A First-order Simplification Rule with Constraints

Martin Giese

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, Germany
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 refinements of clausal first-order proof procedures to non-normal form tableaux. Some experimental results obtained with a prototypical implementation are given.

1 Introduction

Non-normal 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. Another case is the integration of automated and interactive theorem proving [1, 9], 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.

In [11], Massacci presents a simplification rule for propositional and modal tableaux. This rule is of the form

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

where $\psi[\phi]$ is the formula that results form first replacing any occurrence of ϕ in ψ by *true*,[1] and then applying a set of boolean simplifications of the form

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

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], regularity and hyper-tableaux [2]. This is done mainly by specifying the strategy of when and where to apply the *simp* rule. Much of the power of these and other formalisms (e.g. Stålmarck's Proof Procedure [12]) comes from simplification-like rules.

[1] If $\phi = \neg\phi'$, any occurrence of ϕ' may be replaced by *false*. I will omit this complementary case throughout the paper for terseness' sake.

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 [10], 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, that overcome this limitation.

2 Lifting with Constraints

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 is instantiated with a, the disjunction may be simplified to $q(a)$. 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:

$$\begin{array}{c} \psi \\ \phi \end{array} \overset{simp}{\leadsto} \begin{array}{c} \mu(\psi)[\mu(\phi)] \\ \mu(\phi) \end{array}$$

where μ is a mgu of ϕ and some subformula of ψ,
which does not contain bound variables,
and μ is applied on all open branches.

The problem with this approach is that it introduces a new backtracking choice-point, because the applied unifier may not lead to a proof.

2.1 Unification constraints

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

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

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. 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 formula ϕ with constraint C is written as $\phi \ll C$. 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. 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

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

$$\begin{array}{c} \text{where } \xi \text{ is a subformula of } \psi \text{ with no bound variables,} \\ \mu \text{ is a mgu of } \xi \text{ and } \phi, \\ \text{and } C \,\&\, D \,\&\, \phi \equiv \xi \text{ is satisfiable} \end{array}$$

Theorem 1. *The $simp^{c0}$ rule is sound in a free variable tableau calculus with constraints.*

Proof sketch. Let σ be (a ground specialization of) the instantiation used to close a proof. Consider the ground proof-tree obtained by applying σ to all formulae, and omitting those formulae with constraints not satisfied by σ. The resulting proof is easily seen to be closed, and uses a ground version of the simplification rule, which is obviously sound.

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. There is however an important special case: if $D \,\&\, \phi \equiv \xi$ subsumes C, the original formula $\psi \ll C$ may be discarded. Let $simp^{c1}$ be the rule obtained with this modification.

Theorem 2. *The $simp^{c1}$ rule is sound and complete, i.e. a tableaux branch that can be closed by a refutation R, can still be closed by a modified refutation R' after an application of the $simp^{c1}$ rule. Moreover, there is an R' that is at most as large as the original R.*[2]

Proof sketch. Soundness follows from Theorem 1. The rest is shown by constructing R' from R by a proof transformation, in which rule applications on (descendents of) a discarded original formula $\psi \ll C$ can either be applied to (descendents of) the simplified formula, or be discarded altogether.

The $simp^c$ rules enjoy an interesting finiteness property. 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, δ 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.*[3] *The same is true for the $simp^{c1}$ rule.*

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

[3] This might be enforced through a regularity condition that forbids the application of a rule that would lead to variant formulae on one of the extended branches.

Proof sketch. 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 $simp^{c1}$ 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.

2.2 Dis-unification constraints

In most cases, the original formula $\psi \ll C$ has to be kept on the branch in the $simp^{c1}$ rule. This can lead to redundancies as exemplified by the following tableau branch for the set of formulae $\{p(a), q(a), \neg p(X) \vee \neg q(X) \vee r(X)\}$:

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

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.

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

$$\begin{array}{lcc}
\psi \ll C & \overset{simp^{c2}}{\leadsto} & \psi \ll C \mathbin{\&} !(D \mathbin{\&} \phi \equiv \xi) \\
\phi \ll D & & \phi \ll D \\
 & & \mu(\psi)[\mu(\phi)] \ll (C \mathbin{\&} D \mathbin{\&} \phi \equiv \xi)
\end{array}$$

$$\begin{array}{c}
\text{where } \xi \text{ is a subformula of } \psi \text{ with no bound variables,} \\
\mu \text{ is a mgu of } \xi \text{ and } \phi, \\
\text{and } C \mathbin{\&} D \mathbin{\&} \phi \equiv \xi \text{ is satisfiable}
\end{array}$$

In addition, we allow formulae with unsatisfiable constraints to be discarded. 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

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

The constraint $!X \equiv a$ now prevents the simplification of 3 with 2. Simplification of 4 with 2 gives a new constraint of $X \equiv a \,\&\, !X \equiv a$ for 4, which is unsatisfiable, so 4 can be discarded after adding the literal $r(a) \ll X \equiv a$

Theorems 2 and 3 also hold for the $simp^{c2}$ rule. The principal drawback of this variant 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 constraints of the form $C_0 \;\&\; !C_1 \;\&\; !C_2 \ldots$, where the C_i are conjunctive unification constraints as in Sec. 2.1. 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 are fairly easy to check for these constraints.

3 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 $\{p(a), p(b), p(c), \forall x.\neg p(x) \lor q(x)\}$, 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 $\neg p(X_i) \lor q(X_i)$ 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].[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

[4] They have been used to solve a similar problem in equality handling in [3].

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:

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

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:

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

$$\begin{array}{c} \text{where } \xi \text{ is a subformula of } \psi \text{ with no bound variables,} \\ \mu \text{ is a mgu of } \xi \text{ and } \phi, \\ \text{and } C \,\&\, D \,\&\, \phi \equiv \xi \text{ is satisfiable} \end{array}$$

This rule is sound and complete for the free-variable tableau calculus with universal variables, but there is a difficulty with the finiteness property. To get the full power out of this rule, it is necessary to rename universal variables in the original formulae to make them disjoint. But this renaming destroys finiteness. 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

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

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.

There are problems, like e.g. Schubert's Steamroller [13], or the 'Natural Language' problems submitted to this FTP workshop,[5] in which simplification with universal variables actually always terminates. To handle these 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.

4 Experimental Results

PrInS is a theorem prover implemented in the Java programming language. It uses a simplification rule with universal variables, but only uses unification constraints. Like leanT^AP, it works on formulae in negation normal form (NNF) for simplicity, but it would be easy to extend it to non-normal form tableaux. To avoid redundancy, it uses a NNF variant of positive hyper-tableaux:

– For the sake of simplicity, simplification is not performed below quantifiers. Quantified formulae are regarded as positive literals in the determination of disjunctive paths.
– Positive literals are used to simplify leftmost negative literals on disjunctive paths of β formulae.
– A mechanism recording the ancestry of formulae is used as a cycle-detection heuristic.
– β expansion is only performed on formulae with at least one disjunctive path that contains only positive literals or quantified formulae.
– β applications not needed for a closure are *pruned*, see e.g. [4, 2].

Instantiation of rigid variables is handled by a backtracking-free process outlined in [8]. No equality handling is built in, so the standard axiomatization is used, which leads to poor performance on equality porblems.

Running on a 440 MHz Sun SPARC Ultra 10 with JDK 1.2, PrInS can prove, resp. disprove all of the 'johan' problems in the FTP'00 'Natural Language' set in a total of 4.0 CPU seconds. Of the 97 problems in the 'Mathematics' set, 68 are proven and 2 disproven with a time limit of 15 seconds per problem.

Schubert's Steamroller in the TPTP PUZ031+1 formulation is solved in 0.4 seconds. Of the 73 Pelletier problems in the TPTP collection (FOF formulations were taken where available), 63 are solved in 2.3 seconds. The size 8 pigeonhole problem, MSC007-1.008, is solved in 85 seconds. The ones not solved are mainly from the GRP and LCL categories.

To our knowledge, comparable results have not previously been achieved with a non-clausal tableaux prover.

5 Conclusion

Several possibilities for a first-order version of the simplification rule of Massacci [10, 11] were presented. Instead of globally applying unifying substitutions, syntactic constraints are used. Besides soundness and completeness, a finiteness property

[5] See `http://www.uni-koblenz.de/ftp00/ProblemSets/`

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.

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

Acknowledgements: I am grateful to Reiner Hähnle, Elmar Habermalz and Bernhard Beckert for comments on drafts of this paper.

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, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In José Júlio Alferes, Luís Moniz Pereira, and Ewa Orłowska, editors, *Proc. European Workshop: Logics in Artificial Intelligence, JELIA*, volume 1126 of *LNCS*, pages 1–17. Springer-Verlag, 1996.
3. Bernhard Beckert. A completion-based method for mixed universal and rigid e-unification. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI 814, pages 678–692. Springer-Verlag, 1994.
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. Proof search without backtracking using instance streams. Position Paper, submitted to FTP 2000. Available online at `http://i11www.ira.uka.de/~giese/ftp00istr.ps.gz`.
9. Martin Giese. Integriertes automatisches und interaktives Beweisen: Die Kalkülebene. Diploma Thesis, Fakultät für Informatik, Universität Karlsruhe, June 1998.
10. 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.
11. 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.
12. 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.
13. Mark E. Stickel. Schubert's steamroller problem: Formulations and solutions. *Journal of Automated Reasoning*, 2:89–101, 1986.