Simplification Rules for Constrained Formula Tableaux

Martin Giese

Chalmers University of Technology
Department of Computing Science
S-41296 Gothenburg, Sweden
giese@ira.uka.de

Abstract. Several variants of a first-order simplification rule for non-normal form tableaux using syntactic constraints are presented. These can be used as a framework for porting methods like unit resolution or hyper tableaux to non-normal form free variable tableaux.

1 Introduction

Non-clausal form analytic tableaux have a number of advantages over the proof procedures for clausal form implemented in most successful automated theorem provers. For instance, when the logic is enhanced by modal operators, clausal form cannot be used without previously translating the problems into first-order. Another case is the integration of automated and interactive theorem proving [1,8], where normal forms would be counter-intuitive. Unfortunately, standard non-normal form tableaux tend to be rather inefficient, as many of the refinements available to clausal procedures are hard to adapt. Typical cases in point are unit resolution, especially for propositional provers like the Davis-Putnam-Logemann-Loveland (DPLL) procedure [7], the β^c rules of the KE calculus [6], the application of 'result substitutions' in Stålmarcks Procedure [17], and hyper tableaux [2]. The common feature of these techniques is that they involve inferences between several formulae derived from the formula to be proved, either by using one formula to simplify another one, or—for hyper tableau—making tableau expansions depend on the presence of certain literals on a branch.

In [14], Massacci presents a simplification rule for propositional and modal tableau calculi. This rule is of the form

$$\begin{array}{c} \psi \\ \phi \end{array} \quad \overset{simp}{\rightsquigarrow} \quad \begin{array}{c} \psi[\phi] \\ \phi \end{array}$$

where $\psi[\phi]$ is the formula that results from first replacing all occurrences of ϕ in ψ by *true*, and if $\phi = \neg\phi'$, all occurrences of ϕ' by *false*, and then applying a set of boolean simplifications of the form

$$\neg \mathit{true} \to \mathit{false}, \quad \neg \mathit{false} \to \mathit{true}, \quad \mathit{true} \wedge \phi \to \phi, \quad \mathit{false} \wedge \phi \to \mathit{false}, \quad \text{etc.}$$

to eliminate all occurrences of truth constants. Massacci shows that proof procedures using this rule can subsume a number of other theorem proving techniques for propositional logic, e.g. the unit rule of DPLL [7], the β^c rules of KE [6], the regularity restriction, and hyper-tableaux [2]. This is done mainly by specifying the strategy of when and where to apply the *simp* rule.

While DPLL and hyper-tableaux are originally formulated for problems in clause normal form (CNF), the simplification rule is applicable to arbitrary predicate logic formulae, making it a good framework to generalize CNF techniques to the non-normal form case. Massacci gives variants of the simplification rule for various modal logics. In [13], he also gives a variant of the rule for first-order free-variable tableaux. Unfortunately, this rule does *not* in general subsume first-order versions of unit-resolution, hyper-tableaux etc., because it places strong restrictions on the instantiation of free variables.

This paper presents variants of the simplification rule for first order logic which overcome this limitation. The rules were first introduced in [9], but the proofs of most of the theorems were only sketched there.

2 Simplification with Global Instantiation

Consider a free-variable tableau branch with the formulae $p(X) \vee q(X)$ and $\neg p(a)$, where X is a free variable. If X were instantiated with a, the disjunction could be simplified to $q(a)$. Our task is to find a version of this ground simplification that works with free variables. The step from a ground version to a free variable version of a rule or a proof is usually referred to as *lifting*.

One possibility for lifting the simplification rule consists in applying a substitution to the whole proof, that unifies certain subformulae, so that a simplification becomes possible.

Such a rule would be formulated using the most general unifier (mgu) of the simplifying formula and some subformula of the simplified formula. A little care must be taken to prevent the instantiation of bound variables by such a unifier. We call (an occurrence of) a subformula ξ of ϕ *simplifiable*, if no variable occurring free in ξ is bound by ϕ. So $p(x)$ is simplifiable in $\exists y.(q(y) \wedge p(x))$, but not in $\exists x.(p(y) \wedge p(x))$. It is also simplifiable in $(\exists x.q(x)) \wedge p(x)$, because the quantifier does not bind the x occurring free in $p(x)$.

Using this notion, a simplification rule with global instantiation can be given:[1]

$$\begin{array}{cc} \psi & \\ \phi & \end{array} \quad \overset{simp}{\rightsquigarrow} \quad \begin{array}{c} \mu(\psi)[\mu(\phi)] \\ \mu(\phi) \end{array}$$

where μ is a mgu of ϕ and some simplifiable subformula of ψ,
and μ is applied on all open branches.

[1] We use a non-standard notation for tableau rules: the formulae on the left are required to be on the branch and are *replaced* by the ones on the right. This notation has the advantage of making clear which formulae need to be retained after the rule application.

While this approach is sound, it relies on the application of a global instantiation for the free variables. The problem with such a rule is that it introduces a new backtracking point, because the applied unifier might not lead to a proof. Not only does this make the rule unsuitable for a backtracking free proof procedure. It is also problematic for a backtracking prover as efficiency will suffer if more backtracking points than necessary are introduced.

3 Lifting with Constrained Formulae

A universal technique for avoiding the global application of substitutions is to decorate the formulae on tableau branches with unification constraints. A unification constraint C is a conjunction of syntactic equalities between terms or formulae, written as

$$s_1 \equiv t_1 \,\&\, \ldots \,\&\, s_k \equiv t_k \quad .$$

We use the symbol $\equiv$ for syntactic equality in the constraint language to avoid confusion with the meta-level $=$. Let

$$\mathrm{Sat}(C) = \{\sigma \mid \text{for all } i,\ \sigma(s_i) \text{ equals } \sigma(t_i) \text{ syntactically}\}$$

be the set of ground substitutions satisfying a constraint. A constraint is called satisfiable, if $\mathrm{Sat}(C)$ is not empty, which means that there is a simultaneous unifier for the pairs $\{s_i, t_i\}$. A constraint C subsumes a constraint D, iff $\mathrm{Sat}(D) \subseteq \mathrm{Sat}(C)$. Two constraints C and D are equivalent, iff $\mathrm{Sat}(C) = \mathrm{Sat}(D)$.

A *constrained formula* is an ordered pair $\phi \ll C$ of a formula ϕ and a constraint C. The intuition is to consider the formula ϕ as present, only if the free variables are instantiated in a way that satisfies the constraint. The empty constraint, which is satisfied by all ground substitutions, is usually omitted. Instead of globally applying a mgu of two formulae ϕ and ψ to the proof, when a rule application requires some instantiation of free variables, we can annotate the formulae resulting from the rule application with a constraint $\phi \equiv \psi$, which is a local operation that does not lead to a backtracking choicepoint. For instance, simplification of $p(X) \vee q(X)$ with $\neg p(a)$ requires instantiation of X with a leading to $\mathit{false} \vee q(a) \ll X \equiv a$, which is rewritten to $q(a) \ll X \equiv a$.

Obviously, if formulae $\phi_i \ll C_i$ which already carry constraints are involved in a rule application, the conjunction $C_0 \,\&\, C_1 \ldots$ has to be passed on to the resulting formulae. This is referred to as constraint *propagation*. Constraints are propagated through rule applications, until a branch is closed. Closure between two literals $L \ll C$ and $\neg L' \ll C'$ is only allowed if the constraint $C \& C' \& L \equiv L'$ is satisfiable.

Using unification constraints, the simplification rule takes the form

$$\begin{array}{ll} \psi \ll C & \qquad \psi \ll C \\ \phi \ll D \quad \overset{simp^{c0}}{\rightsquigarrow} & \qquad \phi \ll D \\ & \qquad \mu(\psi)[\mu(\phi)] \ll (C \,\&\, D \,\&\, \phi \equiv \xi) \end{array}$$

where ξ is a simplifiable subformula of ψ,
μ is a mgu of ξ and ϕ,
and $C \,\&\, D \,\&\, \phi \equiv \xi$ is satisfiable

The $simp^{c0}$ rule keeps an unsimplified copy of $\psi \ll C$ on the branch. This will change in later versions. An immediate consequence of keeping both original formulae is that completeness follows trivially from the completeness of the calculus without simplification. We only need to ascertain soundness.

Theorem 1. *The tableau calculus with constrained formulae using the $simp^{c0}$ rule is sound, i.e. if a proof exists for a finite set of formulae, then that set is unsatisfiable.*

Proof. Let σ be a closing substitution for the tableau. This means that σ assigns a ground term to each free variable that occurs on the tableau, so that under σ, every branch contains a complementary pair of literals with constraints satisfied by σ. Consider the ground proof-tree obtained by replacing each formula ϕ on the tableau by $\sigma(\phi)$. In particular, this implies omitting any formulae with constraints that are not satisfied by σ. Tableau expansions for formulae with unsatisfied constraints are left out. For a β-expansion this means that only one of the branches needs to be kept, it doesn't matter which.

There is then a complementary pair on each branch of the resulting ground proof. Furthermore, as constraints can only be strengthened by rule applications, all proof steps needed to derive the complementary pair are still present in the reduced proof. Simplification steps are transformed into instances of the ground simplification rule

$$\begin{array}{cc} \psi & \\ \phi & \rightsquigarrow \end{array} \begin{array}{c} \psi \\ \phi \\ \psi[\phi] \end{array}$$

It remains to show that this ground version of the rule is sound. For this, it is sufficient to show that in any model where ψ and ϕ hold, $\psi[\phi]$ also holds, which immediately follows from the definition of $\psi[\phi]$. □

It is a little misleading to call $simp^{c0}$ a simplification rule, because the original formula $\psi \ll C$ has to be retained for completeness. Indeed, one cannot simply delete the original formula, because there is no guarantee that the closing instantiation of the proof will be such that the simplification is possible.

There is however an important special case: if the 'new' part of the constraint $D \,\&\, \phi \equiv \xi$ subsumes the 'old' part C, the original formula $\psi \ll C$ may be discarded, because this means that the simplification step is valid in all ground instances of ψ allowed by the constraint C. Let $simp^{c1}$ be the rule obtained with this modification.

Theorem 2. *The $simp^{c1}$ rule is sound. It is also complete, in the sense that a branch that can be closed under some σ after applying a sequence R of expansion steps, can still be closed under σ by a modified sequence R' after an application of the $simp^{c1}$ rule. Moreover, there is such an R' that is at most as long as the original R.*[2]

[2] Note that the $simp^{c1}$ application is not counted in R'. So the overall proof size may increase by 1.

Proof. Soundness may be shown as for $simp^{c0}$, see Theorem 1.

Completeness would be difficult to show by a Hintikka-style argument, because of the destructive nature of $simp^{c1}$. Apart from that, such a proof would not yield the statement about the proof sizes. We shall construct R' from R by a proof transformation, in which rule applications on (descendants of) a discarded original formula $\psi \ll C$ can either be applied to (descendants of) the simplified or simplifying formula, or be discarded altogether.

In case the $simp^{c1}$ application does not discard the original formula, we can simply take $R' = R$. Assume that the original formula ψ *is* discarded. In that case the new constraint $C \,\&\, D \,\&\, \phi \equiv \xi$ is equivalent to C. We also assume that the closing substitution σ satisfies C, because otherwise, the simplification step could not be useful to close the branch, and R' could be constructed by simply leaving it away. In particular, we thus have $\sigma \in \mathrm{Sat}(D)$ and $\sigma(\phi) = \sigma(\xi)$. We now 'factor' the replacement of ψ by $\mu(\psi)[\mu(\phi)]$ into a sequence of simpler replacements and show for each of these how R is transformed.

Firstly, as σ satisfies C, $\sigma(\mu(\psi)) = \sigma(\psi)$. This implies that ψ can be replaced by $\mu(\psi)$ in the original branch, and the derivation R still closes it under σ.

After this, the calculation $\mu(\psi)[\mu(\phi)]$ from $\mu(\psi)$ consists in replacing occurrences of a sub-formula of $\mu(\psi)$ by *true* or *false*, and performing a number of boolean simplifications in the result.

With the formula $\phi \ll D$ on the branch, let us replace an occurrence of $\mu(\phi)$ in $\mu(\psi)$ by *true*. Let R' mimic all the proof steps in R except those which concern the sub-formula which has been replaced by *true*. If the replaced occurrence in ψ has positive polarity, i.e. it is in the scope of an even number of negations, then R produces the formula $\mu(\phi) \ll C$, while R' produces $\mathit{true} \ll C$. The proof steps of R on $\mu(\phi) \ll C$ are transformed to R' by applying them on the formula $\phi \ll D$, which is also on the branch. The applicability of tableau rules depends only on the top level junctors and quantifiers, so all rules that are applied on $\mu(\phi)$ can also be applied on ϕ. The constraint D on ϕ is also no problem, as the closing substitution σ satisfies D. If the occurrence is of negative polarity, R' produces the formula $\mathit{false} \ll C$, which immediately closes the branch under σ. The dual case, where $\neg\phi \ll D$ is on the branch, and a sub-formula is replaced by *false* is exactly symmetric.

It remains to show how that boolean simplification steps don't affect completeness. We will look only at some representative cases. Assume that an occurrence of $A \wedge \mathit{true}$ is replaced by A in ψ. Again, we let R' mimic the original proof steps until a rule must be applied on the simplified sub-formula. Depending on polarity, we now have only A instead of $A \wedge \mathit{true}$, resp. $\neg A$ instead of $\neg(A \wedge \mathit{true})$. In the first case, an α rule application in R only leads to an additional literal *true*, which is useless for the proof, so all later proof steps in R can be applied in R'. In the second case, the β rule application in R leads to one branch with $\neg A$ and one with $\neg\mathit{true}$. We can use the proof steps of the former to finish R'.

We now consider the case where an occurrence of $A \wedge \mathit{false}$ is replaced by *false* in ψ. Again depending on polarity, the proof steps of R now produce *false* instead of $A \wedge \mathit{false}$, resp. *true* instead of $\neg(A \wedge \mathit{false})$. In the first case, the *false*

literal can be used to close the derivation R' immediately. In the second case, the β split in R produces one branch with $\neg A$ and one with $\neg \mathit{false}$. As the latter formula cannot be used to close the branch, we can take the rule applications of R on that branch to complete R'.

Other boolean simplification steps for quantifiers and negation can be handled similarly. □

The simp^c rules enjoy an interesting relative termination property, which it shares with the α and β rules, namely that only a finite number of simplification steps can be performed without intervening γ-expansions, under certain side conditions. Call two constrained formulae $\phi \ll C$, $\psi \ll D$ *variants*, if C and D are equivalent and for all $\sigma \in \mathrm{Sat}(C)$, $\sigma(\phi) = \sigma(\psi)$. E.g. $p(X) \ll X \equiv a$ and $p(a) \ll X \equiv a$ are variants.

Theorem 3. *Starting from a given tableau branch, only a finite number of α, β, and simp^{c0} rule applications without intervening applications of the γ rule are possible, if simp^{c0} is never applied twice to the same pair of constrained formulae, and any formula which is a variant of a formula already present on a branch is discarded. The same is true for the simp^{c1} rule.*

Proof. A formula ϕ can only be simplified by setting one of its subterms to *true* or *false*, and the resulting simplified formulae are all smaller than ϕ. So the number of distinct formulae that can be generated is finite. On the other hand, all constraints that could be generated are conjunctive combinations of existing constraints and syntactic equations between subformulae of formulae on a branch, so there can be only finitely many non-equivalent constraints. Accordingly, the number of non-variant constrained formulae must be finite. For simp^{c1}, formulae are occasionally discarded from a branch. This implies that even less rule applications are possible, so the same argument holds. □

As a practical consequence of this finiteness property, there is no need to interleave γ and simp^c applications in a proof procedure to guarantee fairness. It is possible to apply all possible simplifications before considering an application of the γ rule.

4 Dis-Unification constraints

Although the simp^{c1} rule permits the original formula $\psi \ll C$ to be deleted from the branch in some cases, it will usually have to be kept. This can lead to redundancies as exemplified by the following tableau branch for the set of formulae $\{p(a), q(a), \neg p(X) \vee \neg q(X) \vee r(X)\}$:

$$
\begin{gathered}
1 : p(a) \\
2 : q(a) \\
3 : \neg p(X) \vee \neg q(X) \vee r(X) \\
4 = \mathit{simp}(3,1) : \neg q(a) \vee r(a) \ll X \equiv a \\
5 = \mathit{simp}(3,2) : \neg p(a) \vee r(a) \ll X \equiv a
\end{gathered}
$$

After generation of 4, formula 5 is redundant, because if X is actually instantiated with a as the constraint of 5 demands, formula 3 could have been discarded after generating 4. $q(a)$ only needs to be used to simplify 4, leading to $r(a) \ll X \equiv a$. In the presence of a large formula and many simplifying literals, a large number of such redundant formulae may be generated.

One way of overcoming this problem is to record instantiations under which a formula could have been discarded in the constraint. To do this, we have to require the constraint language to be closed under negation (denoted '!') as well as conjunction. The resulting constraint satisfiability problems are known as *dis-unification* problems, see e.g. [5], so I will talk of dis-unification or DU constraints.

A little care has to be taken with the semantics of DU constraints: Some DU constraints that are not satisfiable in the current signature might become satisfiable when the signature is extended. E.g., $!\,X \equiv a$, is not satisfiable in a signature consisting only of the constant symbol a, but it is satisfiable in any extended signature. In our context, satisfiability should be considered with respect to a possibly extended signature, because new skolem symbols might be introduced at a later point. In practice, it turns out that the satisfiability and subsumption checks actually get simpler with this definition. The same effect for term ordering constraints was noted in [15].

Using DU constraints, the simplification rule can be reformulated as follows:

$$\begin{array}{c} \psi \ll C \\ \phi \ll D \end{array} \quad \overset{simp^{c2}}{\rightsquigarrow} \quad \begin{array}{c} \psi \ll C \,\&\, !(D \,\&\, \phi \equiv \xi) \\ \phi \ll D \\ \mu(\psi)[\mu(\phi)] \ll (C \,\&\, D \,\&\, \phi \equiv \xi) \end{array}$$

where ξ is a simplifiable subformula of ψ,
μ is a mgu of ξ and ϕ,
and $C \,\&\, D \,\&\, \phi \equiv \xi$ is satisfiable

This rule differs from $simp^{c0}$ in that the constraint of the original formulae ψ is changed by adding $!(D \,\&\, \phi \equiv \xi)$. What this means is that the formula is no longer available for simplification steps requiring an instantiation under which *this* simplification would have been possible.

We now allow formulae with unsatisfiable constraints to be discarded, as they cannot contribute to tableau closure anyway. One easily checks, that this makes it possible to discard ψ at least in all those cases, where $simp^{c1}$ allows it.

The example above now becomes

$$\begin{array}{c} 1 : p(a) \\ 2 : q(a) \\ 3 : \neg p(X) \vee \neg q(X) \vee r(X) \end{array} \quad \rightsquigarrow \quad \begin{array}{c} 1 : p(a) \\ 2 : q(a) \\ 3 : \neg p(X) \vee \neg q(X) \vee r(X) \ll !\,X \equiv a \\ 4 : \neg q(a) \vee r(a) \ll X \equiv a \end{array}$$

The constraint $!\,X \equiv a$ now prevents the simplification of 3 with 2. But we can perform a second simplification step by simplifying 4 with 2, which changes the constraint of 4 to $X \equiv a \,\&\, !\,X \equiv a$, which is unsatisfiable, so 4 can be discarded after adding the literal $r(a) \ll X \equiv a$.

The $simp^{c2}$ rule enjoys similar properties as $simp^{c1}$, as the following theorem shows.

Theorem 4. *The $simp^{c2}$ rule is sound. It is also complete in the sense of Theorem 2.*

Proof. Soundness follows from Theorem 1, as strictly less rule applications are possible than with $simp^{c0}$. Completeness is shown using the same technique as for Theorem 2. We take the addition of the DU constraint into account by considering three cases, depending on which of the constraints involved in the $simp^{c2}$ application are satisfied by the closing substitution σ. If σ does not satisfy C, any proof steps on $\psi \ll C$ can be left out anyway, as they do not contribute to the closure of the subtableau. If $\sigma \in \mathrm{Sat}(C)$, but $\sigma \notin \mathrm{Sat}(D \,\&\, \phi \equiv \xi)$, we can perform all extensions as in R, because σ satisfies the changed constraint $C \,\&\, !(D \,\&\, \phi \equiv \xi)$. Finally, if $\sigma \in \mathrm{Sat}(C)$ and $\sigma \in \mathrm{Sat}(D \,\&\, \phi \equiv \xi)$, we perform the proof transformation as in the proof of Theorem 2, considering the original formula to be deleted, because its constraint and the constraints of any formulae derived from it is not satisfied by σ. □

The principal drawback of the $simp^{c2}$ version of our simplification rule is the high complexity of dis-unification. As a compromise, it is possible to keep unification (U) and dis-unification (DU) parts of constraints separate and to weaken the DU part of constraints if convenient. The unification part has to be left alone, as it is relevant for soundness. The DU part only serves to reduce the necessary proof search, so it may be thrown away without losing correctness.

One possible approach consists in restricting oneself to *conjunctive dis-unification constraints* [11], which are constraints of the form $C_0 \,\&\, !\,C_1 \,\&\, !\,C_2 \ldots$, where the C_i are conjunctive unification constraints as in Sec. 3. Here, C_0 is the U part and $!\,C_1 \,\&\, !\,C_2 \ldots$ the DU part of the constraint. The DU part of the constraint of a formula is discarded before it is used to simplify another one, in order to maintain this form for all constraints. Satisfiability and subsumption (for possibly extended signatures) are fairly easy to check for these constraints. In fact, it is shown in [11], that the conjunctive DU-constraint is satisfiable in a possibly extended signature, exactly if C_0 is satisfiable and C_0 is not subsumed by any of the C_i.

5 Using Universal Variables

In practice, the simplification rules as outlined above tend to require a lot of instances of γ-formulae. E.g., given the formulae $\{p(a), p(b), p(c), \forall x.\neg p(x) \vee q(x)\}$, one can produce after one γ expansion the literals $q(a) \ll X \equiv a$, $q(b) \ll X \equiv b$, and $q(c) \ll X \equiv c$. But these literals have mutually contradictory constraints, so any further rule application or closure can involve at most one of these literals. One needs three instances of the γ formula to produce the compatible literals $q(a) \ll X_1 \equiv a$, $q(b) \ll X_2 \equiv b$, and $q(c) \ll X_3 \equiv c$. But with three instances, not only these three useful literals are deducible, but a total of nine

q-literals coming from the simplification of each instance $\neg p(X_i) \vee q(X_i)$ with each of the three p-literals. As all of these will subsequently be used to simplify any q-subformula on the branch, this can quickly lead to a huge (though finite) number of rule applications.

One way to reduce the number of distinct instances of γ formulae is to use universal variables, see e.g. [4]. A free variable x is called *universal* with respect to a formula ϕ on a tableau branch, if $\forall x.\phi$ is a logical consequence of the formulae on a branch. All other free variables are called *rigid*. This property is of course undecidable. In practice, one uses simple sufficient criteria to detect universality of free variables, the most common one being to flag all free variables introduced in a γ extension as universal, and to preserve universality through all non-splitting rule applications. After a β rule application, those free variables which occur on more than one of the subformulae become rigid. The benefit of universal variables is that they may be instantiated independently for all formulae and may also be renamed as needed, whereas rigid variables have to be instantiated identically on all branches.

I shall write $[\bar{X}]\phi \ll C$ for a constrained formula with universal variables $\bar{X}$. Using universal variables, the following derivation is possible:

$$\begin{array}{c} p(a) \\ p(b) \\ p(c) \\ \forall x.\neg p(x) \vee q(x) \\ [X]\neg p(X) \vee q(X) \end{array} \quad \overset{3\times simp}{\rightsquigarrow} \quad \begin{array}{c} p(a) \\ p(b) \\ p(c) \\ \forall x.\neg p(x) \vee q(x) \\ [X]\neg p(X) \vee q(X) \ll !\, X \equiv a \,\&\, !\, X \equiv b \,\&\, !\, X \equiv c \\ [X]q(a) \ll X \equiv a \\ [X]q(b) \ll X \equiv b \\ [X]q(c) \ll X \equiv c \end{array}$$

The resulting literals are no longer incompatible, because X may be instantiated differently for each of them. It is of course possible to eliminate the universal variable and constraint altogether in these literals, but that is a technical optimization which is not strictly necessary.

Formally, in a simplification, all free variables in the result that were universal in one of the original formulae may be flagged as universal in the result [11]:

$$\begin{array}{c} [\bar{X}]\psi \ll C \\ [\bar{Y}]\phi \ll D \end{array} \quad \overset{simp^{c2u}}{\rightsquigarrow} \quad \begin{array}{c} [\bar{X}]\psi \ll C \,\&\, !(D \,\&\, \phi \equiv \xi) \\ [\bar{Y}]\phi \ll D \\ [\bar{X} \cup \bar{Y}]\mu(\psi)[\mu(\phi)] \ll (C \,\&\, D \,\&\, \phi \equiv \xi) \end{array}$$

where ξ is a simplifiable [3] subformula of ψ,
μ is a mgu of ξ and ϕ,
and $C \,\&\, D \,\&\, \phi \equiv \xi$ is satisfiable

[3] The 'simplifiable subformula' condition could be relaxed to permit, e.g. the simplification of $\exists y.p(y)$ with $[X].p(X)$, but this becomes rather technical, so we won't do it in this paper.

This rule is sound and complete for the free-variable tableau calculus with universal variables. Completeness can be shown by a combination of the technique used in [8], Sect. 7.4, for showing completeness of tableaux with universal variables, and the proof transformation technique of Theorems 2 and 4. By contrast, the termination property does not hold anymore, if universal variables are used. To apply the $simp^{c2u}$ rule, it is necessary in general to rename universal variables in the original formulae to make them disjoint. But this renaming destroys termination. Consider for instance the formulae $p(a)$ and $[X]\neg p(X) \vee p(f(X))$. With simplification and renaming of universal variables, it is possible to consecutively deduce

$$[X_1]p(f(a)) \ll X_1 \equiv a$$
$$[X_1, X_2]p(f(f(a))) \ll X_1 \equiv a \,\&\, X_2 \equiv f(a)$$
$$[X_1, X_2, X_3]p(f(f(f(a)))) \ll X_1 \equiv a \,\&\, X_2 \equiv f(a) \,\&\, X_3 \equiv f(f(a))$$
$$\text{etc.}$$

This means, that in general simplification and γ instantiation need to be interleaved to retain fairness. As the $simp^{c2u}$ rule *without* renaming obviously enjoys the finiteness property, one might alternatively interleave renaming and γ instantiation, but that would amount to ignoring universality for most of the time.

It is interesting to note that there are many problems, Schubert's 'Steamroller' [18] being a particularly prominent example, in which simplification with universal variables actually *does* terminate. This is true, in particular, when some *simplification strategy*, like the hyper strategy discussed in the next section is used, which does not apply arbitrary simplification steps. To handle such cases efficiently, it is advisable to equip a proof procedure with some sort of cycle detection that only interleaves simplifier applications with γ rules, if they threaten to lead to infinite simplification sequences. One possibility is to set a limit to the size of inferred formulae, which can be incrementally increased as the tableau is expanded. This would always allow rule applications which really simplify a formula in the sense of making it smaller.

6 Simplification Strategies

Although we have identified cases in which we can discard the original formula in a simplification step, we should not forget that this is not possible in general. With the $simp^{c2}$ and $simp^{c2u}$ rules, we can at least strengthen the constraint of the original formula, but this does not change the fact that our so-called simplification rule actually makes branches larger in most cases. The reason of using the name simplification is the analogy to the ground and propositional simplification rules which our first-order version subsumes.

In order for the simplification rules to be useful in a prover, one needs a *simplification strategy*, that is a strategy that prescribes when to apply which kinds of simplification steps.

We claimed in the introduction that our simplification rules are capable of simulating first-order versions of various refinements, including hyper tableaux, and regularity. We have yet to show that this has been achieved. In this section, we shall describe a simplification strategy that implements a non-clausal analogue of hyper tableaux. The details of this strategy and corresponding proofs can be found in [11]. In that work, there is also a description of how the simplification rules may be used to introduce a first order, non-clausal version of regularity.

Hyper tableaux are defined for problems stated in clause normal form (CNF), see [12,2]. For clause tableaux, it is customary not to include the clauses in the tableau itself. Instead, one only uses the literals which result from expanding the tableau with a clause. Hyper tableaux permit an expansion with a clause only if all new branches which receive negative[4] literals of the clause are immediately closed. All inner leaves are thus positive literals.

Alternatively, one can take the view of interpreting the clauses as tableau expansion rules themselves. In this view, a clause is 'fired' if there is a positive literal on a branch for every negative literal of the clause. The tableau is then extended by one new branch for each of the positive literals of the clause. One usually writes clauses as implications to support this view.

In the first-order case, one has to apply a substitution to unify the negative literals of the clause with corresponding positive literals on the branch. The way variables are handled differs between the various presentations of hyper tableaux. While [2] uses universal variables in branch literals where possible, that version of hyper tableaux does not use rigid variables. Instead, it uses 'purifying substitutions' which generate ground instances of clauses if necessary. This happens whenever a variable is shared between two positive literals of a clause without occurring in any of the negative literals. A version described in [12] uses rigid variables in such situations, using copies of clauses to avoid destructive instantiation. In [20], a variant with rigid variables and constraints is proposed, but constraints are attached to branches instead of formulae as is done in our calculi.

We can define a version of first-order hyper tableaux using constrained formulae. As usual, we use rigid variables when necessary, and constraints to capture necessary instantiations. Here is an example of this approach:

$$\begin{array}{c} \text{Clause Set:} \\ p(a,b) \\ p(x,z) \rightarrow p(x,y) \vee p(y,z) \\ p(x,f(x)) \rightarrow q(x) \end{array} \qquad \begin{array}{ccc} & \emptyset & \\ & p(a,b) & \\ p(a,Y) & & p(Y,b) \\ q(a) \ll Y \equiv f(a) & & \end{array}$$

After putting the literal $p(a,b)$ on the branch using the first clause, we expand the tableau using the second clause, where $p(x,z)$ is instantiated with $p(a,b)$. As the

[4] We consider *positive* hyper tableaux here. It is possible to exchange the roles of positive and negative literals, which leads to negative hyper tableaux.

two branches share the new variable Y, this has to be rigid. Subsequent expansion of the left branch with the third clause is possible only if Y is instantiated to $f(a)$. This restriction is captured in the constraint of the generated literal.

These rigid variable, constrained formula hyper tableaux can be emulated using our simplification rule with universal variables and a suitable simplification strategy. From now on, we shall consider normal analytic tableaux again. The set of clauses is given as a set of universally quantified disjunctions of literals. The following simplification strategy then emulates hyper tableaux:

> Use simplification only to simplify any *leftmost* negative literals inside disjunctions with positive literals occurring on the branch. Use β-expansion only for disjunctions which contain no negative literals.

With this strategy, the emulation of a hyper tableau expansion will require exactly one intermediate simplification step for every negative literal in the clause/disjunction in question.

There is obviously not much merit in using this emulation of hyper tableaux in an actual implementation, if problems are given as clause sets. It would be simpler and more efficient to implement a rigid variable constrained formula hyper tableau calculus directly, instead of implementing non-clausal tableaux and simplification, and then restricting it to clauses. The interesting point about the emulation is that it suggests a way of generalizing hyper tableaux to non-clausal problems. We show how this works for negation normal form (NNF).

The idea is to look at *disjunctive paths* (d-paths) through formulae instead of clauses. The set of d-paths of a formula ϕ, denoted $dp(\phi)$ is defined by induction over the structure of ϕ as follows.

- If ϕ is a literal or a universally quantified formula, then $dp(\phi) := \{\langle\phi\rangle\}$.
- If $\phi = \alpha_1 \wedge \alpha_2$ is a conjunction, then $dp(\phi) := dp(\alpha_1) \cup dp(\alpha_2)$.
- If $\phi = \beta_1 \vee \beta_2$ is a disjunction, then $dp(\phi) := \{pq \mid p \in dp(\beta_1), q \in dp(\beta_2)\}$.

For instance, for the formula $\phi = (p \wedge \neg p) \vee (q \wedge \neg q)$, this definition gives:

$$\begin{aligned} dp(p \wedge \neg p) &= \{\langle p\rangle, \langle \neg p\rangle\} \\ dp(q \wedge \neg q) &= \{\langle q\rangle, \langle \neg q\rangle\} \\ dp(\phi) &= \{\langle p, q\rangle, \langle p, \neg q\rangle, \langle \neg p, q\rangle, \langle \neg p, \neg q\rangle\} \end{aligned}$$

As d-paths correspond closely to clauses, it is not surprising that the correct generalization of our simplification strategy may be formulated like this:

> Use simplification only to simplify any *leftmost* negative literal of some d-path of a formula on the branch. Use β-expansion only for disjunctions which have at least one d-path that does not contain a negative literal.

For our formula ϕ, β-expansion will thus be applied because of the d-path $\langle p, q\rangle$. Let us call this strategy the *NNF hyper tableau strategy.*

Theorem 5. *The constrained variable tableau calculus with universal variables and the* $simp^{c2u}$ *rule is complete if restricted according to the NNF hyper tableau strategy.*

Proof. See [11]. □

Sometimes, a simplification step permits discarding the original formula ψ. In such cases, a prover using the NNF hyper tableaux strategy has an advantage over usual clausal hyper tableaux, even if the problem is given in clausal form: it can simplify the clause set while proof search is under way. Essentially, unit resolution between a universal branch literal and a clause is performed. For instance, given a literal $[X]p(X)$ and a universal disjunction $[Y]\neg pY \vee rY$, the latter can be destructively simplified to $[Y]rY$ for that branch. This can not be done in normal hyper tableaux, as these do not keep separate clause sets per branch. Note that these separate clause sets do not imply higher memory consumption, because the representation of clauses can easily be shared between branches in an implementation.

The NNF hyper tableau strategy was implemented in the prototypical non-backtracking tableau prover PrInS [10,11]. We are not going to list statistics here, as the power of hyper tableaux has previously been asserted, e.g. in [12]. We shall only state two results on problems found in the TPTP problem library. [19].

With the given strategy, PrInS is able to solve the *Steamroller* problem in the full first order formalization PUZ031+1 in less than 150 ms. This used to be considered a hard problem for a long time, although today, no state-of-the-art theorem prover has difficulties with it. In particular, hyper tableaux are a good way of quickly finding a proof. The interesting aspect of PrInS solving PUZ031+1 is that it does not use CNF transformation. To our knowledge, PrInS is the first *non clausal* theorem prover to have solved the Steamroller problem.

The problem known as *Andrews Challenge* is an example for the advantage of not needing a clause normal form. The full first order formalization SYN036+2 of that problem had a rating of 0.33 up till version 2.4.0 of the TPTP library, meaning that one third of the provers considered state-of-the-art were *not* able to solve it. The reason for this is that the clause normal form for this problem, if computed in the standard way, consists of 128 clauses of length 8. The full first order version in SYN036+2 is built from the equivalence junctor and quantifiers only, and is very small. The NNF PrInS works on is of course significantly larger, because $p \leftrightarrow q$ has to be translated to $(p \vee \neg q) \wedge (\neg p \vee q)$. But the NNF still helps in keeping large parts of the formula nested below the top level operators which are handled first. With the NNF hyper tableaux strategy, PrInS solves SYN036+2 in less than 200 ms. The prover performs 488 α, β and γ expansions and 322 simplification steps. By contrast, a simple version of PrInS without simplification needs 3938 rule applications and about 11 seconds for SYN036+2.

7 Related Work

The idea of using formulae on a branch to simplify other formulae independently been developed by Peltier [16]. The problem of dealing with the instantiation of rigid variables is solved differently however. While we use ordinary first order formulae and attach a syntactic constraint to them, Peltier intertwines constraints and formulae. The possibility of attaching different constraints to different parts of a larger formula might be an advantage of Peltier's approach, but we have not investigated this. Keeping formulae and constraints apart, as we do certainly makes the calculus easier to understand, and easier to reason about.

Recent work by Baumgartner and Tinelli [3] attempts to lift the unit propagation of the Davis Putnam procedure to first order logic. Their *model evolution* calculus does not use rigid variables however, and accordingly does not need constraints.

8 Conclusion

Several possibilities for a first-order version of the simplification rule of Massacci [13,14] were presented. Instead of globally applying unifying substitutions, syntactic constraints are used. Besides soundness and completeness, a finiteness property is discussed, which is important for the design of fair proof procedures. Experimental results are quoted, which show that an efficient proof procedure can be implemented using non-clausal tableaux with a simplification rule. We refer the reader to [11] for a more precise discussion of some of the issues we could only mention briefly here.

Future work includes the refinement of cyclicity tests and development of more goal-oriented simplification strategies than the described hyper-tableaux variant.

Acknowledgments: I am grateful to Reiner Hähnle, Bernhard Beckert, Wolfgang Ahrendt, Magnus Björk and the anonymous referees for their many useful comments.

References

1. Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integration of automated and interactive theorem proving. In W. Bibel and P. Schmitt, editors, *Automated Deduction: A Basis for Applications*, volume II, chapter 4, pages 97–116. Kluwer, 1998.
2. Peter Baumgartner. Hyper Tableaux — The Next Generation. In Harrie de Swart, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, number 1397 in LNCS, pages 60–76. Springer-Verlag, 1998.
3. Peter Baumgartner and Cesare Tinelli. The Model Evolution Calculus. Fachberichte Informatik 1–2003, Universität Koblenz-Landau, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, 2003.

4. Bernhard Beckert and Reiner Hähnle. Analytic tableaux. In W. Bibel and P. Schmitt, editors, *Automated Deduction: A Basis for Applications*, volume I, chapter 1, pages 11–41. Kluwer, 1998.
5. Hubert Comon. Disunification: a survey. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, chapter 9, pages 322–359. MIT Press, Cambridge, MA, USA, 1991.
6. Marcello D'Agostino and Marco Mondadori. The taming of the cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.
8. Martin Giese. Integriertes automatisches und interaktives Beweisen: Die Kalkülebene. Diploma Thesis, Fakultät für Informatik, Universität Karlsruhe, June 1998.
9. Martin Giese. A first-order simplification rule with constraints. In Peter Baumgartner and Hantao Zhang, editors, *3rd Int. Workshop on First-Order Theorem Proving (FTP), St. Andrews, Scotland, TR 5/2000 Univ. of Koblenz*, pages 113–121, 2000.
10. Martin Giese. Incremental closure of free variable tableaux. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proc. Intl. Joint Conf. on Automated Reasoning, Siena, Italy*, volume 2083 of *LNCS*, pages 545–560. Springer-Verlag, 2001.
11. Martin Giese. *Proof Search without Backtracking for Free Variable Tableaux*. PhD thesis, Fakultät für Informatik, Universität Karlsruhe, July 2002.
12. Michael Kühn. Rigid hypertableaux. In *Proc. of KI '97: Advances in Artificial Intelligence*, volume 1303 of *LNAI*, pages 87–98. Springer-Verlag, 1997.
13. Fabio Massacci. Simplification with renaming: A general proof technique for tableau and sequent-based provers. Technical Report 424, Computer Laboratory, Univ. of Cambridge (UK), 1997.
14. Fabio Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In Harrie de Swart, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, volume 1397 of *LNCS*, pages 217–232. Springer-Verlag, 1998.
15. Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computation*, 19(4):321–352, 1995.
16. Nicolas Peltier. Pruning the search space and extracting more models in tableaux. *Logic Journal of the IGPL*, 7(2):217–251, 1999. Available online at http://www3.oup.co.uk/igpl/Volume_07/Issue_02/.
17. Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. In Ganesh Gopalakrishnan and Phillip J. Windley, editors, *Proc. Formal Methods in Computer-Aided Design, Second International Conference, FMCAD'98, Palo Alto/CA, USA*, volume 1522 of *LNCS*, pages 82–99, 1998.
18. Mark E. Stickel. Schubert's steamroller problem: Formulations and solutions. *Journal of Automated Reasoning*, 2:89–101, 1986.
19. Christian B. Suttner and Geoff Sutcliffe. The TPTP problem library — v2.1.0. Technical Report JCU-CS-97/8, Department of Computer Science, James Cook University, 15 December 1997.
20. Jan van Eijck. Constrained hyper tableaux. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, volume 2142 of *LNCS*, pages 232–246. Springer-Verlag, 2001.