Annales Mathematicae et Informaticae
***36** (2009) pp. 85–101*
http://ami.ektf.hu

Introducing general redundancy criteria for clausal tableaux, and proposing resolution tableaux

Gergely Kovásznai[a], Gábor Kusper[b]

[a]Department of Information Technology
Eszterházy Károly College, Eger, Hungary

[b]Department of Computing Science
Eszterházy Károly College, Eger, Hungary

Submitted 8 October 2008; Accepted 6 December 2009

Abstract

Hyper tableau calculi are well-known as attempts to combine hyper-resolution and tableaux. Besides their soundness and completeness, it is also important to give an appropriate redundancy criterion. The task of such a criterion is to filter out "unnecessary" clauses being attached to a given tableau. This is why we investigate what redundancy criteria can be defined for clausal tableaux, in general.

This investigation leaded us to a general idea for combining resolution calculi and tableaux. The goal is the same as in the case of hyper-tableau calculi: to split (hyper-)resolution derivations into branches. We propose a novel method called resolution tableaux. Resolution tableaux are more general than hyper tableaux, since any resolution calculus (not only hyper-resolution) can be applied, like, e.g., binary resolution, input resolution, or lock resolution etc. We prove that any resolution tableau calculus inherits the soundness and the completeness of the resolution calculus which is being applied. Hence, resolution tableaux can be regarded as a kind of parallelization of resolution.

1. Introduction

Hyper tableau calculi (e.g., hyper tableaux [2, 3], constrained hyper tableaux [6], rigid hyper tableaux [11], and hyperS tableaux [8, 9] etc.) are well-known as attempts to combine hyper-resolution and tableaux. Besides their soundness and

completeness[1], it is also important to give an appropriate *redundancy criterion*. The task of such a criterion is to filter out "unnecessary" clauses being attached to a given tableau. This is why we investigate what redundancy criteria can be defined for *clausal tableaux*, in general. A clausal tableau is actually a tableau whose vertices are labeled with literals. Besides hyper tableau calculi, there are other well-known tableau calculi which apply clausal tableaux as well, like, e.g., clause tableaux [7] and connection tableaux [12]. We give a detailed investigation on appropriate redundancy criteria for clausal tableaux, and propose two possible candidates. As it will be seen, the second criterion is more general than the first one, and is also more general than the redundancy criterion for (purified) hyper tableaux [2]. We illustrate how to employ our redundancy criteria, by examples.

Investigation on redundancy leaded us to a general idea for combining resolution calculi and tableaux. The goal is the same as in the case of hyper-tableau calculi: to split (hyper-)resolution derivations into branches. First, we propose a general way of representing any resolution calculus (and illustrate it by examples), and then we introduce a novel method called *resolution tableaux*. Resolution tableaux are more general than hyper tableaux, since any resolution calculus (not only hyper-resolution) can be applied, like, e.g., binary resolution, input resolution, or lock resolution etc. We prove that any resolution tableau calculus inherits *the soundness and the completeness* of the resolution calculus which is being applied. By the use of resolution tableaux, any resolution derivation can be split into separate branches, hence resolution tableaux can be regarded as a kind of *parallelization* of resolution, as it will be illustrated by an example.

The structure of the paper is as follows. In Section 2, basic definitions and concepts are introduced. After this, let us depart from the logical order written above. First, let us propose resolution tableaux in Section 3, and then introduce our results on redundancy criteria for clausal tableaux, in Section 4. As it will be seen, the latter topic is in close connection with resolution tableaux, as detailed in Section 5.

2. Preliminaries

In the followings, we assume that the reader is familiar with the basic concepts of first-order logic. Nevertheless, let us present a few crucial concepts.

A *literal* is a formula either A or $\neg A$ where A is an atomic formula. A is classified as a positive, $\neg A$ as a negative literal.

A *clause* is a formula $L_1 \vee L_2 \vee \ldots \vee L_n$ where $n \geqslant 0$ and each L_i is a literal $(i = 1, \ldots, n)$. A clause can also be regarded as the set of its literals. The *empty clause* is denoted by $\bot$.

A clause is *positive* (negative) iff it consists of solely positive (negative) literals.

Two clauses are *independent* iff there is no variable that occurs in both of them.

[1] *Constrained hyper tableaux* and *hyperS tableaux* are sound and complete in first-order logic. *Hyper tableaux* [2] without purifying substitutions are complete only in Horn logic (c.f. [3] for improvement). *Rigid hyper tableaux* have not been proven to be complete yet.

$C\sigma$ is called an *instance* of a clause C where σ is a variable substitution. $C\sigma$ is a *new instance* if σ is a variable renaming and its range consists solely of new variables.

A clause C *subsumes* a clause D iff C has an instance $C\sigma$ such that $C\sigma \subseteq D$.

Given a formula A, let $\forall A$ denote the *universal closure* of A.

As usual, $M \models A$ denotes the fact that a formula A is *satisfied by a model* M. In the case of A being open, $M \models A$ iff $M \models \forall A$.

Two formulas A and B are *equivalent* (denoted by $A \sim B$) iff for any model M: $M \models A$ iff $M \models B$.

As it is well-known, the *most general unifier* (MGU) of two atomic formulas A and B is the most general variable substitution σ such that $A\sigma = B\sigma$. Let us generalize the definition of MGUs, as follows. The MGU of (A_1, B_1), $(A_2, B_2), \ldots,$ (A_n, B_n), where all A_i and B_i are atomic formulas, is the most general variable substitution σ such that $A_i\sigma = B_i\sigma$ for all $i = 1, \ldots, n$.

Tableaux are regarded as trees whose which vertices are labeled with formulas [15, 7]. Sometimes, for the sake of briefness, we regard a tableau as the set of all its branches. Similarly, a branch is often regarded as the sequence or the set of all the vertices in the branch. Furthermore, let us introduce the following notation:

Notation 2.1. Let $\mathcal{N}$ be a vertex set from a tableau.
(1) Let $\widehat{\mathcal{N}}$ denote the conjunction of all the labels (formulas) in $\mathcal{N}$.
(2) Let $\check{\mathcal{N}}$ denote the disjunction of all the labels (formulas) in $\mathcal{N}$.

Sometimes it is needed to regard a tableau as a sole formula. This is why we need the following definition:

Definition 2.2 (Formula Represented by a Tableau). The *formula* $\mathcal{F}(T)$ *represented* by a tableau T is defined inductively as follows:
(1) If T consists of one single vertex labeled with a formula L, then

$$\mathcal{F}(T) = L$$

(2) If T is a compound tableau, i.e., it is in the form as can be seen in Figure 1, where L is a formula and each T_i is a tableau, then

$$\mathcal{F}(T) = L \wedge \left(\bigvee_{i=1}^{n} \mathcal{F}(T_i) \right)$$

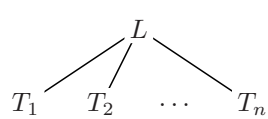


Figure 1: Compound tableau.

Let us note the following obvious fact, which says that any tableau can be regarded as the disjunction of its branches (as conjunctions).

Lemma 2.3. *For any tableau T,*

$$\mathcal{F}(T) = \bigvee_{B \in T} \widehat{B}$$

3. Resolution tableaux

The aim is to introduce a general method for combining resolution calculi and tableaux. This is why a general way of representing resolution calculi is required. We regard a resolution calculus as a *set of inference rules*, which act on clauses. Each resolution inference rule is represented as a function which can assign a clause to one or more clauses. Every time when applying such a rule, it is needed to specify a clause set (denoted by $\mathcal{I}$ and called the *input clause set*) and a sequence of clauses (denoted by $\mathbf{d}$ and called the *resolution derivation*).

Definition 3.1 (Resolution Inference Rule). A *resolution inference rule* is a function $res_{\mathcal{I},\mathbf{d}} : \mathcal{D}om \mapsto \mathcal{C}$, where

- $\mathcal{I}$ is a finite set of clauses;
- $\mathbf{d}$ is a finite sequence of clauses;
- $\mathcal{C}$ is the set of all the clauses;
- $\mathcal{D}om \subseteq P(\mathcal{C})$.

Let us illustrate by examples how well-known resolution calculi can be represented in this form. Of course, other resolution calculi could be represented in a similar way[2].

Example 3.2 (Binary Resolution). The resolution calculus, as was introduced by Robinson [13], can be represented by the set of the following resolution inference rules [1]:

(1) *Binary Resolution*:

$$binres_{\mathcal{I},\mathbf{d}}\big(\ A \vee C\ ,\ \neg B \vee D\ \big) = (C \vee D)\sigma$$

where σ is the most general unifier (MGU) of the atomic formulas A and B.

(2) *(Positive) Factoring*:

$$factor_{\mathcal{I},\mathbf{d}}\big(\ C \vee A \vee B\ \big) = (C \vee A)\sigma$$

where σ is the MGU of the atomic formulas A and B.

[2]It is to be remarked that $\mathcal{I}$ may be defined as a clause sequence (instead of a clause set) in the case of some resolution calculi, where the order of input clauses should not be neglected, like in SLD-resolution [10] and in lock resolution [4, 1, 5].

Example 3.3 (Linear Input Resolution). The linear input resolution calculus [5] can be represented by the same resolution inference rules as binary resolution, but the rule "Binary Resolution" is restricted as follows:

- one of the clauses $C \vee A$ and $D \vee \neg B$ must be the last element of $\mathbf{d}$;
- the other one must be an element of $\mathcal{I}$.

Example 3.4 (Hyper-Resolution). The (positive) hyper-resolution calculus, as was introduced by Robinson [14], can be represented by the following resolution inference rule:

$$\begin{array}{c} hypres_{\mathcal{I},\mathbf{d}}(\ A_1 \vee C_1\ , \ldots,\ A_n \vee C_n\ ,\ \neg B_1 \vee \ldots \vee \neg B_n \vee D\ ) \\ \| \\ (C_1 \vee \ldots \vee C_n \vee D)\sigma \end{array}$$

where

- $n \geqslant 1$;
- C_i is a positive or empty clause ($i = 1, \ldots, n$);
- D is a positive or empty clause;
- A_i and B_i are atomic formulas ($i = 1, \ldots, n$);
- σ is the MGU of $(A_1, B_1), \ldots, (A_n, B_n)$.

When a resolution calculus (as a set of resolution inference rules) is given, an appropriate tableau can be constructed for a given input clause set $\mathcal{I}$. Such a tableau is called a *resolution tableau*, and can be constructed in a quite simple way. In every deduction steps, some clauses are to be selected, each either from a given branch of the tableau or from the input clause set $\mathcal{I}$. To the selected clauses a resolution inference rule is applied, resulting in a clause D. First D must be split into *independent subclauses*, and then these subclauses are attached to the given branch, forming distinct new branches.

Let us define resolution tableaux inductively, as follows:

Definition 3.5 (Resolution Tableaux). Let $\mathcal{R}$ be a set of resolution inference rules. Let $\mathcal{I}$ be a clause set.

(1) One single vertex labeled with $\top$ is a *resolution tableau* for $\mathcal{I}$ w.r.t. $\mathcal{R}$.

(2)
- Let T be a resolution tableau for $\mathcal{I}$ w.r.t. $\mathcal{R}$.
- Let B be a branch of T.
- Let $C_1, \ldots, C_n$ be *new instances* of clauses in $\mathcal{I} \cup B$.
- Let $res \in \mathcal{R}$ such that $res_{\mathcal{I},B}$ is defined on $C_1, \ldots, C_n$, and let

$$D = res_{\mathcal{I},B}(C_1, \ldots, C_n)$$

– Let $D = D_1 \vee \ldots \vee D_k$ such that each distinct D_i and D_j are *independent* clauses $(i, j = 1, \ldots, k)$.[3]

The tableau that can be seen in Figure 2 is a *resolution tableau* for $\mathcal{I}$ w.r.t. $\mathcal{R}$.

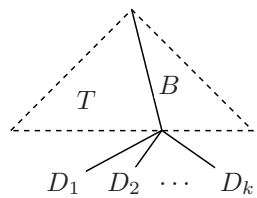


Figure 2: Attaching a clause to a branch B of a resolution tableau T.[4]

Let us point out that it is mandatory to generate *new instances* of the clauses which have been selected. Note that resolution tableau branches can be regarded (and are used) as separate resolution derivations.

A tableau calculus is regarded sound and complete in the following case: any clause set $\mathcal{I}$ is unsatisfiable iff a closed tableau exists for $\mathcal{I}$. A *closed resolution tableau* is defined as follows:

Definition 3.6 (Closed Resolution Tableaux). A resolution tableau is *closed* iff each of its branches contains $\bot$.

Assume a resolution calculus $\mathcal{R}$ which is sound and complete in first-order logic (or in a fragment of first-order logic). It is quite obvious that the resolution tableau calculus applying $\mathcal{R}$ inherits soundness and completeness. For example, since hyper-resolution is sound and complete in first-order logic, so is the hyper-resolution tableau calculus[5]. The linear input resolution tableau calculus[6] is sound and complete in Horn logic.

Theorem 3.7. *If a resolution calculus $\mathcal{R}$ is sound and complete (in a fragment of first-order logic), then so is the resolution tableau calculus applying $\mathcal{R}$.*

Proof. *Soundness:* It is to show that if there is a closed resolution tableau for $\mathcal{I}$ w.r.t. $\mathcal{R}$, then $\mathcal{I}$ is unsatisfiable.

If $\mathcal{R}$ is sound, then each inference rule $res \in \mathcal{R}$ preserves satisfiability. Let

$$res(C_1, \ldots, C_n) = D_1 \vee \ldots \vee D_k$$

where each distinct D_i and D_j are independent. It can be seen that for any model M:

[3] Furthermore, one can additionally demand that no D_i can further be split into independent subclauses $(i = 1, \ldots, k)$. In this case, decomposition of clauses is unique, and can easily be solved algorithmically.

[4] New vertices labeled with $D_1, \ldots, D_k$ are attached to the leaf of B.

[5] I.e., the resolution tableau calculus applying hyper-resolution.

[6] I.e., the resolution tableau calculus applying linear input resolution.

$$\text{if } M \models C_1, \ldots, C_n, \text{ then } M \models D_1 \vee \ldots \vee D_k.$$

Because of independence:

$$\forall\,(D_1 \vee \ldots \vee D_k) \sim \forall D_1 \vee \ldots \vee \forall D_k$$

Summing up, for any model M:

$$\text{if } M \models C_1, \ldots, C_n, \text{ then } M \models D_1 \text{ or } M \models D_2 \text{ or } \ldots \text{ or } M \models D_k.$$

Hence, if $\mathcal{I}$ was satisfiable, then at least one branch could not be closed.

Completeness: It is to show that if $\mathcal{I}$ is unsatisfiable, then there is a closed resolution tableau for $\mathcal{I}$ w.r.t. $\mathcal{R}$. This fact is even more obvious than in the case of soundness. Since $\mathcal{R}$ is complete, there is a resolution refutation from $\mathcal{I}$. Each tableau branch can actually be regarded as a "simplified" variant of that refutation, i.e., only subclauses occuring in the refutation can occur in the branch. Since $\bot$ is deduced in the refutation and all literals of the clauses can be resolved out, obviously $\bot$ can occur in each branch. □

Note that the fact that $D_1, \ldots, D_k$ are pairwise independent has been employed only in the soundness proof.

Example 3.8 (Linear Input Resolution Tableaux). Consider the following input clause set:

$$\mathcal{I} = \left\{ \begin{array}{c} M(a, s(c), s(b)) \\ P(a) \\ M(x, x, s(x)) \vee D(y, x) \\ \neg M(x, y, z) \vee D(x, z) \\ \neg P(x) \vee \neg M(y, z, u) \vee \neg D(x, u) \vee D(x, y) \vee D(x, v) \\ \neg D(a, b) \end{array} \right\}$$

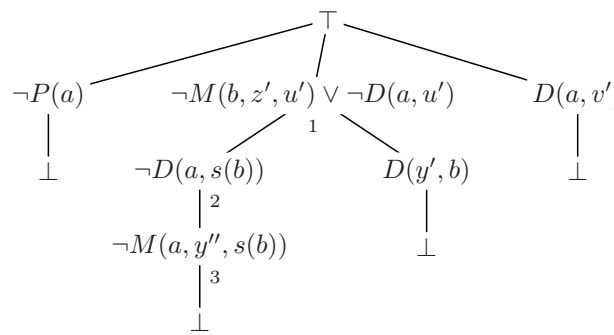
a, b, c are constants, u, v, x, y, z are variables.

In Figure 3, a closed resolution tableau for $\mathcal{I}$ w.r.t. linear input resolution (c.f. Example 3.3) can be seen.

First, the input clauses $\neg P(x) \vee \neg M(y, z, u) \vee \neg D(x, u) \vee D(x, y) \vee D(x, v)$ and $\neg D(a, b)$ are selected; the atomic formulas $D(x', y')$ and $D(a, b)$ are resolved upon by MGU $\{x'/a, y'/b\}$. As can be seen, the resolvent is split into four independent subclauses. Three branches can obviously get closed by resolving with the unit input clauses $P(a)$ and $\neg D(a, b)$.

Let us focus on the branch which contains $\neg M(b, z', u') \vee \neg D(a, u')$. Since the basis is linear input resolution, this clause (as the label of the last vertex in the branch) must be resolved with an input clause. Currently, that input clause is $M(x, x, s(x)) \vee D(y, x)$. The resolvent is split into two subclauses.

The consequent steps can be similarly performed.



Selected input clauses:

[1]: $M(x, x, s(x)) \vee D(y, x)$
[2]: $\neg M(x, y, z) \vee D(x, z)$
[3]: $M(a, s(c), s(b))$

Figure 3: Closed linear input resolution tableau.

4. Redundancy criteria for clausal tableaux

Hyper tableau calculi (e.g., hyper tableaux [2, 3], rigid hyper tableaux [11], constrained hyper tableaux [6], and hyperS tableaux [8, 9]) are well-known in theorem proving. They combine hyper-resolution and tableaux. Why *hyper-resolution*? Because hyper-resolution is long known to be a key ingredient to success in theorem proving (as written in [7]). Why *tableaux*? Because of the same purpose as in resolution tableau calculi: to split hyper-resolution derivations into branches.

Hyper tableau calculi apply *clausal tableaux* as data structures representing the branches of derivations. A clausal tableau is actually a tableau whose vertices are labeled with literals. Besides hyper tableau calculi, there are other well-known tableau calculi which apply clausal tableaux as well, like, e.g., clause tableaux [7] and connection tableaux [12].

As usual, a clausal tableau is constructed by repeatedly "attaching" clauses to its branches. Clausal tableaux can be defined inductively, as follows:

Definition 4.1 (Clausal Tableaux).

(1) One single vertex labeled with $\top$ is a clausal tableau.

(2)
- Let T be a clausal tableau, and B a branch of T.
- Let $E = L_1 \vee L_2 \vee \ldots \vee L_k$ be a clause.

The tableau that can be seen in Figure 4 is denoted by $T +^B E$, and is also a clausal tableau.

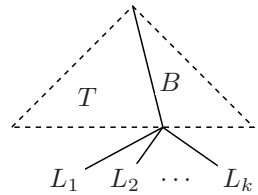


Figure 4: Attaching a clause to a branch B of a clausal tableau T.[7]

When a clause E is being attached to a branch B, the following question is essential to be answered: is it "unnecessary" to attach E to B? In other words: is E *redundant* w.r.t. B? Some clausal tableau calculi define so-called *redundancy criteria* in order to give a precise answer, by regarding only E and B (and maybe T).

Example 4.2 (Redundancy Criterion for Hyper Tableaux). Hyper tableaux have the following redundancy criterion [2]: a clause E is redundant w.r.t. a branch B iff

$\exists L_1 \in E$ and $\exists L_2 \in B$ such that L_1 is an instance of a new instance of L_2.

We are trying to investigate what redundancy criteria can be applied in connection with clausal tableaux, in general. Besides we are extending our preceding results [8, 9], we are going to show that this problem is in close connection with resolution tableaux.

4.1. Preliminaries

First, it is essential to give a precise definition of redundancy for clausal tableaux. Note that the definition regards tableaux as formulas, as it has been introduced in Definition 2.2.

Definition 4.3 (Redundant Clause).

- Let T be a clausal tableau, and B a branch of T.
- Let E be a clause.

E is redundant w.r.t. B in T iff

$$\mathcal{F}(T) \sim \mathcal{F}\left(T +^{B} E\right)$$

Let us prove two lemmas which will be essential for proving the soundness of the redundancy criteria in the following sections.

Lemma 4.4 (Reducing Lemma).

[7] The only difference between Figure 4 and Figure 2 is that new vertices are now labeled only with literals.

- *Let T be a clausal tableau, and B a branch of T.*
- *Let M be a model.*
- *Let E be a clause.*

It holds that

$$\text{if } M \models \mathcal{F}\left(T+^{B} E\right), \text{ then } M \models \mathcal{F}(T).$$

Proof. It is to prove that $M \models \mathcal{F}\left(T+^{B} E\right)\theta$ implies $M \models \mathcal{F}(T)\,\theta$, for any valuation θ. Assume, by Lemma 2.3, we have $M \models (\widehat{B} \wedge E)\theta$ (the opposite case is obvious to prove), i.e., $M \models \widehat{B}\theta \wedge E\theta$. Thus, $M \models \widehat{B}\theta$, which implies that $M \models \mathcal{F}(T)\,\theta$, by Lemma 2.3. □

Lemma 4.5 (Extending Lemma).

- *Let T be a clausal tableau, and B a branch of T.*
- *Let M be a model.*
- *Let E be a clause such that $M \models E$.*

It holds that

$$\text{if } M \models \mathcal{F}(T), \text{ then } M \models \mathcal{F}\left(T+^{B} E\right).$$

Proof. It is to prove that $M \models \mathcal{F}(T)\,\theta$ implies $M \models \mathcal{F}\left(T+^{B} E\right)\theta$, for any valuation θ. Assume, by Lemma 2.3, we have $M \models \widehat{B}\theta$ (the opposite case is obvious to prove). Since $M \models E$, it holds that $M \models \widehat{B}\theta \wedge E$. Hence, $M \models (\widehat{B} \wedge E)\theta$, which implies that $M \models \mathcal{F}\left(T+^{B} E\right)\theta$, by Lemma 2.3. □

Note that E is an arbitrary clause in the Reducing Lemma in contrast with the Extending Lemma, where E must fulfill the following stipulation: $M \models E$.

4.2. Redundancy Criterion I

When checking if a clause E is redundant w.r.t. a tableau branch, an appropriate clause C is needed to be "extracted" from the tableau in order to "compare" C with E via instantiation. A so-called *branch clause* is a suitable candidate for this role.

Definition 4.6 (Branch Clause).

- Let T be a clausal tableau.
- Let $\mathcal{N}$ be a vertex set from T such that each branch of T contains *exactly* one element of $\mathcal{N}$.

The clause $\breve{\mathcal{N}}$ is a *branch clause* in T.

The Extending Lemma is to be applied in the redundancy criterion proposed later in this section. Therefore, each model of a clausal tableau T should be proven to satisfy any branch clause in T.

Lemma 4.7 (Branch Clause Lemma).

- *Let T be a clausal tableau.*
- *Let M be a model.*
- *Let C be a branch clause in T.*

It holds that

$$\textit{if } M \models \mathcal{F}(T), \textit{ then } M \models C.$$

Proof. The statement can be proven by induction, as follows:
(1) If T consists of one single vertex, then the statement obviously holds.
(2) If T is a compound tableau (cf. Figure 1), then

$$\mathcal{F}(T) \sim L \wedge \left(\bigvee_{i=1}^{n} \mathcal{F}(T_i) \right) \tag{4.1}$$

There are two cases:
(a) If $C = L$, then the statement obviously holds.
(b) Otherwise, $C = C_1 \vee \ldots \vee C_n$ where each C_i is a branch clause in T_i. By the inductive hypothesis, if $M \models \mathcal{F}(T_i)$, then $M \models C_i$, for all $i = 1, \ldots, n$. Thus,

$$\text{if } M \models \bigvee_{i=1}^{n} \mathcal{F}(T_i), \text{ then } M \models C.$$

By (4.1), the statement holds. □

The significance of the previous lemma is that a branch clause can be instantiated "without restriction", i.e., any instance (even a new instance) of a branch clause is satisfied by any model of the given tableau. Based on this fact, the following theorem on redundancy can be proven.

Theorem 4.8 (Redundancy Theorem I).

- *Let T be a clausal tableau, and B a branch of T.*
- *Let C be a branch clause in T.*
- *Let C' be a new instance of C, and E a clause such that C' subsumes E.*

It holds that

$$\mathcal{F}(T) \sim \mathcal{F}\left(T +^{B} E\right)$$

Proof. The equivalence in the right-to-left direction is a direct consequence of the Reducing Lemma. Let us prove the equivalence in the left-to-right direction, i.e., prove that for any model M:

$$\text{if } M \models \mathcal{F}(T), \text{ then } M \models \mathcal{F}\left(T +^{B} E\right).$$

Assume that $M \models \mathcal{F}(T)$. By Lemma 4.7, $M \models C$. Thus, $M \models C'$ also holds. So does $M \models E$. By the Extending Lemma, the proof is complete. □

Using this theorem, the following redundancy criterion can be proposed:

Definition 4.9 (Redundancy Criterion I). A clause E is redundant w.r.t. a clausal tableau T iff there is a branch clause C in T such that E is subsumed by a new instance of C.

Example 4.10 (Redundancy Criterion I). Let T be the clausal tableaux in Figure 5.

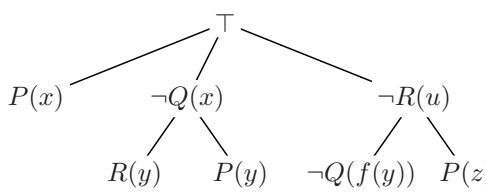


Figure 5: A clausal tableau.

When checking if a clause E is redundant w.r.t. T, Redundancy Criterion I can be applied only if E consists of at least three literals, since any branch clause instance in T consists of at least three literals (except for $\top$). For example, if

$$E = P(a) \vee R(a) \vee \neg Q(f(a))$$

then E is redundant w.r.t. T since the new branch clause instance

$$P(x') \vee R(y') \vee P(y') \vee \neg Q(f(y')) \vee P(z')$$

subsumes E, via the variable substitution $\sigma = \{x'/a, y'/a, z'/a\}$.

Note that none of all the other branch clauses subsumes E, neither $P(x) \vee \neg Q(x) \vee \neg R(u)$ nor $P(x) \vee R(y) \vee P(y) \vee \neg R(u)$ nor $P(x) \vee \neg Q(x) \vee \neg Q(f(y)) \vee P(z)$.

Note that Redundancy Criterion I is not very practicable. The problem is that a branch clause usually contains a lot of literals, and a clausal tableau may have lots of branch clauses. A more sophisticated redundancy criterion is needed.

4.3. Redundancy Criterion II

A restricted variant of a branch clause is needed, which can also be applied by the Extending Lemma. The wanted clause is a subclause of a branch clause (hence, it does not probably consist of too many literals) and does not have a common variable with a considerable part of the tableau (hence, there exist probably not too many such clauses). Let us define this clause as follows:

Definition 4.11 (Separate Branch Clause).

- Let T be a clausal tableau, and B a branch of T.
- Let $\mathcal{N}$ be a vertex set from T such that
  - each branch of T contains *at most* one element of $\mathcal{N}$;
  - B contains one element of $\mathcal{N}$;
  - there is no variable occurring both in $\mathcal{N}$ and in a branch which does not contain any element of $\mathcal{N}$.

The clause $\breve{\mathcal{N}}$ is a *separate branch clause* of B in T.

Such a clause is "separate" from the branches which do not contain it, by demanding it not to share any variable with them. As proven in the following theorem, a separate branch clause can also be used in redundancy criteria, similarly as a branch clause was.

Theorem 4.12 (Redundancy Theorem II).

- *Let T be a clausal tableau, and B a branch of T.*
- *Let C be a separate branch clause of B in T.*
- *Let C' be a new instance of C, and E a clause such that C' subsumes E.*

It holds that

$$\mathcal{F}(T) \sim \mathcal{F}\left(T +^{B} E\right)$$

Proof. Similarly as in the proof of Theorem 4.8, it is the most important to prove that for any model M:

$$\text{if } M \models \mathcal{F}(T), \text{ then } M \models \mathcal{F}\left(T +^{B} E\right)$$

The branches of T can be divided into two groups – those which contain any of the nodes included by C, and those which do not. Let t denote the set of the branches in the first group, and $\overline{t}$ the set of the ones in the second group[8]. First of all, notice the following obvious facts:

$$\mathcal{F}(T) \sim \mathcal{F}\left(\overline{t}\right) \vee \mathcal{F}(t) \tag{4.2}$$

[8] t and $\overline{t}$ are subtableaux of T. It is important that the branches of t and $\overline{t}$ are also branches in T.

$$F(T+^B E) \sim \mathcal{F}\left(\overline{t}\right) \vee \mathcal{F}\left(t+^B E\right) \tag{4.3}$$

Assume that $M \models \mathcal{F}(T)$. Since C is a branch clause in t, (4.2) and Lemma 4.7 together imply that

$$M \models \mathcal{F}\left(\overline{t}\right) \vee C$$

I.e., $M \models \forall\left(\mathcal{F}\left(\overline{t}\right) \vee C\right)$. By the assumption that $\overline{t}$ and C do not share any variable (since C is a separate branch clause), it holds that

$$M \models \forall\mathcal{F}\left(\overline{t}\right) \vee \forall C$$

Thus, there are two cases:
(1) $M \models \mathcal{F}\left(\overline{t}\right)$: It is obvious that $M \models \mathcal{F}\left(T+^B E\right)$, by (4.3).
(2) $M \models C$: Thus, $M \models C'$, and hence $M \models E$. Thus, $M \models \mathcal{F}\left(T+^B E\right)$, by the Extending Lemma. □

The redundancy criterion based on this theorem can be formulated as follows:

Definition 4.13 (Redundancy Criterion II). A clause E is redundant w.r.t. a branch B of a clausal tableau T iff there is a separate branch clause C of B in T such that E is subsumed by a new instance of C.

Example 4.14 (Redundancy Criterion II). Consider the clausal tableau T in Figure 5. Check if

$$E = P(f(a)) \vee \neg Q(f(f(a)))$$

is redundant w.r.t. T. When applying Redundancy Criterion I, the answer is no. However, by applying Redundancy Criterion II, E can be shown to be redundant w.r.t. the rightmost branch, since the new separate branch clause instance $P(z')$ subsumes E, via the variable substitution $z'/f(a)$.

Note that none of all the other separate branch clauses subsumes E, e.g., neither $P(x) \vee \neg Q(x)$ (lack of an appropriate variable substitution) nor $R(y) \vee P(y) \vee \neg Q(f(y))$. Note that E is redundant w.r.t. only the rightmost branch.

The following fact is quite interesting, and shows that some redundancy criteria in literature are specialized variants of Redundancy Criterion II.

Example 4.15 (Redundancy Criterion II for Hyper Tableaux). Let us focus on the redundancy criterion for hyper tableaux (c.f. Example 4.2, [2]). Note that it can be regarded as a special case of Redundancy Criterion II. The literal L_2 is a *separate branch clause* because of the use of *purifying substitutions* [2].

5. Conclusion

Note that Theorem 4.12 is actually applied in resolution tableaux. In a resolution tableau (however it is actually not a clausal tableau), each vertex label can be regarded as a *separate branch clause* in itself, this is why its *new instances* can be generated and used in any resolution inference step. Of course, Redundancy Criterion II (c.f. Definition 4.13) is also appropriate to be used by any resolution tableau calculus, and can be applied in a very direct way. The key is the fact that any tableau literal belongs to exactly one separate branch clause (this fact does not in general hold for clausal tableaux).

One can say that resolution tableaux have been defined according to plan. It is not accidental that vertices can be labeled not solely with literals but clauses. It is not accidental either that vertex labels are independent from each other. This solution can be regarded as the golden mean between resolution calculi and clausal tableau calculi, according to 4.12.

By the use of resolution tableaux, a resolution derivation can be split into branches, and branches (as separate derivations) can be continued simultaneously. I.e., resolution tableaux can be regarded as a kind of parallelization of resolution. Nevertheless, one can ask if it is worth to apply an additional task (namely the splitting of clauses into independent subclauses) in comparison with the advantage of shortening resolution derivations. This question could only be answered by empirical investigations.

5.1. Empirical investigations

In order to examine the practical usefulness of resolution tableaux, we implemented four different resolution calculi: *binary resolution*, *linear resolution*, *linear input resolution*, and *hyper-resolution*. Let us emphasize that purely the basic variants of those calculi have been implemented. We also implemented improved variants of the aforementioned calculi, only by applying resolution tableaux.

Then, we tested all the original and improved calculi on 1642 TPTP problems [16] (from 232 files). As it had been expectable, those calculi could not solve most of the problems in a reasonable time limit. What we primarily tried to investigate are the following questions:

- *How often* an improved calculus can solve such a problem that the original calculus cannot solve?
- If both an original calculus and its improved variant can solve a problem, *how much time* is gained by using the improved calculus?

Table 1 contains all the statistical data we have collected. Let us give an overview on the columns of the table:

- In connection with those cases when either an original calculus or its improved variant does not provide a solution (in a reasonable time limit) for the same problem, let us summarize the following data:

	Solution		Time		
	gained	*lost*	*gained*	*lost*	*gained/lost*
Binary	0.49%	0.59%	10.91%	29.09%	2.93%
Linear	12.3%	0%	7.45%	0.62%	626.67%
Linear input	27.57%	0%	4.48%	0%	–
Hyper	2.01%	0.97%	46.46%	18.77%	310.43%

Table 1: Empirical results.

1. **Gained answers:** The frequency of those cases when the original calculus does not provide a solution, but the improved calculus *does*.
2. **Lost answers:** The frequency of those cases when the original calculus provides a solution, but the improved calculus *does not*.

- In connection with those cases when both an original calculus and its improved variant provide a solution for the same problem, let us summarize the following data:

  1. **Gained time:** The frequency of those cases when the improved calculus provides a solution in *less time* than the original calculus.
  2. **Lost time:** The frequency of those cases when the improved calculus provides a solution in *more time* than the original calculus.
  3. **Gained time/Lost time:** We calculated the ratio of the length of the gained time to the length of the lost time in order to illustrate how it is worth to apply the improved calculus, in respect to execution time.

As it can be noticed, binary resolution tableaux do not seem very practical, in contrast with linear resolution tableaux and linear input resolution tableaux, which are absolutely worth to apply.

The conclusion in the case of hyper-resolution tableaux is quite ambiguous. Since hyper-resolution itself can be regarded as a quite powerful proof method, only in a few cases can hyper-resolution tableaux provide extra solutions. Nevertheless, the frequency of the cases when hyper-resolution tableaux shorten execution time is extremely high.

References

[1] Bachmair, L., Ganzinger, H., Resolution Theorem Proving, in: J. A. Robinson, A. Voronkov, Handbook of Automated Reasoning, Elsevier and MIT Press, North-Holland, Amsterdam, 2001, Vol. 1, Chapter 2, pp. 19–99.

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

[3] Baumgartner, P., Hyper Tableaux – The Next Generation, *Lecture Notes in Artificial Intelligence*, Vol. 1397, 1998, pp. 60–76.

[4] Boyer, R.S., Locking: A Restriction of Resolution, PhD thesis, University of Thexas at Austin, Austin, USA, 1971.

[5] Chang, C.L., Lee, R.C.T., Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1973.

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

[7] Hähnle, R., Tableaux and Related Methods, in: J. A. Robinson, A. Voronkov, Handbook of Automated Reasoning, Elsevier and MIT Press, North-Holland, Amsterdam, 2001, Vol. 1, Chapter 3, pp. 100–178.

[8] Kovásznai, G., HyperS Tableaux – Heuristic Hyper Tableaux, *Acta Cybernetica*, Vol. 17, 2005, pp. 325–338.

[9] Kovásznai, G., Multi-Hyper Tableaux in Automated Theorem Proving, PhD thesis (in Hungarian), University of Debrecen, Debrecen, Hungary, 2007.

[10] Kowalski, R.A., Logic for Problem Solving, Elsevier North Holland, Amsterdam, 1979.

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

[12] Letz, R., Schumann, J., Bayerl, S., Bibel, W., SETHEO: A High-Performance Theorem Prover, *Journal of Automated Reasoning*, Vol. 8(2), 1992, pp. 183–212.

[13] Robinson, J.A., A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the ACM*, Vol. 12, 1965, pp. 23–41.

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

[15] Smullyan, R.M., First-Order Logic, Springer-Verlag, Berlin, Germany, 1968.

[16] Sutcliffe, G., The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0, *Journal of Automated Reasoning*, Vol. 43(4), 2009, pp. 337–362.

Gergely Kovásznai
Department of Information Technology
Eszterházy Károly College
P.O. Box 43
H-3301 Eger
Hungary
e-mail: `kovasz@aries.ektf.hu`

Gábor Kusper
Department of Computing Science
Eszterházy Károly College
P.O. Box 43
H-3301 Eger
Hungary
e-mail: `gkusper@aries.ektf.hu`