

# Unit Propagation in a Tableau Framework<sup>\*</sup>

Gernot Stenz

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

**Abstract.** Unit propagation is one of the most important techniques of efficient SAT solvers. Unfortunately, this technique is not directly applicable to first-order clausal tableaux. We show a way of integrating a variant of unit propagation into the disconnection calculus and present some results obtained with an implementation of unit propagation in the DCTP theorem prover that show the usefulness of our new method.

## 1 Introduction

Over the last years, SAT solvers based on the Davis-Putnam-Loveland-Logeman (DPLL) method have become increasingly popular and successful. One of the single most important features of the DPLL procedure is the use of *unit propagation* to guide the proof search. In fact, when DPLL was used as the basis for a new first-order proof method, the *model evolution calculus* [1], the ability to use unit propagation for guiding the proof search was named as one of the prime motivations. Therefore it would be of great interest to integrate unit propagation into classic clausal tableau calculi as well. However, clausal tableau and semantic trees are of a different nature, as are the respective ways of searching for a proof. In [3], a form of unit propagation is introduced for propositional sequent calculi, but without considering the first-order case. In this paper, we present a method of adapting unit propagation to a first-order clausal tableau calculus, the disconnection calculus implemented in the DCTP prover system.

This paper is organised as follows: First, the disconnection calculus is briefly revisited. Section 3 describes the SAT version of unit propagation. Section 4 explains how unit propagation can be integrated into the disconnection calculus. Finally, after some experimental results have been given in Section 5, we conclude with Section 6.

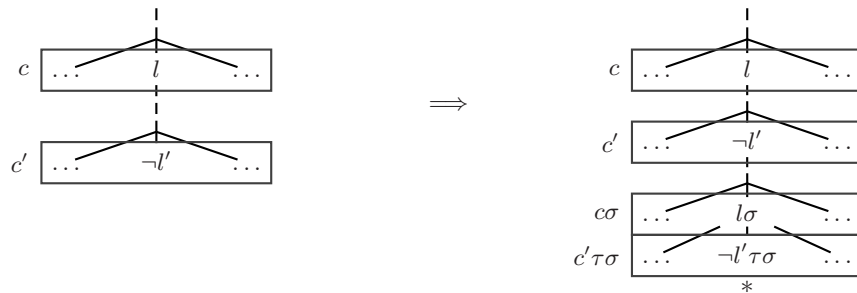
## 2 The Disconnection Tableau Calculus

Detailed descriptions of the disconnection calculus can be found in [6]. Here it is sufficient to recapture the most significant aspects of this calculus. For

---

<sup>\*</sup> 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  $\sigma$  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.



**Fig. 1.** Illustration of a linking step.

### 3 Unit Propagation for Semantic Trees

DPLL procedures [2] operate on a data structure called a *semantic tree*. A semantic tree is a succession of atomic cuts arranged in a binary tree. The cuts that form the semantic tree are made over the propositional variables occurring in the input clauses of the proof problem. A branch  $B$  of the semantic tree is closed if in its context an input clause becomes false, i.e. there is an input clause  $c$  such that for every literal  $l_1, \dots, l_m \in c$  the complement  $\sim l_i$  of  $l_i$  is on  $B$ . A semantic tree whose branches are all closed constitutes a proof for the input formula. An un-extendable open branch indicates a model for the input formula. An important point regarding semantic trees is that the order of the variables used for the cuts may be different on every branch. Also, choosing the right order for the variables is of vital importance.

Unit propagation [7] is a method for finding the locally optimal choice of cut variable. For every input clause the *context length* of that clause is recorded, that is the number of clause literals not complemented by the current branch. The branch literals  $l_j$  are used to successively decrement the context length of all input clauses containing  $\sim l_j$ . When an input clause has a context length of 1, the remaining un-complemented clause variable will be used for the next cut. Thus, one of the new branches will immediately be closed and the proof search can continue without an actual split of the branch and with an extended branch context that can again be used to decrement the context lengths.

## 4 Unit Propagation for Disconnection Tableaux

Disconnection tableaux are fundamentally different from semantic trees. Here, the branches are labelled with the subgoals of linking instances of input clauses. Unlike in the DPLL procedure, where a static check against a fixed set of clauses is made, clause instances in a disconnection tableau change with every instantiation. Therefore, a unit propagation procedure does not work on the tableau clauses, but on the possible clause instances. The counts of un-complemented subgoals consequently are not associated with the individual clauses but with the linking steps and the clause instances generated by them. Figure 2 shows how links create new branch clauses and how context subgoals affect the number of open branches created.

In the disconnection calculus, unit propagation is realised in a number of different ways:

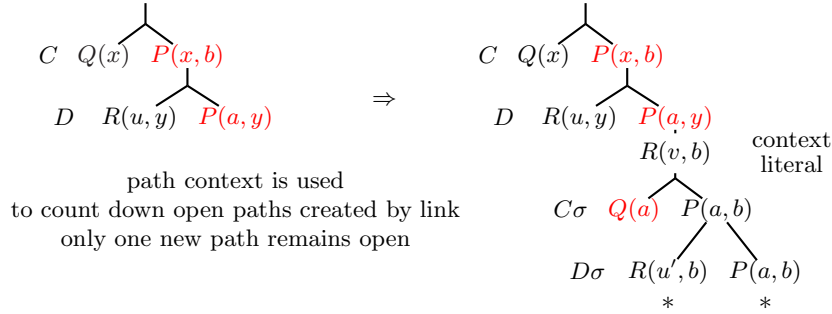
*Ordinary Unit Propagation.* When the current path context ensures that for a link  $\ell$  all but one of the linking subgoals can immediately be closed, applying  $\ell$  will augment the path context without causing actual branching. This is the equivalent of the propositional unit propagation described in Section 3.

*Lemma Generating Propagation.* When all but one of the linking subgoals of a link can be simplified, the remaining linking subgoal will become a unit lemma. This is a special case of ordinary unit propagation, but preferable as unit lemmas are more useful and versatile than path context subgoals.

*Branch Closure Propagation.* When the current path context allows the immediate  $\forall$ -closure of all the linking subgoals of a link  $\ell$ , then an application of  $\ell$  can be used to close the current branch. In a DPLL context, branch closure propagation has no equivalent, since an according branch would be closed without further action.

*Complementary Unit Propagation.* When all linking subgoals of a link  $\ell$  are simplified by the current set of unit lemmas, an application of  $\ell$  will finish the proof.

Unlike in propositional semantic trees, where unit propagation can be used as a deciding strategy, in first-order tableaux unit propagation serves as a mere *heuristic* to guide the proof search. It can easily be seen that, given a unit lemma  $P(a)$  and a path clause  $\neg P(x) \vee P(f(x))$ , the uncontrolled use of unit propagation would merely produce new unit lemmas  $P(f(a)), P(f(f(a))), P(f(f(f(a))))$ , ... that would not necessarily contribute to the proof search. Therefore, in order



**Fig. 2.** Reduction of open instance subgoals by the path context

to maintain the fairness condition required in Section 2, ordinary unit propagation and lemma generating propagation are used only in an intermittent fashion, alternating with regular linking steps. On the other hand, branch closure propagation (and also, trivially, complementary unit propagation) is not subject to such an alternation scheme, since the fairness requirement is restricted to the current (and subsequently closed) branch.

## 5 Implementation and Results

Unit propagation is implemented in the DCTP prover system[5] by means of a *propagation index*. This index is organised as a standard discrimination tree. Upon the generation of a new link, all subgoals of the potential linking instances are entered into the propagation index. Also the number of open subgoals created by the application of the link is stored within the link.

Whenever the branch context is extended by a new subgoal, that subgoal is used to search the propagation index for matching entries. All matching entries not marked by another branch subgoal<sup>1</sup> are marked as closed and the open subgoal counts of their respective links are decremented. Upon backtracking, the markings caused by the branch subgoal are undone and the link counts corrected. Unit lemmas will prune the propagation index destructively, erasing all matching branches, such that the decrement of the open subgoal counts for the affected links cannot be backtracked. All links classified according to the propagation categories of Section 4 are kept in four linear lists. When selecting a new link, DCTP first checks these lists for valid entries, for ordinary unit propagation and lemma generating propagation an additional alternation scheme is operated.

Due to the complications that arose during the implementation of unit propagation, no comprehensive set of test runs has been conducted with the new version of DCTP in comparison to the pre-propagation version. However, a very small number of tests has been run and the results obtained substantiate our

<sup>1</sup> A branch may contain multiple occurrences of the same non-ground subgoal and only the topmost occurrence may be used to decrement link counts.

claims. With the use of unit propagation, DCTP has been able to solve a number of SAT solver benchmark problems (ssa2670-130, ssa2670-141) that previously were simply beyond the abilities of DCTP (and which also cannot be solved by the E prover<sup>[4]</sup>).

In the TPTP SYN class of syntactic ALC problems, two of the most difficult problems SYN440 and SYN457 were tested with 2592 different search strategies of DCTP (among others, with and without unit propagation). The tests were conducted on a 3GHz AMD 64-bit machine for a time of 40 seconds each. For SYN440, the results showed that without the new techniques the fastest solutions were found within 12 seconds. With unit propagation, the fastest proof could be found in less than one second. For SYN457, the problem could not be solved at all within the given 40 seconds without the use of unit propagation, with unit propagation the fastest proof was obtained in 11.15 seconds.

A third problem tested showing the usefulness of unit propagation is the TPTP problem PUZ037-3, the Rubik's Cube problem where three rotations of the cube are needed for a solution. In its standard formulation, this problem had hitherto been out of reach for DCTP. Using unit propagation PUZ037-3 can be solved in less than three seconds.

It should be noted that unit propagation is going to have little effect on problems where the complexity of the problem is encoded in the term structure and not the clause structure such as the TPTP LCL problems.

## 6 Conclusion

In this paper we described the adaptation of the unit propagation SAT technique to a clausal tableau framework and the successful integration of this technique into the DCTP theorem prover. This also serves to show that it is worthwhile looking beyond first-order theorem proving for new techniques. We are confident that other methods from the SAT domain, such as the extensive lemma generation and learning, could be applied in a first-order environment, too.

## References

1. P. Baumgartner and C. Tinelli. The Model Evolution Calculus. In Franz Baader, editor, *Automated Deduction – CADE-19, LNAI 2741*. Springer, 2003.
2. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the Association for Computing Machinery*, pages 394–397, 1962.
3. Fabio Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In Harrie de Swart, editor, *Proceedings, TABLEAUX-1998, LNAI 1397*, pages 217–231. Springer, Berlin, May 1998.
4. S. Schulz. System Abstract: E 0.61. In *Proceedings, IJCAR-2001, LNAI 2083*, pages 370–375. Springer, Berlin, June 2001.
5. G. Stenz. DCTP 1.2 – System Abstract. In *Proceedings, TABLEAUX-2002, LNAI 2381*, pages 335–340. Springer, Berlin, July 2002.
6. G. Stenz. *The Disconnection Calculus*. Logos Verlag, Berlin, 2002. Dissertation, Fakultät für Informatik, Technische Universität München.

7. H. Zhang and M. E. Stickel. An efficient algorithm for unit propagation. In *Proceedings, AI-MATH'96*, December 1996.