this redundancy concept to be sound. We show that neither the rigid hyper tableau calculus nor the multi-hyper tableau calculus can employ this redundancy concept, since they do not support the generation of new instances of *separate branch clauses.* Nevertheless, this fact sets the course to improve the multi-hyper tableau calculus to be applicable in practice.

1. Introduction

As a substantial concept, redundancy must be defined and analyzed in connection with the automated theorem proving methods intended to be applied in practice and in concrete softwares. In spite of the fact that theorem proving in first-order logic is undecidable, theorem provers apply methods in order to avoid endless computation. The concept of redundancy arises actually as the theoretical base of these methods, of which effectiveness depends on how redundancy is defined for a particular theorem prover. In this paper, we deal with redundancy for clausal tableaux. *Tableaux* are widely used in theorem proving, and were introduced by Smullyan [9], who employed a unifying notation, by which first-order formulas are classified as of α, β, γ, and δ types. For a γ-formula, tableau rules can be applied several times, which endangers the finiteness of tableau derivations. This latter problem appears in each tableau calculus, which tries to avoid it by introducing an appropriate redundancy criterion in order to restrict the applicability of the

γ-rule. Usually, the γ-rule generates and inserts so-called *rigid variables* [5] into the tableau, like in Free-Variable Tableaux by Fitting [4] – this is why the central problem of handling redundancy is how to deal with rigid variables.

Clausal tableaux [5, 1, 7] make tableau generation quite transparent by extending tableaux with not arbitrary formulas, but solely with clauses. As a class of clausal tableaux, *hyper tableaux* combine clausal tableaux and hyper-resolution [8]. The feasibility of such calculi was already observed in the 1970s [2], but another work by Baumgartner et al. [1] drew the attention to this class, in 1996. Baumgartner's calculus is special in the sense that it avoids using rigid variables, for the sake of simplicity of handling redundancy, among other reasons. This is done by employing so-called purifying substitutions (hence, Baumgartner's hyper tableaux are also called *purified hyper tableaux*), of which generation cannot be automated, neither can purified hyper tableaux. Nevertheless, the redundancy concept for purified hyper tableaux (and, in general, for *purified clausal tableaux*) is worth to introduce in this paper, in Section 2.

Since the aim is to produce entirely automated calculi, purifying substitutions must be eliminated, i.e. hyper tableaux using rigid variables become focal points of research [3, 7, 6]. In Section 3, a redundancy concept is proposed for *rigid clausal tableaux* in general.

1.1. Preliminaries

In the followings, we assume the reader to be familiar with the basic concepts of first-order logic. Formulas and literals are defined as usual, and so are clauses, i.e. a *clause* is a formula $\forall(L_1 \vee \cdots \vee L_k)$ where each L_i is a literal, $k \geqslant 0$. By $\forall A$ we mean the (universal) *closure* of a formula A. A clause $\forall(L_1 \vee \cdots \vee L_k)$ can be denoted by $L_1 \vee \cdots \vee L_k$ as well, i.e. unifiers can be not shown, since all the variables are basically universally quantified. Similarly, the same clause can be considered as a set of its literals, i.e. it can be denoted by $\{L_1, \ldots, L_k\}$.

As usual, tableaux are defined as trees of which the nodes are labeled with formulas. Nevertheless, a *tableau* can be considered as a set of its branches, and a *branch* can be considered as a set of its formulas, too. In the followings, we deal solely with *clausal tableaux* [5, 1, 7], i.e. with tableaux of which the nodes are labeled with literals. Furthermore, we regard all the variables occurring in a clausal tableau as *rigid variables* ([5], page 114).

Definition 1.1 (Formula Represented by a Tableau). Let

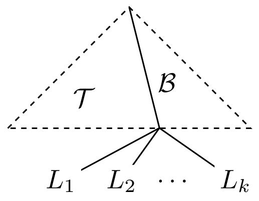
$$\mathcal{T} \quad = \quad \begin{tikzcd}[arrows=-] & & A \arrow[dll] \arrow[dl] \arrow[drr] & & \\ \mathcal{T}_1 & \mathcal{T}_2 & & \cdots & \mathcal{T}_k \end{tikzcd}$$

be a tableau, where A is a formula and $\mathcal{T}_1, \ldots, \mathcal{T}_k$ are tableaux, $k \geqslant 0$. The formula represented by $\mathcal{T}$ is

$$\mathcal{F}(\mathcal{T}) \quad = \quad A \wedge \Big(\bigvee_{1 \leqslant i \leqslant k} \mathcal{F}(\mathcal{T}_i) \Big).$$

According to the above definition, a branch can also be considered as the conjunction of its formulas, and a tableau as the disjunction of its branches (as conjunctions). Sometimes, $\mathcal{T}$ is implicitly considered as $\mathcal{F}(\mathcal{T})$ whenever a formula is needed instead of a tableau. For instance, $\mathcal{FV}(\mathcal{T})$ means the set of all the free variables in $\mathcal{F}(\mathcal{T})$, where $\mathcal{FV}(A)$ denotes the set of all the free variables in a formula or a formula set A.

Definition 1.2 (Attaching a Clause to a Tableau). A clause $C = \{L_1, \ldots, L_k\}$ is attached to a branch $\mathcal{B}$ of a clausal tableau $\mathcal{T}$ by constructing the following tableau:



which is denoted by $\mathcal{T}{+}^{\mathcal{B}}C$.

In the latter definition, all the literals of C are attached to $\mathcal{B}$ one by one, generating k new branches; the other parts of $\mathcal{T}$ remain unchanged.

2. Redundancy for Purified Clausal Tableaux

In order to specify what in general a purified clausal tableau is, let us first introduce the concept of a pure clause [1]:

Definition 2.1 (Pure Clause). A clause C is pure iff $\mathcal{FV}(L_1) \cap \mathcal{FV}(L_2) = \emptyset$ for all $L_1, L_2 \in C$, $L_1 \neq L_2$. A substitution π is a purifying substitution for a clause C iff $C\pi$ is pure.

A *clausal tableau* $\mathcal{T}$ is said *purified* if it has been constructed by attaching solely pure clauses – i.e. the successors of a node do not share any variable. Thus, the closure of the formula represented by $\mathcal{T}$, i.e. $\forall\mathcal{F}(\mathcal{T})$, can be written in the following equivalent form:

$$\bigvee_{\mathcal{B}\in\mathcal{T}} \bigwedge_{L\in\mathcal{B}} \forall L.$$

Thus, each branch of $\mathcal{T}$ can be considered as the conjunction of the closures of its literals. This is why we deal with *redundancy w.r.t. conjunction* in this section.

Theorem 2.2 (Redundancy w.r.t. Conjunction). *Let A and $B_1, \ldots, B_k$ be formulas, $k \geqslant 1$. If there is a substitution σ for some B_i such that $A = B_i\sigma$ then*

$$A \wedge \bigwedge B_i \quad \sim \quad \bigwedge B_i.$$

(For two formulas X and Y, $X \sim Y$ means that X and Y are logically equivalent, i.e. $\mathcal{M} \models X$ iff $\mathcal{M} \models Y$ for all models $\mathcal{M}$.)

Using the above theorem, which is proved in [6], redundancy for purified clausal tableaux can be defined as follows:

Definition 2.3 (Redundancy w.r.t. a Purified Clausal Tableau Branch). A *clause D* is *redundant* w.r.t. a *branch $\mathcal{B}$* of a purified clausal tableau iff there is a substitution σ for two literals $L_1 \in D$ and $L_2 \in \mathcal{B}$ such that $L_1 = \widehat{L}_2\sigma$. The notation $\widehat{L}$ refers a new instance of the literal L (or to be more precise, $\widehat{L}$ is the single literal of a new instance of the clause $\forall L$).

For a purified clausal tableau calculus which has such a derivation rule that provides the generation of new instances of branch literals, the above redundancy concept can be employed. For instance, Baumgartner's (purified) hyper tableau calculus [1] has such a derivation rule, and this is why the above redundancy concept is employed in the so-called Sufficient Redundancy Criterion proposed by Baumgartner [1]. However, as mentioned in Section 1, Baumgartner's calculus employs purifying substitutions, of which generation cannot be automated. This is why we are going investigate on an appropriate redundancy concept for (rigid) clausal tableaux in general, in the next section.

3. Redundancy for (Rigid) Clausal Tableaux

In this section, we are going to formulate a redundancy concept for clausal tableaux in general, including those which are not "purified", i.e. of which the literals may share variables. For example, Kühn's rigid hyper tableau calculus [7] and Kovásznai's multi-hyper (hyperS) tableau calculus [6] employ such tableaux. The formula represented by such a clausal tableau can be written as a conjunction of not by all means pure clauses, and a redundancy concept for such clauses [6] can be specified according to the following theorem:

Theorem 3.1 (Redundancy w.r.t. Disjunction). *Let A and $B_1, \ldots, B_k$ be formulas, $k \geqslant 1$. If*

$$\mathcal{FV}(A) \cap \mathcal{FV}(\{B_1, \ldots, B_k\}) = \emptyset$$

and there is a substitution σ for some B_i such that $A\sigma = B_i$ then

$$A \vee \bigvee B_i \quad \sim \quad \bigvee B_i.$$

As compared to Theorem 2.2, the above theorem, which is proved in [6], is more restrictive since it stipulates that A must not share any free variable with any B_i. This kind of restriction is inherited by the next theorem, which is about redundancy for (rigid) clausal tableaux.

Consider a clausal tableau $\mathcal{T}$ as a tree in Figure 1, where $L_1, \ldots, L_k$ are literals ($k \geqslant 1$), $b_1, \ldots, b_k, b$ are subbranches (which might be empty), and $t_1, \ldots, t_k, t$ are subtableaux (which also might be empty). As can be seen in the figure, the following facts hold:

- $L_1, \ldots, L_k$ are located in distinct branches (i.e. any branch of $\mathcal{T}$ contains at most one L_i).
- Each b_i is the subbranch leading from the root to L_i.

- Each t_i, where $2 \leqslant i \leqslant k$, is the maximal subtableau which has L_i in the root.
- We designate a branch in the maximal L_1-rooted subtableau, and denote it by b. The set of all the other branches of the same subtableau is denoted by t_1.
- t denotes the set of all those branches of $\mathcal{T}$ which do not contain any L_i.

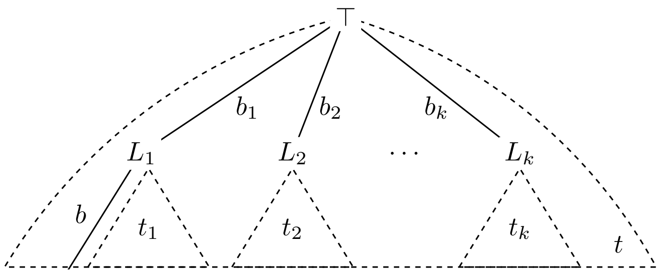


Figure 1: Rigid clausal tableau

$b_1, \ldots, b_k, b$ represent conjunctions, to which we refer also as $b_1, \ldots, b_k, b$, respectively. Similarly, $t_1, \ldots, t_k, t$ represent disjunctions, which are referred to by $t_1, \ldots, t_k, t$, respectively. In the following theorem, in fact, the redundancy of clauses w.r.t. the branch $\mathcal{B} = b_1 \cup \{L_1\} \cup b$ is investigated.

Theorem 3.2. *Let $\mathcal{T}$ be a clausal tableau in the form that can be seen in Figure 1, where*

- *let $\mathcal{B}$ denote the branch $b_1 \cup \{L_1\} \cup b$;*
- *let C denote the clause $L_1 \vee L_2 \vee \cdots \vee L_k$, and let $\mathcal{FV}(C) \cap \mathcal{FV}(t) = \emptyset$.*

The following facts hold:

1. *If $\widehat{C}$ is a new instance of C then*

$$\mathcal{F}(\mathcal{T}) \quad \sim \quad \mathcal{F}\big(\mathcal{T} +^{\mathcal{B}} \widehat{C}\big).$$

2. *If D a clause such that $\widehat{C}\sigma \subseteq D$ for a substitution σ then*

$$\mathcal{F}(\mathcal{T}) \quad \sim \quad \mathcal{F}\big(\mathcal{T} +^{\mathcal{B}} D\big).$$

Proof. $\mathcal{F}(\mathcal{T})$ can be written in the following equivalent form:

$$t \;\vee\; \big(b_1 \wedge L_1 \wedge (b \vee t_1)\big) \;\vee\; \bigvee_{i=2}^{k} (b_i \wedge L_i \wedge t_i)$$

$$\wr$$

$$t \ \vee\ (b_1 \wedge L_1 \wedge b) \ \vee\ (b_1 \wedge L_1 \wedge t_1) \ \vee\ \bigvee_{i=2}^{k} (b_i \wedge L_i \wedge t_i)$$

$$\wr$$

$$t \ \vee\ (b_1 \wedge L_1 \wedge b) \ \vee\ \bigvee_{i=1}^{k} (b_i \wedge L_i \wedge t_i) \tag{3.1}$$

The two statements of the theorem are proved as follows:

1. $\mathcal{F}\big(\mathcal{T} +^{\mathcal{B}} \widehat{C}\big)$ can be written in the following equivalent form:

$$t \ \vee\ \big(b_1 \wedge L_1 \wedge b \wedge \widehat{C}\big) \ \vee\ \bigvee_{i=1}^{k} (b_i \wedge L_i \wedge t_i),$$

which we can use in the following equivalent form, since the *new* clause instance $\widehat{C}$ does not share any variable with any literal in $\mathcal{T}$:

$$t \ \vee\ (b_1 \wedge L_1 \wedge b \wedge \forall\widehat{C}) \ \vee\ \bigvee_{i=1}^{k} (b_i \wedge L_i \wedge t_i). \tag{3.2}$$

Let us prove the following facts for any model $\mathcal{M}$:

(a) <u>if $\mathcal{M} \models (3.2)$ then $\mathcal{M} \models (3.1)$</u>: this fact evidently holds.

(b) <u>if $\mathcal{M} \models (3.1)$ then $\mathcal{M} \models (3.2)$</u>:
Let us rewrite the formula (3.1). First, we rewrite the subformula

$$(b_1 \wedge L_1 \wedge b) \vee \bigvee_{i=1}^{k} (b_i \wedge L_i \wedge t_i)$$

by employing the following rule of distributivity as many times as possible:

$$X \vee (Y \wedge Z) \quad \sim \quad (X \vee Y) \wedge (X \vee Z).$$

So, we get a conjunction of several formulas, including $\bigvee_{i=1}^{k} L_i$, which is actually C itself.
By assumption, $\mathcal{M} \models (3.1)$, i.e. $\mathcal{M} \models \forall(3.1)$. Let us rewrite the formula $\forall(3.1)$:

$$\forall \left[t \vee \left(\Big(\bigvee_{i=1}^{k} b_i \Big) \ \wedge \cdots \wedge\ \Big(\bigvee_{i=1}^{k} L_i \Big) \ \wedge \cdots \wedge\ \Big(b \vee \bigvee_{i=1}^{k} t_i \Big) \right) \right]$$

$$\wr$$

$$\forall\left[\Big(t\vee\bigvee_{i=1}^{k}b_i\Big)\ \wedge\cdots\wedge\ \Big(t\vee\bigvee_{i=1}^{k}L_i\Big)\ \wedge\cdots\wedge\ \Big(t\vee b\vee\bigvee_{i=1}^{k}t_i\Big)\right]$$
$$\wr$$
$$\forall\left[t\vee\bigvee_{i=1}^{k}b_i\right]\ \wedge\cdots\wedge\ \forall\left[t\vee\bigvee_{i=1}^{k}L_i\right]\ \wedge\cdots\wedge\ \forall\left[t\vee b\vee\bigvee_{i=1}^{k}t_i\right]$$
$$\wr$$
$$\cdots\wedge\ \forall\big(t\vee C\big)\ \wedge\ldots$$
$$\wr\quad\longleftarrow\text{ since }\mathcal{FV}(C)\cap\mathcal{FV}(t)=\emptyset$$
$$\cdots\wedge\ (\forall t\vee\forall C)\ \wedge\ldots$$

Hence, for any model $\mathcal{M}$ such that $\mathcal{M}\models(3.1)$, it holds that

$$\mathcal{M}\models\forall t\vee\forall C.$$

Thus, there are two cases:

i. $\underline{\mathcal{M}\models\forall t}$: in this case, $\mathcal{M}\models(3.2)$ evidently holds.

ii. $\underline{\mathcal{M}\models\forall C}$:
It is to prove that if $\mathcal{M}\models(3.1)\varrho$ then $\mathcal{M}\models(3.2)\varrho$, for any valuation ϱ over $\mathcal{M}$. It is sufficient to prove this fact for only such valuations ϱ that

$$\mathcal{M}\quad\models\quad(b_1\wedge L_1\wedge b)\varrho$$

(for other valuations, the proof is trivial). Because of this fact – and since $\mathcal{M}\models\forall C$, and hence $\mathcal{M}\models\forall\widehat{C}$ –, the following fact also holds:

$$\mathcal{M}\quad\models\quad(b_1\wedge L_1\wedge b)\varrho\ \wedge\ \forall\widehat{C}.$$

As a trivial consequence, $\mathcal{M}\models(3.2)\varrho$ holds.

2. The formula $\mathcal{F}\big(\mathcal{T}+^{\mathcal{B}}D\big)$ can be written in the following equivalent form:

$$t\ \vee\ (b_1\wedge L_1\wedge b\wedge D)\ \vee\ \bigvee_{i=1}^{k}(b_i\wedge L_i\wedge t_i). \tag{3.3}$$

It is to prove for any model $\mathcal{M}$ that

(a) $\underline{\text{if }\mathcal{M}\models(3.3)\text{ then }\mathcal{M}\models(3.1)}$: this fact evidently holds.

(b) $\underline{\text{if }\mathcal{M}\models(3.1)\text{ then }\mathcal{M}\models(3.3)}$:
According to 1., it holds that $(3.1)\sim(3.2)$. We show that if $\mathcal{M}\models(3.2)\varrho$ then $\mathcal{M}\models(3.3)\varrho$, for any model $\mathcal{M}$ and any valuation ϱ over $\mathcal{M}$.

It is sufficient to prove this fact for only such models $\mathcal{M}$ and such valuations ϱ that

$$\mathcal{M} \quad \models \quad (b_1 \wedge L_1 \wedge b \wedge \forall\widehat{C})\varrho$$

(for other models and other valuations, the proof is trivial). This fact can be reformulated as follows:

$$\mathcal{M} \quad \models \quad (b_1 \wedge L_1 \wedge b)\varrho \wedge \forall\widehat{C}.$$

Hence,

$$\mathcal{M} \quad \models \quad \forall\widehat{C},$$

and hence,

$$\mathcal{M} \quad \models \quad \forall(\widehat{C}\sigma).$$

Since $\widehat{C}\sigma \subseteq D$, it holds that

$$\mathcal{M} \quad \models \quad \forall D.$$

Thus,

$$\mathcal{M} \quad \models \quad (b_1 \wedge L_1 \wedge b)\varrho \wedge \forall D$$

also holds, which implies that

$$\mathcal{M} \quad \models \quad (b_1 \wedge L_1 \wedge b \wedge D)\varrho.$$

As a trivial consequence, $\mathcal{M} \models (3.3)\varrho$ holds.

□

In order to characterize the clause C in the above theorem, let us introduce the following concept:

Definition 3.3 (Separate Branch Clause). Let $\mathcal{T}$ be a clausal tableau. Let $L_1, \ldots, L_k$ ($k \geqslant 1$) be literals in $\mathcal{T}$ such that

1. any branch of $\mathcal{T}$ contains at most one L_i;
2. for each branch $\mathcal{B}$ of $\mathcal{T}$ such that $\mathcal{B}$ does not contain any L_i, it holds that

$$\mathcal{FV}\big(\{L_1, \ldots, L_k\}\big) \cap \mathcal{FV}(\mathcal{B}) = \emptyset.$$

$\{L_1, \ldots, L_k\}$ is a *separate branch clause* of any branch containing some L_i.

According to the above theorem, the following redundancy concept can be formulated for any clausal tableau:

Definition 3.4 (Redundancy w.r.t. a (Rigid) Clausal Tableau Branch). A clause D is redundant w.r.t. a branch $\mathcal{B}$ of a clausal tableau $\mathcal{T}$ iff there is an $L \in \mathcal{B}$ and there is a separate branch clause C of $\mathcal{B}$ such that C contains L and the following statement holds: some subclause of D is an instance of a new instance of C.

For a clausal tableau calculus which has such a *derivation rule that provides the generation of new instances of separate branch clauses*, the above redundancy concept can be employed.

For instance, Kühn's rigid hyper tableau calculus [7] and Kovásznai's multi-hyper tableau calculus [6] employ rigid variables in order to eliminate purifying substitutions from hyper tableaux. Kühn's calculus provides the generation of new instances of so-called *branch clauses* [7], which are not by all means "separate", i.e. they may share variables with the rest of the tableau. The main problem with Kühn's calculus is that the generation of branch clauses in general destroys soundness, and hence this calculus is not sound. Futhermore, neither the redundancy concept for purified clausal tableaux nor the one for (rigid) clausal tableaux can be employed.

Kovásznai's multi-hyper tableau calculus eliminates purifying substitutions from hyper tableaux successfully, i.e. it is sound and complete in first order logic. But it does not support the generation of new instances of separate branch clauses, either. This is why it cannot employ the above redundancy concept. It would be expedient to improve the multi-hyper tableau calculus to be able to generate new instances of separate branch clauses, without losing soundness and completeness.

4. Summary

In this paper, we proposed a redundancy concept for (rigid) clausal tableaux, and proved that it was sound. We showed how rigid variables complicated the redundancy check, and introduced a simple redundancy criterion for purified clausal tableaux. However, the generation of purifying substitution cannot be automated, this is why we investigated on a redundancy concept for clausal tableaux in general. This concept is not completely general in the sense that only such a clausal tableau calculus can employ it which has a derivation rule that provides the generation of new instances of separate branch clauses. We pointed out that it would have been expedient to incorporate such a rule in the multi-hyper tableau calculus in order to get a sound and complete clausal tableau calculus which had an appropriate redundancy criterion and hence could be used in practice.

References

[1] Baumgartner, P., Furbach, U., Niemelä, I., Hyper Tableaux, *Lecture Notes in Computer Science,* Vol. 1126, (1996), 1–17.

[2] Brown, F. M., Towards the Automation of Set Theory and its Logic, *Artificial Intelligence,* Vol. 10, (1978), 281–316.

[3] J. van Eijck, Constrained Hyper Tableaux, *Lecture Notes in Computer Science,* Vol. 2142, (2001), 232–246.

[4] Fitting, M., First-Order Logic and Automated Theorem Proving, *Springer-Verlag*, (1996).

[5] Hähnle, R., Tableaux and Related Methods, Handbook of Automated Reasoning, by J. A. Robinson and A. Voronkov, Vol. 1, Chapter 3, *Elsevier and MIT Press*, (2001), 100–178.

[6] Kovásznai, G., HyperS Tableaux – Heuristic Hyper Tableaux, *Acta Cybernetica,* Vol. 17, (2005), 325–338, (ZBL#1099.68096, MR#2183822).

[7] Kühn, M., Rigid Hypertableaux, *Lecture Notes in Artificial Intelligence,* Vol. 1303, (1997), 87–98.

[8] Robinson, J. A., Automated Deduction with Hyper-Resolution, *International Journal of Computer Mathematics,* Vol. 1, (1965), 227–234.

[9] Smullyan, R. M., First-Order Logic, *Springer-Verlag,* (1968).