

Proof Output and Transformation for Disconnection Tableaux^{*}

Philipp Correll & Gernot Stenz

Institut für Informatik
Technische Universität München
D-85748 Garching, Germany
{correll,stenzg}@in.tum.de

Abstract. For applications of first-order automated theorem provers in a wider verification context it is essential to provide a means of presenting and checking automatically found proofs. In this paper we present a new method of transforming disconnection tableau proofs found by the prover system DCTP into a series of resolution inferences representing a resolution refutation of the proof problem.

1 Introduction

Recent years have seen an increased interest in the use of formal methods, not only in scientific environments but also in industrial applications. First-order theorem provers have also been used in such contexts but usually just as proof oracles returning *yes* or *no* wrt. a given proof task. However, the *pervasive* use of formal methods requires a more detailed form of output. Not only are automated theorem provers required to return proofs but those proofs should be presented in a form that allows them to be mechanically checked. Here, we present an algorithm that transforms disconnection tableau proofs found by the automated theorem prover DCTP into ordinary resolution proofs. Resolution has been chosen because available proof checking tools are mostly built for checking resolution proofs; also switching to a different (preferably simpler) paradigm for proof checking simplifies the task of finding errors in the original proof method.

This paper is organised as follows: First, the disconnection calculus is briefly revisited. Then in Section 3 the transformation procedure is outlined. Section 4 explains how certain calculus refinements can be integrated into the transformation procedure. Finally, the concluding Section 5 describes possible extensions of the proof transformation procedure.

2 The Disconnection Tableau Calculus

Detailed descriptions of the disconnection calculus can be found in [2, 3, 5]. Here it is sufficient to recapture the most significant aspects of this calculus. For

^{*} This work was partially funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the author(s).

this we use the standard terminology for clausal tableaux. The disconnection tableau calculus consists of a single complex inference rule, the so-called *linking rule*. This linking rule is used to systematically expand a clausal tableau with suitable instances of the input clauses. The guidance for the application of the linking rule is provided by *links*, pairs of potentially complementary subgoals on the tableau path of the open subgoal to be extended. All variables on the tableau are considered to be implicitly universally quantified. In order to provide an initial set of links to start the tableau construction, an arbitrary *initial active path* is selected, marking one subgoal from each of the input clauses. A branch in a disconnection tableau is called \forall -*closed* if it contains two literals K and L such that $K\sigma$ is the complement of $L\sigma$ where σ is a substitution identifying all variables in the tableau. The disconnection tableau calculus is refutation complete for first-order clause logic, provided the linking rule is applied in a fair manner. Additionally, it is sufficient to use each link only once on each branch. If the set of links on an open branch is exhausted, the literals on the branch describe a model for the clause set.

3 The Basic Procedure

The aim of the basic proof transformation procedure is to transform its input, a closed disconnection tableau for a clause set S , into a resolution refutation of S .

Prover Output. In our system, proof search and proof output and transformation are kept separate. While searching for a proof, DCTP simply outputs its actions in a sequential manner. For the most basic proof output, there are three types of output statements:

```

initial(clause_id, literals) .
instance(clause_id, parent_literal, initial_id, literals) .
close(leaf_literal, branch_literal) .

```

First, the **initial** statements list the set of input clauses. The **instance** statements denote which instances of the input clauses have been placed where on the tableau. Finally, the **close** statements denote where branch closures have occurred and which branch literals were used for these closures. The parameters of all statements contain the necessary information to enable the complete reconstruction of the proof tableau. Except for the **initial** statements, DCTP will only output ground clauses. In accordance with the closure condition given in Section 2, all variables are replaced by the special constant $\$$.

The Transformation Procedure. The main problem encountered here is that proof tableaux represent dual clause forms (a tableau branch is the conjunction of its literals, while the entire tableau is the disjunction of its branches) whereas resolution proofs are clause sets (conjunctions of the input clauses, the resolvents and the empty clause). Also, the disconnection calculus is (in its pure form) purely analytic; clauses are not recombined as in a resolution step. So proof tableaux cannot be translated into resolution derivations in a straightforward manner. We solve this problem by replacing the actual tableau structure

by the series of branch closures indicated by the `close` statements. Basically each one of these closures is translated into a resolution step. To obtain the proper input clauses for these resolution steps we have introduced the concept of the *annotated resolvent*. An annotated resolvent r_c is assigned to each tableau clause c and is initialised to the set of literals of c . The resolution steps generated by our procedure operate solely on these annotated resolvents, not on the original tableau clauses. Every time a subgoal $L \in c$ is solved, r_c is updated to the resolvent resulting from the solution of L . As the number of unresolved literals is never greater than the number of unsolved tableau subgoals, running the procedure on a closed tableau will finally produce the empty clause.

<pre> Input : a clause set S and a closed disconnection tableau T for S Output: a resolution proof for S 1 foreach clause $s \in S$ do print(s) 2 foreach clause $c \in T$ do setAnnotatedRes(c, c) 3 repeat 4 $L \leftarrow$ getOpenLeaf(T) 5 $\sim L \leftarrow$ getComplementOnBranch(L, T) 6 $r_{new} \leftarrow$ resolve(getAnnotatedRes(L) getAnnotatedRes($\sim L$)) 7 markLiteralSolved(L) 8 currentClause \leftarrow getClause(L) 9 while (allNodesMarkedInClause(currentClause)) do 10 markLiteralSolved(parentNodeOf(currentClause)) 11 currentClause \leftarrow getClause(parentNodeOf(currentClause)) 12 end 13 setAnnotatedRes(currentClause, r_{new}) 14 printResolutionStep(r_{new}) 15 until ($r_{new} = \square$) </pre>

Fig. 1. The basic transformation algorithm.

The transformation algorithm starts by reconstructing the proof tableau based on the prover output. Then, the input clause set is written to output and the annotated resolvents of the tableau clauses are initialised. This is followed by an iteration over all branch closures. For each closure, the annotated resolvents of the clauses containing the closing branch literals L and $\sim L$ are used as the input for a resolution step producing a new resolvent r_{new} . The leaf L is marked as solved and r_{new} is passed on as the new annotated resolvent to the clause containing the next unsolved subgoal in the tableau in clause `currentClause`. Finally the resolution step producing r_{new} is written to output.

It should be noted that in the algorithm of Figure 1, the function `getOpenLeaf` and the variable `currentClause` identify different concepts. `getOpenLeaf` always returns the next *leaf node* in a depth-first manner, while `currentClause` holds the lowermost clause of the leaf node's branch containing an unsolved subgoal. Also,

the depth-first processing order of function `getOpenLeaf` is a basic requirement for securing the correctness of the algorithm depicted in Figure 1.

The transformation output consists of two kinds of statements: the `initial` statements, displaying the input clause set and the `resolvent` statements specifying the resolution inferences.

```

initial(clause_id, literals) .
      ⋮
resolvent(resolvent_id, [annotatedResolvent1_id, annotatedResolvent2_id]) .

```

An example showing how the transformation procedure works is presented in Figure 2. The example depicts a sequence of two transformation steps of a partial tableau containing the clauses c_1, \dots, c_3 . The boxed tableau nodes indicate the complementary literals to be resolved. An annotated resolvent r_{c_i} highlighted in grey is assigned to each clause c_i . In subfigure (a) the annotated resolvents have been initialised to the respective clause literals. The annotated resolvents r_{c_2} and r_{c_3} are resolved and the resulting resolvent replaces r_{c_3} in subfigure (b). There, the second literal L_1 of c_3 is solved by a closure with $\sim L_1$ of c_1 . Again, the annotated resolvents are used for a resolution step. Now all literals of c_3 have been solved (and with them $\sim L_3$ of c_2), so the resulting resolvent $L_2 \vee L_4$ is passed up to replace r_{c_2} in subfigure (c).

Factorisation Effects. Analytic tableau calculi do not allow certain optimisations across proof clauses such as factorisation. This means that in a tableau proof a subgoal has to be solved several times, whereas in the corresponding resolution refutation the same subgoal has to be resolved only once. Although a partial remedy to this problem will be discussed in Section 4, it is still possible that the necessity arises to repeatedly solve the same subgoal. These multiple solutions can be ignored in the resolution context. Therefore, a closure statement will produce a resolution step only if the closing leaf $K \in d$ is contained in the current annotated resolvent r_d of d . This may also lead to a premature termination of the transformation algorithm when the empty clause is produced before all closure statements have been processed.

Tautological Resolvents. The disconnection calculus can delete tautological clauses, yet it must allow linking steps producing the tableau equivalent of tautological resolvents in order to remain complete [5]. Therefore, the transformed tableau proof can contain tautological resolvents. The resolution proofs thus generated are sound but not optimal. We have extended the transformation algorithm by a markup technique allowing the removal of tautological resolvents and their descendants.

4 Extensions

The algorithm of Figure 1 covers only the most basic version of the disconnection tableau calculus. The calculus of DCTP makes use of many advanced refinements, so the transformation procedure has to be enhanced in order to handle these techniques.

This also indicates how folding-up is transformed into resolution steps: below the c-point of the subgoal folded up, an atomic cut is inserted and becomes part of the transformation output.

Unit Theory. In order to obtain a more efficient proof procedure, it is vital to make use of *unit clauses* via *unit subsumption* and *unit simplification* [5]. As subsumption is only used to guide the proof search it does not affect the proof transformation. *Unit simplification* can be directly transformed into unit resolution.

5 Conclusion and Future Work

In this paper we have introduced a method to transform and output disconnection tableau proofs in resolution style. As most of the complications of the disconnection calculus pertain only to the proof search and not to the proofs themselves, the transformation can be realised in an elegant way. In addition, the procedure can produce redundancy-free tableau proof trees in PSTricks format. The transformation procedure described here has been implemented in Scheme as a separate converter tool communicating with the DCTP prover. Both the converter and DCTP are available from the authors. Two important topics not addressed in this paper will be mentioned briefly.

Equality. The transformation of disconnection proofs for problems with equality is not part of our current work. The main problem here is that the built-in theory equality reasoning of DCTP [3] is not analytical due to the addition of new *expansion literals* to clause instances. Therefore, a translation of these *eq-linking* steps into paramodulation steps is far more difficult.

CNF Conversion. As most proof problems are stated in full first-order logic, the prover output should include the clause form transformation. However, the resulting output would be checkable only by a higher order prover system.

References

1. R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–338, 12/1994.
2. R. Letz and G. Stenz. Proof and Model Generation with Disconnection Tableaux. In *Proceedings, LPAR 2001*, pages 142–156. Springer, Berlin, 12/2001.
3. R. Letz and G. Stenz. Integration of Equality Reasoning into the Disconnection Calculus. In *Proceedings, TABLEAUX-2002, LNAI 2381*, pages 176–190. Springer, Berlin, 7/2002.
4. F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.
5. G. Stenz. *The Disconnection Calculus*. Logos Verlag, Berlin, 2002. Dissertation, Fakultät für Informatik, Technische Universität München.