Duality in Knowledge Compilation Techniques*

Neil V. Murray[1] and Erik Rosenthal[2]

[1] Department of Computer Science, State University of New York, Albany, NY 12222, USA
nvm@cs.albany.edu
[2] Department of Mathematics, University of New Haven, West Haven, CT 06516, USA
erosenthal@newhaven.edu

Abstract. Several classes of propositional formulas have been used as target languages for knowledge compilation. Some are based primarily on c-paths (essentially, the clauses in disjunctive normal form); others are based primarily on d-paths. Such duality is not surprising in light of the duality fundamental to classical logic. There is also duality among target languages in terms of how they treat links (complementary pairs of literals): Some are link-free; others are pairwise-linked (essentially, each pair of clauses is linked). In this paper, both types of duality are explored, first, by investigating the structure of existing forms, and secondly, by developing new forms for target languages.

1 Introduction

Several classes of propositional formulas have been used as target languages for knowledge compilation, including *Horn clauses*, *ordered binary decision diagrams*, *tries*[1], and sets of *prime implicates/implicants*—see, for example, [2, 3, 8, 12, 20, 21]. The discovery of formula classes that have properties that are useful for target languages is ongoing. Within the past several years, *decomposable negation normal form* [6] (DNNF), linkless formulas called *full dissolvents* [15], *factored negation normal form* [9] (FNNF), and *pairwise-linked* clause sets, in which every pair of clauses contain complementary literals (called EPCCL in [10]), have been investigated as target languages. It is not surprising that dualities arise when comparing target languages, but the extent of their prevalence may be surprising.

Most research has restricted attention to *conjunctive normal form* (CNF). This may be because the structure of *negation normal form* (NNF) can be quite complex. However, there is growing interest in target languages that are subclasses of NNF. Decomposable negation normal form, studied by Darwiche [5, 6], is one such class. They are linkless and have the property that atoms are not shared across conjunctions. Every DNNF formula is automatically a *full dissolvent*—the end result of applying path dissolution to a formula until it is linkless. Although linkless, full dissolvents may share atoms across conjunctions. It turns out that many of the applications of DNNF depend primarily on

* This research was supported in part by the National Science Foundation under grant CCR-0229339.

[1] A variation of a tree, often used to store dictionaries.

M.-S. Hacid et al. (Eds.): ISMIS 2005, LNAI 3488, pp. 182–190, 2005.

the linkless property and are available and equally efficient with full dissolvents [15]. Moreover, full dissolvents can be advantageous both in time and in the size of the resulting formula.

The class FNNF of factored negation normal form formulas is introduced in [9]. They provide a canonical representation of arbitrary boolean functions. They are closely related to BDDs, but there is a DPLL-like tableau procedure for computing them that operates in PSPACE.

The pairwise-linked formulas introduced by Hai and Jigui [10] appear to be structurally quite different from—and rather unrelated to—DNNF, full dissolvents, or to FNNF formulas. For one thing, pairwise-linked formulas are in CNF and the others are not; for another, pairwise-linked formulas have many links while the others are link-free. It turns out, however, that these classes are closely related, not only through traditional **AND/OR**-based duality, but also through a kind of link-based duality. These formula classes are examined in light of both types of duality; the resulting insight leads to new compilation techniques and to new target languages.

A brief summary of the basics of NNF formulas is presented in Section 2.1. There is also a short discussion of *negated form* (NF), which is useful as the assumed input language since any logical formula whatsoever can be converted to equivalent negated form in linear time. Path dissolution is described in Section 2.2; greater detail can be found in [13]. Decomposable negation normal form is also described in that section.

The class of pairwise-linked clause sets is examined in Section 3.1. Several new observations are made, and an alternative to the compilation technique of [10] is proposed that does not require subsumption checks. This is generalized to *d-path linked formulas* in Section 3.2. Section 4 focuses on duality relationships: Both traditional and/or based duality and link-based.

Proofs are omitted due to lack of space; they can be found in [16].

2 Linkless Normal Forms

There is growing interest [3, 11, 5, 6, 13, 19, 22] in non-clausal representations of logical formulas. The NNF representation of a formula often has a considerable space advantage, and many CNF formulas can be *factored* into an NNF equivalent that is exponentially smaller. Similar space saving can be realized using structure sharing, for example with tries. On the other hand, there is experimental evidence [13] that factoring a CNF formula can provide dramatic improvements in performance for NNF based systems.

Linkless formulas have several properties desirable for knowledge compilation. In [9], *factored negation normal form* (FNNF) and *ordered factored negation normal form* (OFNNF), both linkless, are introduced. The latter is a canonical representation of *any* logical formula (modulo a given variable ordering of any superset of the set of variables that appear in the formula). The FNNF of a formula can be obtained with Shannon expansion. The dual Shannon expansion and the dual of FNNF are described in Section 3.2.

2.1 Negated Form and Negation Normal Form

A logical formula is said to be in *negated form* (NF) if it uses only three binary connectives, $\wedge, \vee, \Leftrightarrow$, and if all negations are at the atomic level. Negated from is introduced in [9] and is useful as an input form because any logical formula can be converted to negated form in linear time (and space). An NF formula that contains no $\Leftrightarrow$'s is in negation normal form (NNF).

Two literal occurrences are said to be *c-connected* if they are conjoined within the formula (and *d-connected* if they are disjoined within the formula). A *c-path* (*d-path*) is a maximal set of c-connected (d-connected) literal occurrences. A *link* is a complementary pair of c-connected literals.

Related to factoring is the *Shannon expansion* of a formula $\mathcal{G}$ on the atom p, also known as *semantic factoring*; it is defined by the identity, $\mathcal{G} \equiv (p \wedge \mathcal{G}[\mathrm{true}/p]) \vee (\neg p \wedge \mathcal{G}[\mathrm{false}/p])$, where $\mathcal{G}[\beta/p]$ denotes the replacement of all occurrences of atom p by β in $\mathcal{G}$. Observe that any formula $\mathcal{G}$ may be replaced by the formula on the right. In fact, this rule can be applied to any subformula, and, in particular, to the smallest part of the formula containing all occurrences of the variable being factored. The term semantic factoring reflects the fact that all occurrences of the atom p within the subformula under consideration have in effect been 'factored' into one positive and one negative occurrence. Darwiche's *conditioning* operation, the original Prawitz Rule [18], BDDs, and Step 4 in Rule III of the Davis-Putnam procedure [7] are closely related to Shannon expansion.

2.2 Path Dissolution and Decomposable Negation Normal Form

Path dissolution [13] is an inference mechanism that works naturally with formulas in negation normal form. It operates on a link by restructuring the formula so that all paths through the link are deleted. The restructured formula is called the *dissolvent*; the c-paths of the dissolvent are precisely the c-paths of the original formula except those through the activated link. Path dissolution is strongly complete in the sense that any sequence of link activations will eventually terminate, producing a linkless formula called the *full dissolvent*. The paths that remain are models of the original formula. Full dissolvents have been used effectively for computing the prime implicants and implicates of a formula [19, 20]. Path dissolution has advantages over clause-based inference mechanisms, even when the input is in CNF, since CNF can be factored. The time savings is often significant, and the space savings can be dramatic.

Let $\mathbf{atoms}(\mathcal{F})$ denote the atom set of a formula $\mathcal{F}$. An NNF formula $\mathcal{F}$ is said to be in *decomposable negation normal form* (DNNF) if $\mathcal{F}$ satisfies the *decomposability property*: If $\alpha = \alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n$ is a conjunction in $\mathcal{F}$, then $i \neq j$ implies that $\mathbf{atoms}(\alpha_i) \cap \mathbf{atoms}(\alpha_j) = \emptyset$; i.e., no two conjuncts of α share an atom. Observe that a DNNF formula is necessarily linkless since a literal and its complement cannot be conjoined—after all, they share the same atom. The structure of formulas in DNNF is much simpler than the more general NNF. As a result, many operations on DNNF formulas can be performed efficiently. Of course, obtaining DNNF can be expensive, but if this is done once as a preprocessing step, the "expense" can be spread over many queries.

2.3 Testing for Entailment

One common query of a knowledge base $\mathcal{K}$ is the question,"Does $\mathcal{K}$ logically entail a clause C?" A yes answer is equivalent to $(\mathcal{K} \wedge \neg C)$ being unsatisfiable. Both DNNF formulas and full dissolvents, which are linkless, can answer such a query in time linear in the size of the knowledge base[2] For DNNF formulas, this is accomplished by converting $(\mathcal{K} \wedge \neg C)$ to DNNF with a technique called *conditioning* [6]. If the resulting DNNF reduces to empty, then the answer to the query is yes.

Full dissolvents can also determine entailment in linear time. If a knowledge base $\mathcal{K}$ is a full dissolvent, $\mathcal{K}$ contains no links, and all links in $\mathcal{K} \wedge \neg C$ go between $\mathcal{K}$ and $\neg C$; i.e., between $\mathcal{K}$ and a unit. Dissolving on such links strictly decreases the size of the formula. Each operation is no worse than linear in the amount by which the formula shrinks, so the total time required to dissolve away all links is linear in the size of $\mathcal{K}$. The formula that is produced is again linkless, and if this formula is empty, the answer to the query is yes.

There is another approach for answering this query; it is based on Nelson's Theorem, a proof of which can be found in [19]:

Theorem 1. In any non-empty formula $\mathcal{K}$ in which no c-path contains a link, i.e., if $\mathcal{K}$ is a full dissolvent, then every implicate of $\mathcal{K}$ is subsumed by some d-path of $\mathcal{K}$. □

A clause C can then be tested for entailment as follows: First, consider the subformula of $\mathcal{K}$ consisting of all complements of literals of C. Then C is entailed if this subgraph contains a full d-path through $\mathcal{K}$. This computation can be done in time linear in the size of $\mathcal{K}$—for a proof see [15].

3 Pairwise-Linked Formulas

In Section 2, classes of linkless formulas used as target languages for knowledge compilation were described. The opposite approach is considered in this section: Classes of formulas with many links are explored. As we shall see in Section 4, there is a duality to these two approaches.

3.1 Pairwise-Linked Clause Sets

Hai and Jigui introduced pairwise-linked clause sets in [10]. They call them *EPCCL Theories*: *E*ach *P*air (of clauses) *C*ontains *C*omplementary *L*iterals. This class of formulas has a number of nice properties, including the fact that satisfiability can be determined in linear time. The authors provide an elegant proof of this result, included here, with a clever combination of the inclusion/exclusion principle and the observation that every proper subset of the complete matrix, defined below, on m variables is satisfiable.

A *maximal term* on m variables is a clause that contains all m variables (positively or negatively). The *complete matrix* (see [1]) on m variables is the clause set $\mathcal{C}_m$ consisting of all 2^m maximal terms. It is easy to see that $\mathcal{C}_m$ is minimally unsatisfiable; that is,

[2] Linear time is not very impressive: The knowledge base is typically exponential in the size of the original formula.

$\mathcal{C}_m$ is unsatisfiable, but every subset is satisfiable. The authors use their *extension rule*, defined below, to extend any (non-tautological) clause[3] to a logically equivalent set of maximal terms. They observe that a clause set is thus unsatisfiable if and only if extending every clause to maximal terms produces all of $\mathcal{C}_m$, i.e., exactly 2^m clauses. As a result, satisfiability can be reduced to a counting problem. The authors use the inclusion/exclusion principle, stated below, to do this count in linear time on pairwise-linked clause sets.

The extension rule can be defined as follows: Let C be a clause, and let p be an atom not appearing in C. Then the clause set $C' = \{C \vee p, C \vee \bar{p}\}$ is the extension of C with respect to p. Observe that C' is logically equivalent to C. With repeated applications of extension, a set of maximal terms equivalent to any clause can be obtained. If m is the size of maximal terms, then $2^{m-|C|}$ is the size of the equivalent set of maximal terms.

The inclusion/exclusion principle can be stated as follows: Let $P_1, P_2, ..., P_n$ be any collection of finite sets. Then $(*)\ |\cup_{i=1}^{n} P_i| = \sum_{i=1}^{n} |P_i| - \sum_{1 \le i < j \le n} |P_i \cap P_j| + \ldots + (-1)^{n+1} |P_1 \cap P_2 \cap \ldots \cap P_n|$.

Given a set $\mathcal{F} = \{C_1, \ldots, C_n\}$ of clauses (with m variables), let P_i denote the set of maximal terms obtained by extending C_i. Then $\mathcal{F}$ is equivalent to $\bigcup_{i=1}^{n} P_i$ and thus is satisfiable iff $\left|\bigcup_{i=1}^{n} P_i\right| < 2^m$. Observe that, for pairwise-linked clause sets, since the C_i's are pairwise linked, the sets P_i are pairwise disjoint. Thus, while for arbitrary clause sets the inclusion/exclusion principle may be impractical to use, for pairwise-linked clause sets, the formula $(*)$ reduces to $|\cup_{i=1}^{n} P_i| = \sum_{i=1}^{n} |P_i|$.

The satisfiability of a pairwise-linked clause set can thus be determined in linear time by determining whether $\sum_{i=1}^{n} |P_i| = \sum_{i=1}^{n} 2^{m-|C_i|}$ is 2^m. Determining whether a clause is entailed by a pairwise-linked clause set can also be done in polynomial time. If $\mathcal{F}$ is the clause set, and $C = \{p_1, p_2, ..., p_k\}$ is the clause, then $\mathcal{F} \models C$ iff $\mathcal{F} \wedge \neg C$ is unsatisfiable. Since $\neg C$ can be thought of as a set of unit clauses, whether $\mathcal{F} \models C$ can be determined by finding a pairwise-linked clause set that is equivalent to $\mathcal{F} \cup \bigcup_{i=1}^{k} \{p_i\}$. The computation time will be polynomial if the equivalent pairwise-linked clause set can be constructed in polynomial time.

The method used by Hai and Jigui to accomplish this is simple and elegant. If {p} is a unit clause to be added to $\mathcal{F}$, partition $\mathcal{F}$ into three sets of clauses: Let $\mathcal{F}_1$ be the set of clauses containing $\bar{p}$, let $\mathcal{F}_2$ be the set of clauses containing p, and let $\mathcal{F}_3$ be the remaining clauses. The clauses of $\mathcal{F}$ are of course already pairwise linked, so we need only be concerned about links between one clause in $\mathcal{F}$ and the unit clause $\{p\}$. The clauses in $\mathcal{F}_1$ are already linked to $\{p\}$, and $\{p\}$ subsumes each clause in $\mathcal{F}_2$, so they may be deleted. If C is a clause in $\mathcal{F}_3$, then extend C with respect to p, producing $C \cup \{\bar{p}\}$ and $C \cup \{p\}$. The latter is subsumed by p and thus may be deleted. We thus have a pairwise linked clause set that is logically equivalent to $\mathcal{F} \cup \{\{p\}\}$. Observe that the method used to produce this clause set amounts to the following: Add the unit $\{p\}$ to the given pairwise-linked clause set, delete all clauses subsumed by the unit, and add $\bar{p}$ to all clauses not containing $\bar{p}$. This process is clearly linear in the size of the original clause set $\mathcal{F}$ and thus is at most quadratic for adding several unit clauses. The results of this discussion are summarized in the next theorem.

[3] Tautologies cannot be extended to maximal terms because they contain an atom and its negation.

Theorem 2. (Hai and Jigui [10]) If $\mathcal{F}$ is a set of pairwise-linked clauses, then determining whether $\mathcal{F}$ is satisfiable can be done in time linear in the size of $\mathcal{F}$, and whether $\mathcal{F}$ entails a given clause can be determined in polynomial time. □

A note of caution: The extension rule can be used to create an equivalent pairwise-linked set of clauses $\mathcal{F}'$ from any clause set $\mathcal{F}$, and these operations are polynomial in $\mathcal{F}'$. However, $\mathcal{F}'$ may be exponentially larger than $\mathcal{F}$.

Quite a bit more can be said about pairwise-linked clause sets. Recall that a literal occurring in a clause set is said to be *pure* if its complement does not occur in the clause set. It is well known (and very easy to verify) that a clause set is unsatisfiable if and only if the clause set produced by removing all clauses containing pure literals is unsatisfiable. The next theorem may therefore be surprising — see [16] for a proof.

Theorem 3. Let $\mathcal{F}$ be an unsatisfiable set of tautology-free pairwise-linked clauses. Then $\mathcal{F}$ contains no pure literals. □

It is often desirable to work with minimally unsatisfiable clause sets, but, typically, it is not easy to find such a set, even knowing that the clause set is unsatisfiable. The next theorem may therefore also be surprising. (The proof — see [16] — is immediate from observations in [10], but there is also an elegant proof that relies only on first principles.)

Theorem 4. Let $\mathcal{F}$ be an unsatisfiable set of tautology-free pairwise-linked clauses. Then $\mathcal{F}$ is minimally unsatisfiable. □

3.2 DPL Formulas

The class of pairwise-linked formulas discussed in Section 3.1 is restricted to CNF. Not surprisingly, there is a corresponding class of NNF formulas that contains the CNF class as a special case; they are called *d-path linked* (DPL) formulas. Satisfiability of certain DPL formulas can also be determined in polynomial time, and so there is the potential advantage that such an NNF formula may be exponentially smaller than the equivalent pairwise-linked clause set.

An NNF formula $\mathcal{G}$ is said to be d-path linked if every pair of distinct d-paths is linked, and if no d-path contains more than one occurrence of an atom. Note that the latter condition forces each d-path, which is an occurrence set, to correspond exactly to a non-tautological clause in a CNF equivalent of $\mathcal{G}$; i.e., the set of d-paths is an equivalent pairwise-linked clause set.

It may seem that working with DPL formulas is impractical. Determining whether an NNF formula is d-path linked would appear to be as hard as determining whether a clause set is pairwise-linked, and it would appear that few arbitrary NNF formulas are in this class. A DPL formula can be obtained by factoring a pairwise-linked clause set, but then the full cost of producing the clause set must be paid. Nevertheless, the situation is more promising than this.

A compilation algorithm is presented in [10] that requires CNF input and produces a pairwise-linked clause set as output. At the heart of the algorithm is a triply-nested loop; at the innermost level resides a subsumption check. Here, a compilation algorithm is introduced that does not use subsumption and that compiles arbitrary formulas directly into DPL formulas. The key to the algorithm is the *dual Shannon expansion* of a formula $\mathcal{G}$, which is defined to be $\neg SE(\neg\mathcal{G}, p) = (p \vee \mathcal{G}[\mathrm{false}/p) \wedge (\neg p \vee \mathcal{G}[\mathrm{true}/p])$.

The dual of the FNNF operator is a DPL formula called *disjunctive factored negation normal form*; it is defined using duality by D-FNNF$(L, \mathcal{F}) = \neg$ FNNF$(L, \neg\mathcal{F})$.

If p_i is chosen so that i is maximal, the result is called ordered D-FNNF and denoted D-OFNNF. The remarks about FNNF apply in a straightforward but dual manner to these formulas. Thus a knowledge base will compile to a formula that can be regarded as a binary tree with root 1, and whose leaves do not have leaf siblings. Branches of D-FNNF (FNNF) trees correspond to d-paths (c-paths). Just as FNNF formulas have no c-links, in D-FNNF there are no d-links. But it is evident from the definition that the d-paths are pairwise linked: The left and right subtrees are rooted at p and $\neg p$, respectively, and so all branches within the left subtree contain p, and all branches within the right subtree contain $\neg p$.[4]

If a knowledge base $\mathcal{K}$ is compiled by the D-OFNNF operator, the question, Given clause $C = \{p_1, \ldots, p_n\}$, does $\mathcal{K}$ entail C, i.e., is $\mathcal{K} \wedge \neg C$ unsatisfiable? can be answered as follows: Substitute 0 for p_i and 1 for $\neg p_i$, $1 \leq i \leq n$, throughout D-OFNNF$(L, \mathcal{K})$ and apply the $SIMP$ rules. If the query is entailed by $\mathcal{K}$, the tree simplifies to 0. This process is linear in D-OFNNF$(L, \mathcal{K})$. It is interesting to note that for an arbitrary entailment $\mathcal{F} \models C$, determining whether $\mathcal{F} \wedge \neg C$ is unsatisfiable cannot done by substituting constants; indeed, this is an $\mathcal{NP}$-complete problem. But for a D-OFNNF tree, simplification alone is sufficient because the D-OFNNF of an unsatisfiable formula is a root labeled 0.[5]

It is not always necessary to perform all the substitutions and simplifications to determine whether the query is entailed. Suppose C is an implicate of $\mathcal{K}$—i.e., suppose $\mathcal{K} \models C$. Then $\neg C \models \neg\mathcal{K}$, so an assignment making all literals of C false must falsify $\mathcal{K}$. This viewpoint is useful because, just as branches of an OFNNF formula represent satisfying interpretations, D-OFNNF branches represent falsifying ones; i.e., setting all literals on a branch to false falsifies the formula. Thus, the set of literals labeling a branch is an implicate of the formula. Recall that an implicate is *prime* if it is minimal in the sense that no proper subset is also an implicate.

Lemma 1. Let $\mathcal{F}$ be an arbitrary logical formula, and let $\mathcal{I}_P$ be a clause containing q such that $\mathcal{F} \wedge \neg(\mathcal{I}_P - \{q\})$ is not unsatisfiable. Then $\mathcal{I}_P$ is a prime implicate of $\mathcal{F}$ iff $\mathcal{I}_P - \{q\}$ is a prime implicate of $\mathcal{F} \wedge \neg q$. □

Theorem 5. Let the nodes on branch B in D-OFNNF$(L, \mathcal{K})$ be labeled $q_1, \ldots, q_m$, in that order, so that q_m is the leaf. Then there is a unique subset $\mathcal{I}_P$ of $\{q_1, \ldots, q_m\}$ that is a prime implicate of $\mathcal{K}$. Moreover, $\mathcal{I}_P$ contains q_m. □

Theorem 5 provides another way to test a clause $C = \{p_1, \ldots, p_n\}$ for entailment. Suppose the tree is stored so that each node has a bit vector indicating the branch leading to it from the root. The occurrences of p_n in the tree may be scanned, and each node's vector can be examined to determine whether $p_1, \ldots, p_{n-1}$ are on that branch. Such branches are *consistent with C*. If a consistent branch is found in which p_n is not a leaf, then C is not entailed by $\mathcal{K}$ (since the assignment falsifying every literal on this branch falsifies C but not $\mathcal{K}$). If no such branch is found, substitute 0 for p_n and 1 for $\neg p_n$, apply $SIMP$, and repeat for p_{n-1}. Queries for which the answer is yes require as

[4] In a dual manner, an FNNF formula represents a pairwise d-linked DNF clause set.

[5] But let us not forget that building D-OFNNF may be expensive.

much computation time with either approach, but if the answer is no, this method may terminate more quickly.

4 Dualities

That duality should arise in a variety of ways in the study of propositional languages is hardly surprising. But in this setting it is so pervasive that a brief synopsis may be illuminating.

Perhaps the most familiar duality is that of CNF and DNF. The latter is (conjunctive) link-free, but typically has many d-links; the former is free of d-links but may have many c-links. From Nelson's Theorem [17], we know that the prime implicants of a tautology-free CNF formula are present as non-contradictory c-paths, and the prime implicates of a contradiction-free DNF formula are present as non-tautological d-paths. It turns out that producing implicants and implicates syntactically is the result not of CNF/DNF per se, but of the absence of links, which is a side effect of converting to CNF or to DNF.

Computing the full dissolvent of a formula removes links without producing DNF; similarly, the disjunctive dual of dissolution produces a formula without d-links. This leads to a version of Nelson's Theorem based only on the absence of links [19].

The work of Hai and Jigui [10] pointed towards an additional layer of duality—namely that implicates can be extracted not only from c-linkless formulas, but also from pairwise-linked clause sets. Thus entailment testing is facilitated either by removing links or by adding enough of them. An immediate consequence is that a DNF clause can be polynomially[6] checked for being a prime implicant of a pairwise d-linked DNF formula.

In [15, 9], Shannon expansion is used to compile to DNNF and to FNNF; these target languages are regarded essentially as c-linkless. But of course FNNF is an NNF generalization of a pairwise d-linked DNF formula, and this realization led the authors to develop D-FNNF, an NNF generalization of a pairwise c-linked CNF formula, using the dual of Shannon expansion. It is by now obvious that the prime implicant status of literal conjunctions can be conveniently tested using OFNNF.

Nelson's Theorem provides a duality between the type of link that is absent and the type of query that is easily answered. The additional duality between many links and the absence of links induces a symmetry: Both prime implicants and prime implicates can be tested with both pairwise-linked formulas and with link-free formulas. This link-based duality — link-free versus d-path linked — appears to be heretofore unnoticed.

References

1. Bibel, W., Tautology testing with a generalized matrix method, *Theoretical Computer Science* **8** (1979), 31–44.
2. Bryant, R. E., Symbolic Boolean manipulation with ordered binary decision diagrams, *ACM Computing Surveys* **24, 3** (1992), 293–318.

[6] We repeat the caveat that the pairwise-linked formula may be large.

3. Cadoli, M., and Donini, F. M., A survey on knowledge compilation, *AI Communications* **10** (1997), 137–150.
4. Chatalic, P. and Simon, L., Zres: The old Davis-Putnam procedure meets ZBDDs. *Proc. CADE'17*, David McAllester, ed., LNAI 1831 (June 2000), Springer-Verlag, 449-454.
5. Darwiche, A., Compiling devices: A structure-based approach, Proc. *International Conference on Principles of Knowledge Representation and Reasoning (KR98)*, Morgan-Kaufmann, San Francisco (1998), 156–166.
6. Darwiche, A., Decomposable negation normal form, *J.ACM* **48,4** (2001), 608–647.
7. Davis, M. and Putnam, H. A computing procedure for quantification theory. *J.ACM* **7**, 1960, 201–215.
8. Forbus, K.D. and de Kleer, J., *Building Problem Solvers*, MIT Press, Mass. (1993).
9. Hähnle, R., Murray, N.V., and Rosenthal, E. Normal forms for knowledge compilation, *Proc ISMIS 2005*, Saratoga Springs, NY, *LNAI*, Springer-Verlag, to appear.
10. Hai, L. and Jigui, S., Knowledge compilation using the extension rule, *J. Automated Reasoning*, 32(2), 93-102, 2004.
11. Henocque, L., The prime normal form of boolean formulas. Submitted. Preliminary version available as a technical report at http://www.Isis.org/fiche.php?id=74&page=.
12. Marquis, P., Knowledge compilation using theory prime implicates, Proc. *International Joint Conference on AI (IJCAI)* (1995), Morgan-Kaufmann, San Mateo, California, 837-843.
13. Murray, N.V., and Rosenthal, E. Dissolution: Making paths vanish. *J.ACM* **40,3** (July 1993), 504–535.
14. Murray, N.V., and Rosenthal, E. On the relative merits of path dissolution and the method of analytic tableaux, *Theoretical Computer Science* **131** (1994), 1–28.
15. Murray, N.V. and Rosenthal E., Tableaux, path dissolution, and decomposable negation normal form for knowledge compilation, *Proc TABLEAUX 2003*, Rome, Italy, *LNAI* 2796 (M. Mayer and F. Pirri, Eds.), Springer-Verlag, 165-180.
16. Murray, N.V., and Rosenthal, E. Knowledge compilation and duality, Technical Report SUNYA-CS-04-07, Dept of Computer Science, SUNY Albany, October, 2004.
17. Nelson, R.J., Simplest normal truth functions, *J. of Symbolic Logic* **20**, 105-108 (1955).
18. Prawitz, D. A proof procedure with matrix reduction. *Lecture Notes in Mathematics* **125**, Springer-Verlag, 1970, 207–213.
19. Ramesh, A., Becker, G. and Murray, N.V. CNF and DNF considered harmful for computing prime implicants/implicates. *J. of Automated Reasoning* **18**,3 (1997), Kluwer, 337–356.
20. Ramesh, A. and Murray, N.V. An application of non-clausal deduction in diagnosis. *Expert Systems with Applications* **12**,1 (1997), 119-126.
21. Selman, B., and Kautz, H., Knowledge compilation and theory approximation, *J.ACM* **43,2** (1996), 193-224.
22. Walsh, T., Non-clausal reasoning, *Workshop on Disproving Non-Theorems, Non-Validity, and Non-Provability*, IJCAR 2004, Cork, Ireland.