Anti-Links for Boolean Function Manipulation[*]

Bernhard Beckert Reiner Hähnle
Institute for Logic, Complexity, and Deduction Systems
University of Karlsruhe, D-76128 Karlsruhe
$\langle name \rangle$@ira.uka.de
http://i12www.ira.uka.de/~$\langle name \rangle$/

Neil V. Murray Anavai Ramesh
Institute for Programming & Logics
Department of Computer Science
University at Albany, Albany, NY 12222
{rameshag,nvm}@cs.albany.edu

Manipulation of Boolean functions has important applications in such fields as hardware verification, non-monotonic reasoning, and decision support. Most often, in such applications, the set of prime implicates/implicants of a large formula must be computed.

Many algorithms have been proposed to compute the prime implicates of propositional Boolean formulas, e.g. [4, 5]. Most of them assume that the input is in conjunctive/disjunctive normal form (CNF/DNF).

The bottleneck of such algorithms is usually the large number of subsumed (i.e. non-minimal) disjunctive/conjunctive paths through the given formula. Therefore, it is crucial to remove as much redundancy of this kind as possible, before any explicit enumeration of prime implicates/implicants starts. Even if such explicit computation of prime implicates/implicants is not an immediate goal, removing any redundancies due to subsumed disjunctive/conjunctive paths is usually highly desirable.

Among the most successful methods used to deal with this task are binary decision diagrams (BDDs) [2, 3]. It is, however, well known that BDD methods do not work well on some classes of formulas, in particular, when they are used to compute prime implicants. Another restriction of BDDs is that they (at least the efficient versions) essentially produce an XOR-AND normal form, but this is not always what is needed.

Here, we present a technique based on anti-links for identifying and removing subsumed paths, that operates on formulas in negation normal form (NNF).[1] It is not an alternative to BDDs and other methods for computing prime implicants, but is intended to be used as a supplementary pre-processing step that enhances the overall performance of the system.

An *anti-link* $\{A_X, A_Y\}$ consists of occurrences A_X and A_Y of a literal A in an NNF formula F. If the (largest) subformulas X and Y of F that contain A_X and A_Y, respectively, are disjunctively connected, we call $\{A_X, A_Y\}$ a *disjunctive anti-link*. If X and Y are conjunctively connected, then we call $\{A_X, A_Y\}$ a *conjunctive anti-link*.

If the formula F contains a disjunctive path p that is subsumed by a distinct path p', then F contains a disjunctive or a conjunctive anti-link. Unfortunately, the converse is not true: the presence of anti-links does not imply the presence of subsumed paths. There is, however, a large class of anti-links, that we call *redundant*, that is always an indicator for subsumed paths: A disjunctive anti-link $\{A_X, A_Y\}$ in F is *redundant* if either A_X or A_Y is the argument of a conjunction. In that case certain disjunctive paths through X and Y that contain A_X or A_Y are subsumed by a disjunctive path in F that contains the anti-link.

Example 1 *Consider the formula*

$$F \;=\; ((A_X \vee C) \wedge B) \;\vee\; (A_Y \wedge (E \vee C)) \;.$$

The two occurrences of A form a redundant disjunctive anti-link. The disjunctive path $p = \{A_X, C, E, C\}$ is subsumed by the path $p' = \{A_X, C, A_Y\}$ (with literal set $\{A, C\}$), that contains the anti-link.

The two occurrences of C are both arguments of disjunctions, and thus comprise a non-redundant anti-link.

[*] A long version of this abstract is currently under review. Some of the results to be presented in the talk have been published in preliminary form in [1, 7].

[1] This restriction is reasonable, since formulas that contain negations, conjunctions, disjunctions, and implications at any level can be converted to NNF in polynomial time.

Although only redundant disjunctive anti-links contribute directly to subsumed paths, non-redundant anti-links do not prohibit their existence. However, non-redundant anti-links do not themselves provide any evidence that such paths are in fact present.

The identification of redundant disjunctive anti-links can be done easily by checking whether at least one of A_X and A_Y is the argument of a conjunction. After identifying a redundant anti-link, it is possible to remove it using the *disjunctive anti-link dissolvent* (DADV) operator;[2] in the process, paths through A_X and A_Y that are subsumed by a path containing the anti-link are eliminated, and the two occurrences of the anti-link literal are collapsed into one. All other disjunctive paths remain unchanged, and the resulting formula is therefore logically equivalent to the formula F. Note, that the DADV operator does not in general produce a formula in clause form (CNF or DNF).

The operation is closely related to Path Dissolution [6]; DADV is similar to the operator used there to remove unsatisfiable (or tautological, in the dual case) paths. Path Dissolution works by selecting a link (i.e., occurrences of a literal A and its *negation* $\neg A$) and restructuring the formula so that all paths through the link are eliminated. One consequence of eliminating all paths through a link is strong completeness: *Any* sequence of dissolution steps will eventually create a linkless formula.

Example 2 *Consider again the formula F from Example 1. The result of applying DADV to F is*

$$(B \vee (A \wedge (E \vee C))) \wedge (A \vee C) .$$

The path $\{A, C, E, C\}$ has been removed, and the path $\{A_X, C, A_Y\}$ has been replaced by $\{A, C\}$. All other disjunctive paths are still present.

The cost of the DADV operation is linear in the size of the smallest subformula of F containing the anti-link. Also, conjunctively connected literals in F do not become disjunctively connected in DADV(F). Thus, truly new disjunctive anti-links are not introduced. However, parts of the formula may be duplicated, and this may give rise to additional copies of anti-links not yet removed. Nevertheless, persistent removal of redundant disjunctive anti-links is a terminating process, because the number of disjunctive paths is strictly reduced at each step.

The general problem of computing, for a given formula F, an equivalent formula that is minimal w.r.t. its disjunctive paths is NP-hard. Nevertheless, redundant disjunctive anti-links are easily recognized, and eliminating their corresponding subsumed paths can be done without direct subsumption checks.

Although prime implicate/implicant problems are intractable in general, our techniques perform exponentially better than others on certain examples. In addition, we are able to improve performance greatly on some inherently exponential examples.

Some experimental results of a system based on dissolution and the algorithm PI for computing prime implicates are reported in [7]. That system is currently being extended; some anti-link operations are already implemented and have shown to improve performance.

Possible extensions include the generalization of anti-links to many-valued logic, and the application to identical subformulas instead of literals (based on a generalization of the well-known purity rule for CNF formulas).

References

[1] B. Beckert, R. Hähnle, N. Murray, and A. Ramesh. On anti-links. In *Proc., 5th Int. Conf. on Logic Programming and Automated Reasoning (LPAR), Kiev*, LNCS 882, pages 275–289. Springer, 1994.

[2] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE trans. on computers*, c-35(8):677–691, August 1986.

[3] O. Coudert and J.-C. Madre. Implicit and incremental computation of primes and essential implicant primes of boolean functions. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 36–39, 1992.

[4] J. de Kleer. An improved incremental algorithm for computing prime implicants. In *Proceedings, AAAI-92, San Jose, CA*, pages 780–785, 1992.

[5] P. Jackson. Computing prime implicants incrementally. In *Proceedings, 11th International Conference on Automated Deduction (CADE), Saratoga Springs, NY*, LNCS 607, pages 253–267, June 1992.

[6] N. Murray and E. Rosenthal. Dissolution: Making paths vanish. *Journal of the ACM*, 48(3):504–535, July 1993.

[7] A. Ramesh and N. Murray. Avoiding tests for subsumption. In *Proc. National Conference on Artificial Intelligence, Seattle/WA, USA*, 1994.

[2] Due to space restrictions, we cannot give its general definition here.