Reduction Rules and Universal Variables for First Order Tableaux and DPLL

Fabio Massacci
Dipartimento di Ingegneria dell'Informazione
Università di Siena
via Roma 56, 53100 Siena, Italy,
e-mail: massacci@dii.unisi.it

Abstract

Recent experimental results have shown that the strength of resolution, the propositional DPLL procedure, the KSAT procedure for description logics, or related tableau-like implementations such as DLP, is due to reduction rules which propagate constraints and prune the search space.

Yet, for first order logic such reduction rules are only known for resolution. The lack of reduction rules for first order tableau calculi (DPLL can be seen as a tableau calculus with semantic branching) is one of the causes behind the lack of efficient first order DPLL-like procedures.

The difficulty is that first order splitting rules force tableau and DPLL calculi to use rigid variables which hinder reduction rules such as unit subsumption or unit propagation.

This paper shows how reduction and simplification rules can be lifted to a first order tableau-like calculus using universal variables and a new rule called renaming. It also shows how to optimize the calculus for exploiting universal variables and reduction rules when semantic branching aka split DPLL-style is chosen as the inference rule with DFS-search.

1 Introduction

The last years have shown a clear trend in the development of efficient and effective procedures for automated reasoning. The renewed interest for experimental analysis and competition between systems has shown that what really makes a practical difference between theorem provers or satisfiability checkers is their ability of pruning the search space.

The importance of rules for reducing the search space has been stressed since the very beginning in the design of the Davis-Putnam-Loveland-Procedure [11, 10] and in the early Russian studies in automated deduction [28]. The ability of performing fast and effective unit propagation has been a key for the successes in finite algebra of the theorem prover SATO by Zhang [33].

In the realms of modal and description logics, a major qualitative advance has been the introduction of the same pruning techniques of DPLL in the KSAT decision procedure for modal logic K by Giunchiglia and Sebastiani [16]. Further experimental analysis by Hustadt and Schmidt [20] has confirmed the importance of reduction rules over inference rules: once reduction rules were added to the tableau system KRIS its performance increased by orders of magnitude. The DLP system by Patel-Schneider, "winner" of the recent TANCS competition of modal logics [24], uses many reduction rules such as boolean constraint propagation [19].

In first order logic, the importance of reduction rules for resolution (such as subsumption) has been stressed since its inception by Robinson [27] and its implementation has been the subject of an intense study (see e.g. [14, 30]). The competitive results by Hustadt and Schmidt [20] for the translation of modal logic into first order logic are also due to the subsumption routines of the prover SPASS. The performance of Vampire at the last CASC competition is mainly due to its sophisticated subsumption architecture by Voronkov [30]

If we were allowed to condense these experimental findings in one sentence we could just say that *reduction rules are more effective than inference rules.* In practice, the most (if not the only) effective reduction rules are those which act locally, affect one or few formulae at the time and do not require to compare whole parts of the proof search tree. The locality of the rules makes then possible optimized implementations in which such rules are applied to all formulae at once [30, 33].

The practical importance of local reduction rules also explains one surprising duality in experimental results: in the propositional and modal realms, tableau-like systems (DPLL can be seen as a tableau calculus using semantic branching) are the undisputed front runners; in the first-order domain, systems based on hyper-resolution or superposition (e.g. Vampire, Gandalf, SPASS) take the front stage.

Indeed, for propositional, modal, and description logics, local, formula-wise reduction rules are well studied and go along with different forms of tableau calculi (i.e. branching rules) and can fully exploit the features of the underlying logic [16, 18, 19, 23].

In contrast, for first order logic, a key formalisms for knowledge representation, there is only one competitor which fully exploits first order features: the subsumption rule for resolution (and superposition). Techniques for search pruning in tableau-based approach such as factoring and merging do exist but they require considering whole parts of the proof tree [1, 21, 32]. In other cases, such as the first order version of the DPLL procedure [11], reduction rules are gained only by grounding formulae to propositional logic.

The major problem is that first order branching rules force tableau, model elimination, and DPLL-like calculi to use rigid variables[1] and this substantially hinders the ability of performing first order local reduction rules such as unit subsumption or unit propagation.

This paper proposes a general solution based on the observation that in the resolution framework all variables are "implicitly quantified". We propose a modified tableau calculus in which all variables are also implicitly universally quantified unless we explicitly mark them as rigid. Then, we introduce a rule called *renaming* which generalizes the use of universal variables proposed by Beckert et al. [6, 7] and by Baumgarten [4], and the use of conjunctive superformulae proposed by Degtyarev and Voronkov [13].

In this new framework we can now introduce full first order reduction rules for tableau-like calculi and prove them sound (and sometimes invertible). We also show how the calculus can be optimized to fully exploit universal variables and reduction rules when semantic branching aka split DPLLstyle is chosen as the main inference rule with a depth-first-search strategy and discuss how a lean implementation in sicstus prolog can be done for this calculus.

The introduction of first order reduction rules provides also an *unifying perspective* of many (tableau based) deduction techniques and "explains away" the stand-alone (inefficient) nature of the classical first order DPLL.

[1] Loosely speaking, when we split the search in two branches with $P(x)$ on one side and and $\neg P(x)$ on the other, the variable x in $P(x)$ is called rigid because choosing its value affects another branch. As a consequence, $P(x)$ cannot be used to unit-subsume $P(b) \vee Q$, because we are not free to let x range over all possible values.

2 Notation and Terminology

We assume a basic knowledge of first order syntax and semantics [15, 29]. First order *terms*, denoted by t, are constructed from constants $a, b, c \in \mathcal{C}on$ and variables $x, y, z \in \mathcal{V}ar$ using functions $f, g \in \mathcal{F}un$. The usual definition of ground terms (terms without variables) is adopted.

We construct *formulae* A, B, C from predicate symbols $P(t_1, \ldots, t_n)$ and $Q(t_1, \ldots t_n)$ and the boolean constants $\top$ and $\bot$ with the propositional connectives $\wedge$, $\vee$, $\neg$, and the first order connectives $\forall$ and $\exists$: e.g. $A \wedge B, \forall x.A, \exists x.A$. Other connectives can be seen as abbreviations, e.g. $A \supset B \doteq \neg A \vee B$.

If A is a formula, its *conjugate* $\overline{A}$ is obtained in the standard way: $\overline{\neg A} = A$ and $\overline{A} = \neg A$ if the main connective of A is not a negation.

By $FV(A)$ we denote the *set of free variables occurring in the formula* A. By $A(x)$ we mean that the variable x may occur free in the formula A and $A(t)$ denotes the simultaneous substitution of t in all (if any) occurrences of x in A. $FV(\cdot)$is extended to set of formulae or set of terms in the obvious way. A *sentence* is a formula A such that $FV(A) = \emptyset$.

By $\{t_1/x_1, \ldots, t_n/x_n\}$ we denote the *substitution* which simultaneously replaces all occurrences of each variable x_i with the corresponding term t_i. Substitutions are usually denoted by σ. A *renaming*, denoted by η, is a particular substitution $\{y_1/x_1, \ldots, y_n/x_n\}$ where all y_i are fresh variables.

By $A\sigma$ we denote result of the application of the substitution σ to the formula A. We employ the usual concept of *substitution free for a formula* A, i.e. the variables of t_i do not get bound after the substitution of t_i for x_i. To this extent we rename, if necessary, bound variables in a formula before applying a substitution. In the sequel substitutions are assumed free.

The *domain* of a substitution $\{t_1/x_1, \ldots, t_n/x_n\}$ is the set of variables $\{x_1, \ldots, x_n\}$. If $\mathcal{R}$ is a set of variables we denote by $\sigma_{\overline{\mathcal{R}}}$the restriction of the domain of σ to variables which are *not* in $\mathcal{R}$, i.e. $domain(\sigma_{\overline{\mathcal{R}}}) \cap \mathcal{R} = \emptyset$.

3 A First Order Calculus with Renaming

Before introducing first order reduction rules, we need a calculus which can keep track of rigid and universal (i.e. implicitly universally quantified) variables. Standard free-variables tableau calculi use only rigid variables [26, 15, 13, 31]; more advanced methods make a limited use of universal variables [6, 7, 5, 4]. Here all variables are assumed to be universal ones unless we explicitly consider them rigid.

Definition 1 *A* prefixed formula *is a pair $\mathcal{R}$:A where A is a first order formula and $\mathcal{R}$ is a set of variables. A* prefixed sequent*, denoted by $\mathcal{S}$, is a set of prefixed formulae.*

The variables in $\mathcal{R}$ are the only *rigid variables* whose value may affect other parts of the proof search tree. All free variables in A which are not in $\mathcal{R}$ are implicitly universally quantified. In the sequel we use $\mathcal{S}, \mathcal{R}{:}A$ as a short-cut for $\mathcal{S} \cup \{\mathcal{R}{:}A\}$ and $\mathcal{R}, x$ as a shortcut for $\mathcal{R} \cup \{x\}$.

The definition of search tree is standard [15, 29]:

Definition 2 *A* proof search tree $\mathcal{T}$ *for a formula A is a dyadic tree where each node is labelled by a prefixed sequent so that*

- *the root is labelled by the prefixed sequent* $\{FV(A) : \neg A\}$*;*
- *if a node is labelled by a sequent its children are labelled with the corresponding consequents of a rule of the calculus from Fig. 1 and Fig. 2.*

Intuitively, we may say that a child of a node is actually labelled by replacing one formula in the sequent of its parents by the consequence of a rule applied to that formula, and then copying the other formulas in the sequent unchanged.

Remark. For what regard visual presentation, the *proof search tree grows upward*, with the sentence to be proved at the bottom and the leaves (axioms) at the top.

In the sequel we use the notion of a *frontier* of a proof search tree $\mathcal{T}$: it is the collection of all leaves of the tree and is denoted by $\mathcal{S}_1 \mid \ldots \mid \mathcal{S}_n$. If we view the proof search tree as a tableau, the frontier is simply the collection of the current branches of the tableau [7, 13, 15].

Figure 1 contains rules which do not introduce new branches in the proof search tree. A general observation is that we assume formulae to be "definitively"

$$\frac{\mathcal{S}, \mathcal{R}{:}A, \mathcal{R}{:}B}{\mathcal{S}, \mathcal{R}{:}A \wedge B}\ \alpha_\wedge \qquad \frac{\mathcal{S}, \mathcal{R}{:}\neg A, \mathcal{R}{:}\neg B}{\mathcal{S}, \mathcal{R}{:}\neg(A \vee B)}\ \alpha_\vee \qquad \frac{\mathcal{S}, \mathcal{R}{:}A}{\mathcal{S}, \mathcal{R}{:}\neg\neg A}\ \alpha_\neg$$

$$\frac{\mathcal{S}, \mathcal{R}{:}A(y)}{\mathcal{S}, \mathcal{R}{:}\forall x.A(x)}\ \gamma_\forall \qquad \frac{\mathcal{S}, \mathcal{R}{:}\neg A(y)}{\mathcal{S}, \mathcal{R}{:}\neg\exists x.A(x)}\ \gamma_\exists$$

$$\frac{\mathcal{S}, \mathcal{R}{:}\neg A(f_A(x_1 \ldots x_n))}{\mathcal{S}, \mathcal{R}{:}\neg\forall x.A(x)}\ \delta_\forall \qquad \frac{\mathcal{S}, \mathcal{R}{:}A(f_A(x_1 \ldots x_n))}{\mathcal{S}, \mathcal{R}{:}\exists x.A(x)}\ \delta_\exists$$

$$\frac{\mathcal{S}, \mathcal{R}{:}A\eta_{\overline{\mathcal{R}}}}{\mathcal{S}, \mathcal{R}{:}A}\ (ren) \qquad \frac{\mathcal{S}, \mathcal{R} \cap FV(A){:}A}{\mathcal{S}, \mathcal{R}{:}A}\ (norm)$$

Where y is a fresh variable in the γ-rules; f_A is a skolem function symbol and $\{x_1, \ldots x_n\}$ is the set of free variables of A in the δ-rules.

Figure 1: First order calculus with renaming

reduced after an application of a rule, unless they are not explicitly duplicated in the consequent(s). So we only have duplication in the β-rules of Fig. 2.

There are various ways to optimize the generation of skolem functions in the δ rules for the existential quantifiers. This topic is orthogonal to the one we are treating here, and we refer to the literature for discussion [17, 3].

The key rule is the (ren) rule, which we use to boost the usage of universal variables. Indeed its task is to minimize the number of variables that are shared between formulae. In this way, the substitution of a term for a variable in a formula will not affect other formulae, unless the variable is rigid. Notice that the renaming $\eta_{\overline{\mathcal{R}}}$ is *local* to the formula A: it is not applied to the whole sequent $\mathcal{S}$ nor, a fortiori, to the whole search tree. This renaming rule is such that the standard soundness proof of rigid and universal variables tableau calculi fails for it, so we will use a novel technique to prove its soundness.

In general, renaming and normalization should be immediately applied to the formulae newly introduced by other rules.

Figure 2 contains two alternative set of rules for branching. The first set is a variant of the classical β-rule of tableau calculi whereas the second set of rules is usually called semantic branching (see e.g. the modal optimizations by Horrocks and Patel-Schneider [19]).

Branching rules are responsible for the introduction of rigid variables. Indeed if we have $A(x) \vee B(x)$ and found out that $A(x)$ is contradictory for some substitution t/x, we have to propagate this substitution also to the branch of the search tree containing the disjunct $B(x)$. This operation is costly and, if the implementation does not guarantee proof-confluence as in [4]

Rules for symmetric branching:

$$\frac{\mathcal{S}, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}A \quad \mathcal{S}, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}B}{\mathcal{S}, \mathcal{R}{:}A \vee B}\ \beta_\vee \qquad \frac{\mathcal{S}, \mathcal{R}{:}\neg(A \wedge B), \mathcal{R}_{A*B}{:}\neg A \quad \mathcal{S}, \mathcal{R}{:}\neg(A \wedge B), \mathcal{R}_{A*B}{:}\neg B}{\mathcal{S}, \mathcal{R}{:}\neg(A \wedge B)}\ \beta_\wedge$$

Rules for semantic branching:

$$\frac{\mathcal{S}, \mathcal{R}{:}A \vee B, \mathcal{R}_A{:}A \quad \mathcal{S}, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}B, \mathcal{R}_A{:}\neg A}{\mathcal{S}, \mathcal{R}{:}A \vee B}\ s\beta_\vee \qquad \frac{\mathcal{S}, \mathcal{R}{:}\neg(A \wedge B), \mathcal{R}_A{:}\neg A \quad \mathcal{S}, \mathcal{R}{:}\neg(A \wedge B), \mathcal{R}_{A*B}{:}\neg B, \mathcal{R}_A{:}A}{\mathcal{S}, \mathcal{R}{:}\neg(A \wedge B)}\ s\beta_\wedge$$

where $\mathcal{R}_{A*B} = \mathcal{R} \cup (FV(A) \cap FV(B))$ and $\mathcal{R}_A = \mathcal{R} \cup FV(A)$

Figure 2: Symmetric and semantic branching rules for the first order calculus with renaming

we might be forced to backtrack over previous unifications. Thus, a major task is to minimize the number of rigid variables introduced after a branching point.

Symmetric branching allows for a good minimization of rigid variables in both branches: the rigid variables introduced by the symmetric branching rule are only the free variables of A and B which occur in both conjuncts. We shall see that we can do better once a depth-first strategy is assumed. Unfortunately, semantic branching is not equally well suited. To retain soundness we are forced to keep rigid all variables of the first disjunct A. An alternative is to introduce new skolem terms in the $\neg A$ added on the right branch and we have not pursued it here.

The differences between branching rules can be intuitively explained by considering the formula $\forall xyz.P(x,y) \vee Q(y,z)$. This formula is logically equivalent to the following three formulae:

1. $\forall y.\,(\forall x.P(x,y)) \vee (\forall z.Q(y,z))$
2. $\forall yx.P(x,y) \vee (\neg P(x,y) \wedge \forall z.Q(y,z))$
3. $\forall y.(\forall x.P(x,y)) \vee ((\exists x.\neg P(x,y)) \wedge (\forall z.Q(y,z)))$

Loosely speaking, the first alternative is what our rule for symmetric branching does. The second alternative is what we have chosen for semantic branching and the third one is what we decided to discard, because the optimized version of the semantic branching rule makes it possible to simulate its effects without introducing new skolem terms (see section 5).

One may also wonder why we do not just pre-process the input following the intuition above: after all this is a variant of a technique known as *mini-scoping* which is used for skolemization by theorem provers like SPASS. The problem is that mini-scoping is static, whereas in with an explicit representation of rigid variables we can benefit from the universal variables that are introduced during the search.

We can now state what a proof is:

Definition 3 *A sequent $\mathcal{S}$ is closed by a substitution σ iff either $\mathcal{R}{:}\neg\top \in \mathcal{S}$, $\mathcal{R}{:}\bot \in \mathcal{S}$, or there are two formulae $\mathcal{R}_A{:}A \in \mathcal{S}$ and $\mathcal{R}_B{:}B \in \mathcal{S}$ such that $A\sigma = \overline{B\sigma}$.*

Definition 4 *A pair $\langle \mathcal{T}, \sigma_\mathcal{T} \rangle$, where $\mathcal{T}$ is a search tree and σ a substitution, is a* proof *for the sentence A if*

- *$\{\emptyset{:}\neg A\}$ labels the root of the tree $\mathcal{T}$;*
- *each $\mathcal{S}_i$ in the frontier of $\mathcal{T}$ is closed by $\sigma_\mathcal{T}$*

We refer to $\sigma_\mathcal{T}$ as a *closing substitution* for $\mathcal{T}$. We have chosen this formulation so that one can apply the substitution $\sigma_\mathcal{T}$ to the whole tableau and obtain a result close to a standard proof search tree of the ground version of tableaux [15] or DPLL [11].

In alternative, if we choose to represent only the frontier of the tree, as done by Beckert and Possega [7] or Degtyarev and Voronkov [13], we could have described the proof as a sequence of tableau and substitutions as in [7] or the sequent elimination rule (*abc*) as in [13]. For instance, the (*abc*) rule allows to reduce the frontier of not-closed sequents $\mathcal{S}_1 \mid \mathcal{S}_2 \mid \ldots \mid \mathcal{S}_n$ into the frontier $\mathcal{S}_2\sigma \mid \ldots \mid \mathcal{S}_n\sigma$ provided that σ is a closing substitution for $\mathcal{S}_1$.

Since the definition of proof is fairly close to the classical one, we may wonder where renaming plays a role. The key observation is that, if we apply renaming to all formulae of a leaf then the only shared variables between two leaves will be the rigid variable introduced at some branching points. After renaming we can use the same formula for closing different branches. For instance, one may consider the formula $(\forall x.P(x)) \wedge (\neg P(a) \vee P(b))$. We have two branches $\{\emptyset{:}\neg P(a), \emptyset{:}P(x)\}$ and $\{\emptyset{:}\neg P(b), \emptyset{:}P(x)\}$ and we can close them simultaneously rename x into x_1 in the first branch and into x_2 in the second one.

Renaming makes us available as many instances of non-rigid variables as we want. Thus, if new rigid variables are not introduced, an optimization is possible:

Proposition 1 *In the branching rules of Figure 2, the disjunction $A \vee B$ can be deleted from both consequents if it is a ground formula.*

The first result of this paper is then the following:

Theorem 1 *A sentence A is valid iff there is a proof for A.*

So far we have not tackled the issue of fairness or termination. The simplest (but least effective) technique is to use a bound on the number of times that a disjunction can be duplicated by the β-rules and then use backtracking over all possible unifications for a given bound. Mutatis mutandis, this strategy is employed by the standard prolog implementation of first order tableau calculi [15, 7].

Advanced implementations, such as the Hypertableaux calculus [5], use a bound on the depth of terms that are used by the unification $\sigma_{\mathcal{T}}$ for closing each branch. This technique can be lifted to this framework by introducing a notion of reduced formula (modulo renaming), using depth first search and forbidding substitutions that, when applied to the current sequent to close it, generate two instances of the same disjunction. We sketch its implementation in section 6.

4 First Order Reduction Rules

The next step is the definition of the reduction rules.

The basic operation is *boolean reduction*. This operation, sometimes called lexical normalization or boolean constraint propagation, is one of the strength of description logic theorem provers such as DLP [19], KSAT [16] or FaCT [18]. Advanced modal and propositional reductions have been also proposed by Massacci [23].

We denote the result of boolean reduction of a formula A by $A\downarrow$, and the corresponding rules are shown in Fig. 3. Although this reduction already yields a substantial simplification of a formula, it is still not sufficient as it does not fully exploit the power of first order logic.

For first order reductions we must take care of rigid variables. Thus, if $\mathcal{R}$ is a set of rigid variables, we denote the result of the *first order reduction modulo* $\mathcal{R}$ of a formula A as $(A)\Downarrow_{\mathcal{R}}$. Corresponding rules are in Fig. 4.

The main intuition is that we cannot use rigid variables for unification and therefore we must keep track of which variable is rigid and which is (implicitly) universal. This explains the treatment of the existentially quantified variables, which are added to the set of rigid variables as the reduction proceeds down to the subformulae, and the treatment of disjunction, in which we must add the set of variables shared between the two disjuncts to the set of rigid variables. Once rigid variables are set aside, we can exploit the fact that all other variables are implicitly universally quantified.

For example, in the replacement of two conjuncts by $\top$, suppose we have the formula $\forall xyz.(P(x,a) \wedge \neg P(f(y),y)) \vee Q(z)$. Clearly, the variables x, y are implicitly universally quantified. So, whenever this formula is introduced in the proof search tree we can apply a γ rule, a β and then an α rule. The resulting formulae $P(x,a)$ and $\neg P(f(y),y)$, thanks to renaming, can immediately close the sequent without affecting any other part of the search tree with the substitution $\{f(a)/x, a/y\}$. Thus, we can gain from this knowledge and immediately replace our subformula with $\bot$.

For the merging of two conjuncts into one formula, suppose that we have the formula $\forall xy.P(f(x)) \wedge P(y)$. Whenever this formula is added to the proof search tree we obtain $\emptyset : P(f(x))$ and $\emptyset : P(y)$. Now the second formula obviously subsumes the first. With renaming we can generate fresh instances of $P(y)$ in different sequent leaves of the proof search tree and we never need to use $P(f(x))$. Thus, we keep only the most general conjunct.

Disjunction is subtler. At first we have not used unification but only identity for the $\top$ clause of the disjunction. If we had used unification the calculus would have been incomplete. Consider the formula

$$(\neg P(a) \vee \neg P(b)) \wedge (P(c) \vee P(d)) \wedge (\forall x.\forall y.P(x) \vee \neg P(y)).$$

This formula is unsatisfiable but if we had applied first order reduction using unification we would have "reduced" it to a satisfiable formula.

Remark. The conditions for merging two disjuncts into one disjunct is the dual of the condition used for conjunction: we keep only the most specific disjunct. This is the only place where we must give away equivalence preserving rules in favor of satisfiability preserving rules.

Finally we can devise the last and most important reduction rule: *simplification*. The intuition is that once a subformula is added to a sequent, we commit to consider it true and then we may simplify other formulae using this information. We denote the simplification of a formula A using another prefixed formula $\mathcal{R}{:}B$ as $A\,[\mathcal{R} : B]$ and show its working in Fig. 5. The formula A is sometimes referred to as the simplified formula and the formula B as the simplifying formula.

Notice that the substitution $\sigma_{\overline{\mathcal{R}}}$is not applied to the

$$
\begin{array}{rcl}
(A \wedge B)\downarrow & = & \begin{cases} \bot & \text{if } A\downarrow = \bot \text{ or } B\downarrow = \bot \text{ or } A\downarrow = \overline{B\downarrow} \\ A\downarrow & \text{if } A\downarrow = B\downarrow \\ A\downarrow \wedge B\downarrow & \text{otherwise} \end{cases} \\
(A \vee B)\downarrow & = & \begin{cases} \top & \text{if } A\downarrow = \top \text{ or } B\downarrow = \top \text{ or } A\downarrow = \overline{B\downarrow} \\ A\downarrow & \text{if } A\downarrow = B\downarrow \\ A\downarrow \vee B\downarrow & \text{otherwise} \end{cases} \\
\neg\neg A\downarrow & = & A\downarrow \\
\neg(A \wedge B)\downarrow & = & \begin{cases} \top & \text{if } \neg A\downarrow = \top \text{ or } \neg B\downarrow = \top \text{ or } \neg A\downarrow = \overline{\neg B\downarrow} \\ \neg A\downarrow & \text{if } \neg A\downarrow = \neg B\downarrow \\ \neg(A\downarrow \wedge B\downarrow) & \text{otherwise} \end{cases} \\
\neg(A \vee B)\downarrow & = & \begin{cases} \bot & \text{if } \neg A\downarrow = \bot \text{ or } \neg B\downarrow = \bot \text{ or } \neg A\downarrow = \overline{\neg B\downarrow} \\ \neg A\downarrow & \text{if } \neg A\downarrow = \neg B\downarrow \\ \neg(A\downarrow \vee B\downarrow) & \text{otherwise} \end{cases}
\end{array}
$$

Figure 3: Boolean reduction rules

$$
\begin{array}{rcl}
(\neg\neg A)\Downarrow_{\mathcal{R}} & = & (A)\Downarrow_{\mathcal{R}} \\
(\forall x.A)\Downarrow_{\mathcal{R}} & = & \begin{cases} (A)\Downarrow_{\mathcal{R}} & \text{if } x \notin FV((A)\Downarrow_{\mathcal{R}}) \\ \forall x.(A)\Downarrow_{\mathcal{R}} & \text{otherwise} \end{cases} \\
(\exists x.A)\Downarrow_{\mathcal{R}} & = & \begin{cases} (A)\Downarrow_{\mathcal{R},x} & \text{if } x \notin FV((A)\Downarrow_{\mathcal{R},x}) \\ \forall x.(A)\Downarrow_{\mathcal{R},x} & \text{otherwise} \end{cases} \\
(\neg\forall x.A)\Downarrow_{\mathcal{R}} & = & \begin{cases} (\neg A)\Downarrow_{\mathcal{R},x} & \text{if } x \notin FV((\neg A)\Downarrow_{\mathcal{R},x}) \\ \neg\forall x.\neg((\neg A)\Downarrow_{\mathcal{R},x})\downarrow & \text{otherwise} \end{cases} \\
(\neg\exists x.A)\Downarrow_{\mathcal{R}} & = & \begin{cases} (\neg A)\Downarrow_{\mathcal{R}} & \text{if } x \notin FV((\neg A)\Downarrow_{\mathcal{R}}) \\ \neg\exists x.\neg((\neg A)\Downarrow_{\mathcal{R},x})\downarrow & \text{otherwise} \end{cases} \\
((A \wedge B))\Downarrow_{\mathcal{R}} & = & \begin{cases} \bot & \text{if } (A)\Downarrow_{\mathcal{R}} = \bot \text{ or } (B)\Downarrow_{\mathcal{R}} = \bot \\ & \text{or } ((A)\Downarrow_{\mathcal{R}})\sigma_{\overline{\mathcal{R}}} = \overline{((B)\Downarrow_{\mathcal{R}})\sigma_{\overline{\mathcal{R}}}} \text{ for some substitution } \sigma \\ (A)\Downarrow_{\mathcal{R}} & \text{if } ((A)\Downarrow_{\mathcal{R}})\sigma_{\overline{\mathcal{R}}} = (B)\Downarrow_{\mathcal{R}} \\ (B)\Downarrow_{\mathcal{R}} & \text{if } (A)\Downarrow_{\mathcal{R}} = ((B)\Downarrow_{\mathcal{R}})\sigma_{\overline{\mathcal{R}}} \\ (A)\Downarrow_{\mathcal{R}} \wedge (B)\Downarrow_{\mathcal{R}} & \text{otherwise} \end{cases} \\
((A \vee B))\Downarrow_{\mathcal{R}} & = & \begin{cases} \top & \text{if } (A)\Downarrow_{\mathcal{R}_{A*B}} = \top \text{ or } (B)\Downarrow_{\mathcal{R}_{A*B}} = \top \\ & \text{or } (A)\Downarrow_{\mathcal{R}_{A*B}} = \overline{(B)\Downarrow_{\mathcal{R}_{A*B}}} \\ (A)\Downarrow_{\mathcal{R}_{A*B}} & \text{if } (A)\Downarrow_{\mathcal{R}_{A*B}} = ((B)\Downarrow_{\mathcal{R}_{A*B}})\sigma_{\overline{\mathcal{R}_{A*B}}} \\ (B)\Downarrow_{\mathcal{R}_{A*B}} & \text{if } ((A)\Downarrow_{\mathcal{R}_{A*B}})\sigma_{\overline{\mathcal{R}_{A*B}}} = (B)\Downarrow_{\mathcal{R}_{A*B}} \\ (A)\Downarrow_{\mathcal{R}_{A*B}} \wedge (B)\Downarrow_{\mathcal{R}_{A*B}} & \text{otherwise} \end{cases}
\end{array}
$$

Figure 4: First order reduction

$$
A\,[\mathcal{R} : B] \quad = \quad \begin{cases} \top & \text{if } A = B\sigma_{\overline{\mathcal{R}}} \text{ for some } \sigma \\ \bot & \text{if } \overline{A} = B\sigma_{\overline{\mathcal{R}}} \text{ for some } \sigma \\ \neg(C\,[\mathcal{R} : B]) & \text{if } A = \neg C \\ (C\,[\mathcal{R} : B]) \wedge (D\,[\mathcal{R} : B]) & \text{if } A = C \wedge D \\ (C\,[\mathcal{R} : B]) \vee (D\,[\mathcal{R} : B]) & \text{if } A = C \vee D \\ \forall x(C\,[\mathcal{R} : B]) & \text{if } A = \forall x C \\ \exists x(C\,[\mathcal{R} : B]) & \text{if } A = \exists x C \end{cases}
$$

Figure 5: First order simplification

$$\frac{\mathcal{S}, \mathcal{R}{:}A \downarrow}{\mathcal{S}, \mathcal{R}{:}A}\ bool - red \quad \frac{\mathcal{S}, \mathcal{R}{:}(A) \Downarrow_{\mathcal{R}}}{\mathcal{S}, \mathcal{R}{:}A}\ fo - red$$

$$\frac{\mathcal{S}, \mathcal{R}_A{:}A\left[\mathcal{R}_B : B\right], \mathcal{R}_B{:}B}{\mathcal{S}, \mathcal{R}_A{:}A, \mathcal{R}_B{:}B}\ simp$$

Figure 6: First order reduction rules

tableau nor the simplified formula, nor it modifies the rigid variables in B. This is in common to many techniques used by clausal tableaux [5, 21]. We go further than that since different substitutions can be used for different occurrences of a subformula. So the substitution is local to the subformula.

For instance, if we apply simplification as follows

$$\forall y.((\neg P(x,a) \vee Q(y)) \wedge (Q(b) \vee \neg P(x,b)))\,[\{x\} : P(x,y)]$$

we can use the substitution a/y for $P(x,a)$ and the substitution b/y for $P(x,b)$. Then we get the formula $\forall y.(\bot \vee Q(y)) \wedge (Q(b) \vee \bot))$ and by first order reduction we get just $\forall y.Q(y)$.

The proper reduction rules for our first order calculus with renaming are shown in Fig. 6.

Then we can state the other main results of this paper:

Theorem 2 *The boolean reduction rule is a sound and invertible rule for the first order calculus with renaming.*

Theorem 3 *The first order reduction rule is a sound rule for the first order calculus with renaming.*

Theorem 4 *The first order simplification rule is a sound and invertible rule for the first order calculus with renaming.*

5 Optimizations for Branching Rules

The usage of universal variables in the branching rules presented in Fig. 2 can be further optimized if a depth first strategy is used. The optimization is also necessary for semantic branching since we have only a limited number of universal variables in one branch due to the presence of $\mathcal{R}_A{:}A$ on one branch and $\mathcal{R}_A{:}\neg A$ on the other of branch the search tree.

The intuition is that with a depth first strategy the amount of information that is available when the right branch of a disjunct is considered is much more than that actually expressed by the formulae introduced by the rule. Indeed we already know that the left branch labelled with $\mathcal{S}, \mathcal{R}_A : A$ can be closed by a certain substitution σ_A. In many cases, the formula on the right $\mathcal{R}_A : \neg A\sigma_A$ will not be ground. With the current branching rule all remaining free variables will be treated rigidly. The key of the optimization is how can we transform (some of) them into universal variables.

We need a definition:

Definition 5 *Let $\mathcal{S} = \{\mathcal{R}_1{:}A_1, \ldots, \mathcal{R}_n{:}A_n\}$ be a sequent and σ be a substitution. The result of the application of σ to $\mathcal{S}$ is the sequent $\{FV(\mathcal{R}_1\sigma){:}A_1\sigma, \ldots, FV(\mathcal{R}_n\sigma){:}A_n\sigma\}$.*

It can be shown that if $\mathcal{R}_i$ contains the only free variables of A_i which are shared with some other A_j then one also has that $FV(\mathcal{R}_i\sigma)$ contains the only free variables of $A_i\sigma$ which are shared with another $A_j\sigma$.

Then we can optimize the symmetric branching rules of Fig. 2 as follows. Let $\mathcal{S}, \mathcal{R}{:}A \vee B$ be a sequent and let σ_A be a closing substitution for the proof search tree whose root is labelled by $\mathcal{S}, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}A$. Then the right successor of $\mathcal{S}, \mathcal{R}{:}A \vee B$ in the proof search tree can be $\mathcal{S}\sigma_A, FV(\mathcal{R}\sigma_A){:}B\sigma_A$. Notice that we use $FV(\mathcal{R}\sigma_A)$ and not $FV(\mathcal{R}_{A*B}\sigma_A)$.

The optimization can be better understood if we reason about the transformation of the *active frontier* of the tree constituted by the leaves of the proof search tree which are not (yet) closed by our procedure for constructing the search tree in depth-first. Assume that the frontier is the following:

$$\mathcal{S}_A, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}A \mid \mathcal{S}_B, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}B \mid \ldots \mid \mathcal{S}_n$$

Suppose also that σ_A is a closing substitution for the leftmost sequent $\mathcal{S}_A, \mathcal{R}{:}A \vee B, \mathcal{R}_{A*B}{:}A$. Then, the standard branching rule used in a depth first visit of the proof search tree would transform it into the following active frontier:

$$\mathcal{S}_B\sigma_A, FV(\mathcal{R}\sigma_A){:}(A \vee B)\sigma_A, FV(\mathcal{R}_{A*B}\sigma_A){:}B\sigma_A \mid \ldots \mid \mathcal{S}_n\sigma_A$$

Thus, the rigid variables of $B\sigma_A$ in the next sequent to be considered are those of $FV(\mathcal{R}_{A*B}\sigma_A)$ which means $FV(\mathcal{R}\sigma_A) \cup (FV(A\sigma_A) \cap FV(B\sigma_A))$.

In contrast, the optimized branching rule would yield the following frontier (again deleting closed branches):

$$\mathcal{S}_B\sigma_A, FV(\mathcal{R}\sigma_A){:}(A \vee B)\sigma_A, FV(\mathcal{R}\sigma_A){:}B\sigma_A \mid \ldots \mid \mathcal{S}_n\sigma_A$$

Thus, the rigid variables of $B\sigma_A$ in the next sequent considered by a DFS-algorithm for proof search are only $FV(\mathcal{R}\sigma_A)$, usually a subset of the previous case.

For the semantic branching rules of Fig. 2 we can label the right successor of $\mathcal{S}, \mathcal{R} : A \vee B$ in the proof search tree by $\mathcal{S}\sigma_A, FV(\mathcal{R}\sigma_A)\!:\!B\sigma_A, FV(\mathcal{R}\sigma_A)\!:\!\neg A\sigma_A$. Again we used $FV(\mathcal{R}\sigma_A)$ and neither $FV(\mathcal{R}_{A*B}\sigma_A)$ nor $FV(\mathcal{R}_A\sigma_A)$.

With semantic branching we do not need to visit the left branch containing $\mathcal{R}_A : A$ first. The search process could proceed through the right branch containing $\mathcal{R}_{A*B}\!:\!B$ and $\mathcal{R}_A\!:\!\neg A$ first. Indeed this could be the best choice since B has more universal variables. Then we can reformulate the branching rule as follows: let $\mathcal{S}, \mathcal{R}\!:\!A \vee B$ be a sequent and let $\sigma_{B,\neg A}$ be a closing substitution for the proof search tree whose root is labelled by $\mathcal{S}, \mathcal{R}\!:\!A \vee B, \mathcal{R}_{A*B}\!:\!B, \mathcal{R}_A\!:\!\neg A$. Then the right successor of $\mathcal{S}, \mathcal{R}\!:\!A \vee B$ in the proof search tree can be $\mathcal{S}\sigma_{B,\neg A}, FV(\mathcal{R}\sigma_{B,\neg A})\!:\!A\sigma_{B,\neg A}$.

6 An efficient yet lean implementation

The calculus with renaming, first order reduction and simplification can be easily implemented in sicstus prolog. We only sketch key issues of the prover Beatrix, which has been designed as a lean implementation of the calculus along the line of Beckert and Possega [7].

The first tricky bit is the implementation of renaming modulo rigid variables and of the "one-sided-restricted-unification" denoted by $A = B\sigma_{\overline{\mathcal{R}}}$ and used extensively in the first order reduction and simplification operations in Figure 4 and Figure 5.

In a lean prolog implementation we use prolog variables for first order variables and this makes it possible a simple implementation of the operation of renaming of a formula A modulo a set of rigid variables $\mathcal{R}$:

```
rename_fml(A,R,NewA):- copy_term(A:R,NewA:R).
```

Restricted unification $A = B\sigma_{\overline{\mathcal{R}}}$ are implemented using a predicate of the standard `terms` library of sicstus:

```
restricted-unify(A,B,R):- subsumes(B:R,A:R).
```

The first advantage of having universal variables is that we can give precedence to closing substitutions using only universal variables. Indeed, one of the difficulty faced by any depth-first-algorithm is the choice of a closing substitution when considering sequents one by one: for one sequent we may choose a "locally good but globally wrong" closing substitution that may affect negatively the proof search for another sequent. This happens because the closing substitution may bind the rigid variables shared between the two sequents to the "wrong" values. With closing substitutions with domains of only universal variables, "locally good" means "globally good": by construction that such substations do not affect other branches of the proof search tree. It is therefore important to search for such closing substitution first. This means that given a sequent we look for two formulae $\mathcal{R}_A : A$ and $\mathcal{R}_B : \neg B$ in the sequent such that $A\sigma_{\overline{\mathcal{R}_A}} = B\sigma_{\overline{\mathcal{R}_B}}$. This is implemented with the prolog rule below:

```
unifiable(A:RA,[B:RB|_],_,_):-
  copy_term([A:RA,B:RB],[AN:RAN,BN:RBN]),
  unify_with_occurs_check(AN,BN),
  subsumes_chk(RAN,RA),
  subsumes_chk(RBN,RB).
```

All predicates used in the body come with the standard sicstus distribution.

For the implementation of the main procedure, we search for a proof of a not empty sequent by applying simplification first[2], then first order reduction and finally selecting a formula for reducing it according the rules of the calculus. This can be implemented as

```
prove_seq([A:RA|Seq],Beta,Inst,Lits,Depth):-
  simplify_all(Lits,[A:RA|Seq],SSeq),
  reduce_seq(SSeq,[F:R|RSeq]),
  prove_fml(F:R,RSeq,Beta,Inst,Lits,Depth).
```

where `[A:RA|Seq]` is the set of unprocessed formulae in the sequent, `Beta` is the set of disjunction which have been duplicated in the course of the proof search by the branching rule, `Inst` is used to keep track of the different instances of the disjunctions in `Beta` which have been actually generated by various substitutions, `Lits` is the number of literals so far collected in the current sequent, and `Depth` is the maximum term depth allowed for closing substitutions which bind rigid variables. Instances and term depth are necessary for fairness and termination. As usual, completeness is achieved by iterative deepening over the term depth.

The last tricky bits are the branching rules:

```
prove_fml([or,A|B]:R,Seq,Beta,...):-
 shared_rigid_vars(A,[or|B],R,RA,RB),
 rename_fml(A,RA,NA),
 rename_fml([or|B],RB,[or|NB]),
 rename_fml([or,A|B],R,[or|D]),
 prove_seq([NA:RA|Seq],[[or|D]:R|Beta],...),
 normalise([[or|NB]:RB|Seq],NewSeq),
 normalise([[or|D]:R|Beta],NewBeta),
 ...
 prove_seq(NewSeq,NewBeta,...).
```

[2]The simplified formula might be arbitrary whereas we have restricted the simplifying formula to be a literal.

An intuitive explanation of this prolog clause is that the first predicate in the body constructs the set of rigid variables for the left disjunct A and the right disjunct A. They will be different whether we use semantic or symmetric branching, and then the optimized or the unoptimized version. Then we rename immediately the formula A and B and the disjunction $A \vee B$. Then we try to prove the left sequent. Since this operation might bind some rigid variables shared between the two sequents we need to Norma-Lise them before going on with the search on the right.

7 Correctness

For the soundness proof we cannot use the standard techniques, with a rigid assignment to all variables in a proof tree: renaming and the β-rule would be unsound. We must associate the frontier of a proof tree to a formula:

$$\begin{array}{rcl} fml(\mathcal{S}_1 \mid \ldots \mid \mathcal{S}_n) & = & fml(\mathcal{S}_1) \vee \ldots \vee fml(\mathcal{S}_n) \\ fml(\mathcal{S}) & = & \forall x_1 \ldots x_n . \bigwedge_i A_i \end{array}$$

where $x_k \in FV(\mathcal{S}) \setminus \bigcup_i \mathcal{R}_i$ and $\mathcal{R}_i{:}A_i \in \mathcal{S}$.

Then we must prove that, given the frontier of a proof tree $\mathcal{S}_1 \mid \ldots \mid \mathcal{S}_n$ and the new frontier due to some rule application $\mathcal{S}'_1 \mid \ldots \mid \mathcal{S}'_{n'}$, for all interpretations $\mathcal{I}$ and all assignments $\mathcal{A}$ if $\mathcal{I}, \mathcal{A} \models fml(\mathcal{S}_1 \mid \ldots \mid \mathcal{S}_n)$ then for all assignments $\mathcal{A}'$ extending $\mathcal{A}$ it is $\mathcal{I}, \mathcal{A}' \models fml(\mathcal{S}'_1 \mid \ldots \mid \mathcal{S}'_{n'})$.

The case for the *ren*-rule is then easy and the case for β-rule stems from the observation that $\forall x.(A \vee B)$ is equivalent to $(\forall x.A) \vee B$ if $x \notin FV(B)$. The soundness of the *simp*-rule and the reduction rules is similar: in the replacement steps we only unify universal variables.

The optimized version of either branching rules can be proven sound by a cut and paste argument over the unoptimized proof of a formula.

8 Comparison with related works

Already at the propositional level the calculus for simplification subsumes a number of related works. Here we just list some of them and refer to Massacci [23] for further analysis.

For instance the DPLL-unit rule [12, 11], the β_i^c rules for KE by D'Agostino and Mondadori [9], modus ponens, tollens etc. of HARP [25] and the $\vee - simp_{0,1}$ rules in KRIS [2] or the boolean constraint propagation operation in DLP [19] and KSAT [16] are all (restricted) instances of the simplification rule (*simp*) immediately followed by $(bool - red)$. Propositional hyper-tableaux [5] can be simulated by our rule.

The traditional way to lift the propositional framework to first order logic is to use the *ground version* of the calculus. We need a γ rule á la Smullyan [29]:

$$\frac{\mathcal{S}, \mathcal{R}{:}A(t)}{\mathcal{S}, \mathcal{R}{:}\forall x.A(x)} \quad \text{where } t \text{ is a ground term}$$

With this rule, no fresh variable is introduced in the proof search and hence we have no rigid variable. Simplification is simply propositional simplification and we can answer an interesting question: why first order DPLL is so inefficient in comparison with its propositional version?

At first we observe that the first order DPLL procedure is a calculus with off-line skolemisation, simplification, boolean reduction, semantic branching restricted to literals, and the ground γ-rule. Then, recall that using the γ-rule à la Smullyan rather than those with unification leads to a *non-elementary* slow down [3]. The exponential speed-up due to propositional simplification cannot compensate it.

The tableau calculus of Beckert and Possega [7] with universal variables is an instance of the calculus with renaming but without simplification where

1. in the β rule $\mathcal{R}_{A+B} \doteq \mathcal{R} \cup FV(A) \cup FV(B)$,
2. renaming is applied only to the literals in $\mathcal{S}$ before any β-rule.

This general technique also subsumes the use of conjunctive super-formulae by Degtyarev and Voronkov [13] which is based on restriction (1).

To ease the comparison, it is useful to consider the following fragment of a proof tree in which we apply a β rule and then immediately apply renaming to all formulae of both branches.

$$\dfrac{\dfrac{\dfrac{\mathcal{S}'', \mathcal{R}_{A*B}{:}B\eta_{\overline{\mathcal{R}_{A*B}}}}{\mathcal{S}'', \mathcal{R}_{A*B}{:}A}\ ren}{\mathcal{S}', \mathcal{R}_{A*B}{:}A}\ ren \qquad \mathcal{S}', \beta_2{:}\mathcal{R}_D}{\dfrac{\mathcal{S}', \mathcal{R}{:}A \vee B}{\mathcal{S}, \mathcal{R}{:}A \vee B}\ ren}\ \beta$$

where $\mathcal{S}' = \left\{\varphi_i \eta_{i\overline{\mathcal{R}}} \mid \varphi_i \in \mathcal{S}\right\}$ is obtained by a sequence of renaming steps and $\mathcal{S}''$ is obtained from $\mathcal{S}'$ in a similar way, using $\mathcal{R}_{A*B}$ rather than $\mathcal{R}$.

This proof fragment can be "approximated" with the following one:

$$\frac{\mathcal{S}\eta'_{\overline{\mathcal{R}}}, \mathcal{R}_{A*B}{:}A\eta_{\overline{\mathcal{R}_{A*B}}} \quad \mathcal{S}, \mathcal{R}_{A*B}{:}B}{\mathcal{S}, A \vee B{:}\mathcal{R}}\ \beta + ren$$

In practice, we apply one renaming to the whole sequent rather than renaming each formula. Notice that we apply $\eta'_{\overline{\mathcal{R}}}$ and not $\eta'_{\overline{\mathcal{R}_{A*B}}}$.

The net effect is to *separate* the universal variables of $A \vee B$ from those of $\mathcal{S}$ and the universal variables of the two branches. In this way we avoid the potential waste of an universal variable which may be present in formulae like $\forall x.P(x) \wedge (Q(x,y) \vee R(x,y))$. Clearly the x in $P(x)$ is universal and so is the the x in $Q(x,y) \vee R(x,y)$. However, without renaming, the application of the β-rule makes x a rigid variable. This is what happen if we use Beckert and Possega handling of universal variables which limits renaming to literals.

A comparison with the first order hyper-tableau expansion step by Baumgarten et al. [5, 4] is interesting: the idea of a purifying (grounding) substitution used in the extension step of hyper-tableaux can be casted in our framework with a particular search strategy and a general constraint on the final frontier of our proof tree: all rigid variables must be grounded by the closing substitution $\sigma_{\mathcal{T}}$.

For sake of example, assume that we have a binary clause $A \vee B$ and that we can apply to $A \vee B$ a substitution σ such that $FV(A\sigma)$ and $FV(B\sigma)$ are disjoint. Then the hyper-tableaux extension step would yield two branches one with $A\sigma$ and one with $B\sigma$, the instantiated formula $(A \vee B)\sigma$ would be discarded and we would only need to remember the initial clause $A \vee B$.

This is also the case for our branching rule. Indeed, we can strengthen proposition 1 as follows:

Proposition 2 *In the symmetric branching rules of Figure 2, the disjunction $\mathcal{R} : A \vee B$ can be deleted from both consequents if $FV(A) \cap FV(B) = \emptyset$.*

Non-clausal resolution by Manna and Waldinger [22] or the recent version by Björner et al. [8] shares some features of our proposal. Indeed, at propositional level unit non-clausal resolution is a particular instance of simplification. However, when performing a first-order non-clausal resolution step between two subformulae, the unifying substitution must be applied to the main formulae (tableau in the terminology of Manna and Waldinger). In contrast, our simplification rules allow for local substitutions and even different substitutions within the same formula.

The other close cousin of our approach is the FDPLL procedure developed by Baumgarten for the latest CASC competition. However, there are no simplification rules in his calculus which is limited to the CNF format and the usage of universal variables is more restricted.

9 Conclusion

In this paper we have presented a general framework for the introduction of full first order reduction rules for tableau and DPLL-like calculi which can make them fully competitive with resolution in the first order theorem proving setting.

The reduction rules which have been proposed play the same role of the unit rule for propositional DPLL, of subsumption inferences for resolution, and of a number of boolean and modal constraint propagation techniques applied in the description and modal logic communities. Our uniform framework thus subsumes a number of techniques for the improvement of tableau-based methods. Moreover, a prototype implementation is easily implemented using sicstus prolog, along the line of leanTAP [7].

Still, a doubt remains: the "beauty" of the tableau calculi and DPLL is their simplicity, so by adding these reduction rules for constraint propagation are not we abandoning the very calculus we try to enhance?

The author believes that the best answer is probably a question: is resolution with subsumption no longer resolution?

Acknowledgments

I would like to acknowledge many fruitful discussions with B. Beckert, L. Carlucci Aiello, F. Donini, U. Furbach, E. Giunchiglia, F. Giunchiglia, L. Paulson, and F. Pirri. I would like to thank P. Baumgarten for commenting on an earlier draft of this paper and for explaining to me the tricky bits of his calculus underlying FDPLL.

This work started while I was at the University of Cambridge (UK) and the Dipartimento of Informatica and Sistemistica of the University of Roma I "La Sapienza". It has been finalized thanks to a CNR short-term mobility grant at the University of Koblenz-Landau (D).

References

[1] Astrachan, O., and Stickel, M. Caching and lemmaizing in model elimination theorem provers. In *Proc. of the 11th International Conference on Automated Deduction (CADE-90)* (1992), vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 224–238.

[2] Baader, F., and Hollunder, B. A terminological knowledge representation system with

complete inference algorithms. In *Proc. of the International Workshop on Processing Declarative Knowldege (PDK-91)* (1991), H. Boley and M. Richter, Eds., vol. 567 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 67–86.

[3] Baaz, M., and Fermüller, C. G. Non-elementary speedups between different versions of tableaux. In *Proc. of the 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods* (1995), P. Baumgartner, R. Hähnle, and J. Possega, Eds., vol. 918 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 217–230.

[4] Baumgartner, P. Hyper tableau - the next generation. In *Proc. of the International Conference on Analytic Tableaux and Related Methods (TABLEAUX-98)* (1998), vol. 1397 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 60–76.

[5] Baumgartner, P., Furbach, U., and Niemelä, I. Hyper tableaux. In *Proc. of the 6th European Workshop on Logics in Artificial Intelligence (JELIA-96)* (1996), J. Alferes, L. Pereira, and E. Orlowska, Eds., vol. 1126 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 1–17.

[6] Beckert, B., and Hänle, R. An improved method for adding equality to free variable semantic tableaux. In *Proc. of the 11th International Conference on Automated Deduction (CADE-90)* (1992), D. Kapur, Ed., vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 507–521.

[7] Beckert, B., and Possega, J. `leanTAP`: Lean tableau-based deduction. *J. of Automated Reasoning 15*, 3 (1995), 339–358. A preliminary version appeared in *Proc. of the 12th International Conference on Automated Deduction (CADE-94)* (1994), A. Bundy, Ed., vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

[8] Björner, N., Stickel, M. E., and Uribe, T. E. A practical integration of first-order reasoning and decision procedures. In *Proc. of the 14th International Conference on Automated Deduction (CADE-97)* (1997), Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 101–115.

[9] D'Agostino, M., and Mondadori, M. The taming of the cut. *J. of Logic and Computation 4*, 3 (1994), 285–319.

[10] Davis, M. Eliminating the irrelevant from mechanical proofs. In *Proc. of the 15th Symposium in Applied Mathematics* (1963), N. Metropolis, A. H. Taub, J. Todd, and C. Tompkins, Eds., American Mathematical Society, pp. 15–30.

[11] Davis, M., Longemann, G., and Loveland, D. A machine program for theorem-proving. *Communications of the ACM 5*, 7 (1962), 394–397.

[12] Davis, M., and Putnam, H. A computing procedure for quantificational theory. *J. of the ACM 7*, 3 (1960), 201–215.

[13] Degtyarev, A., and Voronkov, A. Equality elimination for the tableau method. In *Proc. of the International Symposium on the Design and Implementation of Symbolic Computation Systems (DISCO-96)* (1996), J. Calmet and C. Limongelli, Eds., vol. 1128 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 46–60.

[14] Eisinger, N., Ohlbach, H. J., and Präcklein, A. Reduction rules for resolution-based systems. *Artificial Intelligence 50*, 2 (1991), 141–181.

[15] Fitting, M. *First Order Logic and Automated Theorem Proving*. Springer-Verlag, 1990.

[16] Giunchiglia, F., and Sebastiani, R. Building decision procedures for modal logics from propositional decision procedures - the case study of modal k. In *Proc. of the 13th International Conference on Automated Deduction (CADE-96)* (1996), M. A. McRobbie and J. K. Slaney, Eds., vol. 1104 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 583–597.

[17] Hähnle, R., and Schmitt, P. The liberalized δ-rule in free variable semantic tableaux. *J. of Automated Reasoning 13*, 2 (1994), 211–222.

[18] Horrocks, I. Using an expressive description logic: FaCT or finction? In *Proc. of the 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)* (1998), Morgan Kaufmann, Los Altos, pp. 636–647.

[19] Horrocks, I., and Patel-Schneider, P. F. Optimising propositional modal satisfiability for description logic subsumption. In *Proc. of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)* (1998), vol. 1476 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 234–246.

[20] Hustadt, U., and Schmidt, R. A. On evaluating decision procedure for modal logic. In *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)* (1997), M. Pollack, Ed., pp. 202–207.

[21] Letz, R., Mayr, K., and Goller, C. Controlled integration of the cut rule into connection tableau calculi. *J. of Automated Reasoning 13*, 3 (1994), 297–337.

[22] Manna, Z., and Waldinger, R. Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering 18*, 8 (1992), 674–704.

[23] Massacci, F. Simplification: A general constraint propagation technique for propositional and modal tableaux. In *Proc. of the International Conference on Analytic Tableaux and Related Methods (TABLEAUX-98)* (1998), H. de Swart, Ed., vol. 1397 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 217–231. An extended version is available as Technical Report 424, Computer Laboratory, University of Cambridge (UK), 1997.

[24] Massacci, F. Design and results of the tableaux-99 non-classical (modal) systems comparison. In *Proc. of the International Conference on Analytic Tableaux and Related Methods (TABLEAUX-99)* (1999), N. Murray, Ed., vol. 1617 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 14–18.

[25] Oppacher, F., and Suen, E. HARP: a tableau-based theorem prover. *J. of Automated Reasoning 4* (1988), 69–100.

[26] Prawitz, D. An improved proof procedure. *Theoria 26* (1960), 102–139.

[27] Robinson, J. A. A machine oriented logic based on the resolution principle. *J. of the ACM 12*, 1 (1965), 23–41.

[28] Shanin, N., Davydov, G., Yu., M. S., Mints, G. E., Orenkov, V. P., and Slisenko, A. O. An algorithm for machine search of a natural logical deduction in a propositional calculus. In *Automation of Reasoning (Classical Papers on Computational Logic. Vol. 1 (1957-1966)*, J. Siekmann and G. Wrightson, Eds. Springer-Verlag, 1983.

[29] Smullyan, R. M. *First Order Logic*. Springer-Verlag, 1968. Republished by Dover, New York, in 1995.

[30] Voronkov, A. The anatomy of vampire (implementing bottom-up procedures with code trees). *J. of Automated Reasoning 15*, 2 (1995), 237–265.

[31] Voronkov, A. Strategies in rigid-variable methods. In *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)* (1997), pp. 114–121.

[32] Wallace, K., and Wrightson, G. Regressive merging in model elimination tableau-based theorem provers. *J. of the Interest Group in Pure and Applied Logic 3*, 6 (1995), 921–937.

[33] Zhang, H., and Stickel, M. E. An efficient algorithm for unit-propagation. In *Proc. of the 4th International Symposium on Artificial Intelligence and Mathematics* (1996).