Five Isomorphic Boolean Theories and Four Equational Decision Procedures

Camilo Rocha
José Meseguer

February, 2007

Technical Report
UIUCDCS-R-2007-2818

Formal Methods and Declarative Languages Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N Goodwin Ave
Urbana, IL 61801

And, finally, when the symbol manipulation would become too labor-intensive, computing science could provide the tools for mechanical assistance. In short, the world of computing became Leibniz's home; that it was my home as well was my luck.

E.W. Dijkstra, "Under the spell of Leibniz's dream"
Information Processing Letters, Vol.77, 2001.

Table of Contents

List of Figures

Abstract

We present four equational theories that are isomorphic to the traditional Boolean theory and show that each one of them gives rise to a canonical rewrite system modulo associativity, thus providing four decision procedures for propositional logic. The four theories come in two pairs of isomorphic dual theories. The first pair corresponds to J. Hsiang's rewrite system for the theory of Boolean rings, and to a rewrite system we propose for Dijkstra-Scholten propositional logic. The second pair uses the same Boolean operators as the previous pair but in a "twisted" fashion. These procedures, when run on a high performance rewrite engine, are quite efficient, but can be further sped up by the use of *optimizing equations* that perform obvious simplifications in the input expression before a decision procedure is invoked. Based on their Maude implementation, we present experimental results comparing the performance of the different procedures, and showing that they outperform a DPLL(T)-based SAT-solver.

Extended Abstract

We present four equational theories that are isomorphic to the traditional Boolean theory and show that each one of these gives rise to a canonical rewrite system (modulo associativity and commutativity of some Boolean operators), thus solving the word problem of the theory. The four theories come in pairs of isomorphic dual theories. The first pair exploits the known connection between Boolean algebras and Boolean rings. Its two dual theories, respectively, coincide with the well-known equational axiomatization of J. Hsiang, and with the axiomatization of propositional logic of E.W. Dijkstra and C.S. Scholten. The second pair, for which we are not aware of previous equational formulations, uses the same Boolean operators as the previous pair, but is not obtained from the connection between Boolean algebras and Boolean rings. We also introduce a generic technique to improve the efficiency of the executable specifications of the four decision procedures, in which *optimizing equations* are used, prior to any evaluation of the decision procedure, to 'pre-process' the input proposition by first 'replacing variables with constants' and then simplifying subexpressions of the resulting expression in the presence of annihilators for some operators. In this paper we use the Maude system, which is an efficient and general implementation of equational logic, to specify the decision procedures and mechanize their execution. Moreover, results from different experiments involving the executable specifications of the decision procedures are presented and analyzed. The first experiment compares the four canonical systems in terms of time and number of simplification steps needed to bring arbitrary propositional expressions to their canonical forms. In the second experiment, we exercise the decision procedures enabling the optimizing equations and compare their behavior with the previous ones. In the third and last experiment, the decision procedures are compared together with a DPLL(T) SAT solver (also specified in Maude) when solving the satisfiability problem for arbitrarily formed propositional expressions.

1 Introduction

The key challege in (semi-)automated deduction is *scaling up*. For the large proof efforts involved in nontoy mathematical and system verification proofs it is essential to raise the level of abstraction, so that the person performing the proofs can delegate large chunks of the effort to automated proof assistants, and can guide the entire effort, like a good general, at the strategic level by marshaling such automated "troops." This need is widely felt, and approaches to meet it take different guises, such as the growing support for decision procedures, the autarkic/skeptical distinction between proofs and computations [2], and the so-called "deduction modulo" approach [6], which, as shown by Viry [23], is just the use of rewriting logic as a logical framework [19], so that one decomposes the proof system as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, with the automated components consisting of a set E of confluent and terminating equations (often modulo some axioms), and R the high-level rules that are applied *modulo* E.

Our original motivation for this work is as part of a bigger effort to mechanize the Dijkstra-Scholten logic [5]. This logic has been shown by Dijkstra and Scholten to be very useful in program correctness proofs in the Dijkstra style, and has attracted a substantial following, including textbooks such as [10]. It has the same expressive power as standard first-order logic [18]; and includes an interesting propositional fragment ([11]). However, to the best of our knowledge, due perhaps to the preferential option for proofs by Montblanc fountain pen and paper in the Dijkstra school, this logic has not yet been mechanized. The obvious approach to obtain a scalable mechanization is to specify it as a rewrite theory $\mathcal{R}_{\rm DS} = (\Sigma_{\rm DS}, E_{\rm DS}, R_{\rm DS})$, where $E_{\rm DS}$ includes a *decision procedure*, in the form of confluent and terminating rewrite rules, for Dijkstra-Scholten propositional logic. We have indeed developed an axiomatization $\mathcal{R}_{\rm DS}$ in exactly this way, and have implemented it in Maude, for *monadic* first-order Dijkstra-Scholten logic [21]. This paper provides both the foundations and the experimental performance evaluation for our decision procedure for Dijkstra-Scholten propositional logic.

However, we observed that Dijkstra-Scholten propositional logic, which uses equivalence ($\equiv$) and disjunction ($\vee$), is the exact *dual* of another well-known Boolean axiomatization going back to Stone, namely Boolean *rings* [14, 22], which uses exclusive or ($\oplus$) and conjunction ($\wedge$). The Boolean ring axiomatization has an important history in rewriting-based automated deduction. Finding a good axiomatization of Boolean algebra by confluent and terminating rewrite rules was a problem that eluded researchers for quite some time. The breakthrough came in the early 1980s with J. Hsiang's thesis [13], who gave a confluent and terminating set of equations for the theory of Boolean rings *modulo* associativity and commutativity. Several inference systems for first-order logic based on the above-mentioned idea of a rewrite theory $\mathcal{R}_{\rm BR} = (\Sigma_{\rm BR}, E_{\rm BR}, R_{\rm BR})$ in which the equations for Hsiang's propositional decision procedure are in $E_{\rm BR}$, were then given, such as Hsiang's refutation rewrite-based *N-strategy* method, [15], and the Bachmair-Dershowitz first-order inference system [1]. Furthermore, A. Foret extended Hsiang's system to give a complete term rewriting system for

the K, Q, T and S5 modal propositional logics [8]. Hsiang's rewrite system was also adopted for the `BOOL` module in both OBJ3 [9], and in Maude [4].

There was, however, and important performance obstacle for Hsiang's decision procedure, due to the fact of its being based on associative-commutative (AC) matching, which is known to be an NP-complete problem [3]. For example, some uses of OBJ3 in hardware verification theorem proving found a major bottleneck in the Boolean simplification part supported by Hsiang's algorithm. A second breakthrough came with Steven Eker's stunning algorithmic solution to the AC matching problem for frequently occurring AC matching patterns based on red-black trees and implemented in Maude [7]. For such patterns, although the general problem remains NP-complete, it is often possible to find a match in time proportional to the *logarithm* of the subject term. So, the situation in terms of implementation algorithms has drastically improved, but as far as we know has never been experimentally evaluated. In fact, this matter raises interesting questions such as the following: are SAT-solvers the only game in town for propositional logic? Can rewriting-based methods compete with them? The experimental performance evaluation results we present in this paper indicate that, when implemented in Maude, both Hsiang's procedure, and our rewriting-based decision procedure for Dijkstra-Scholten propositional logic *outperform* by two orders of magnitude a DPLL(T)-based SAT-solver developed by Joe Hendrix as a component of Maude's inductive theorem prover.

Of course, although dual, *both* Hsiang's procedure and our decision procedure for Dijkstra-Scholten propositional logic stand on their own and can support different axiomatizations of first-order logic. The point is that the particular syntax of the *canonical forms* provided by each procedure matters a grear deal in a "deduction modulo" approach. For example, somebody used to proofs of program correcteness in the Dijkstra-Scholten style who wants to use the rewrite theory $\mathcal{R}_{\mathrm{DS}} = (\Sigma_{\mathrm{DS}}, E_{\mathrm{DS}}, R_{\mathrm{DS}})$ as a proof assistant will heavily rely on having formulas expressed in terms of the connectives $\equiv$ and $\vee$, and would have a very hard time reasoning with formulas expressed in terms of the dual connectives $\oplus$ and $\wedge$, which is what a Boolean-ring-based rewrite theory $\mathcal{R}_{\mathrm{BR}} = (\Sigma_{\mathrm{BR}}, E_{\mathrm{BR}}, R_{\mathrm{BR}})$ would provide. The opposite would of course happen for a user accustomed to think in terms of the $\oplus$ and $\wedge$ connectives. Since diversity and multiculturalisms and present-day truisms, we have thought about providing even more options for diverse classes of users. What about users who might prefer an axiomatization in terms of $\equiv$ and $\wedge$? Or users who might instead prefer the dual connectives $\oplus$ and $\vee$? In this work we show that those users can also be happy, by providing sets of confluent and terminating equations modulo AC for both of those syntax choices. This additional pair of dual decision procedures have also good performance but, as it turns out from our evaluation, not as good as Hsiang's and our Dijkstra-Scholten procedures.

The first mater we address is the *correctness* of all these procedures. This can be naturally decomposed into two substasks; (i) their *mathematical* correctness (studied in Section 3), which amounts to proving that they are all *isomorphic* to the standard theory of Boolean algebras; and (ii) their *operational* correctness

(studied in Section 4), which involves checking that they are all indeed confluent and terminating modulo AC. What we mean by a theory *isomorphism* is not an entirely trivial matter, so we spell out all the mathematical details in Section 2. It turns out that, from the performance point of view, we can do even better, gaining an additional 20–30-fold speedup, by performing some obvious simplifications on the input Boolean expression *before* we invoke any of the four decision procedures. Our optimizing equations, as well the mathematical proof of their soundness, are discussed in Section 5. Our experimental performance evaluation results are presented in Section 6. We present some final conclusions in Section 7.

2 Theories, Morphisms, and Definitional Extensions

This section gathers basic notions on equational theories, theory morphisms, and definitional extensions that are needed in the rest of the paper. Although the subject matter is well-known, there are some technical points that may not be so well-known. For example, the usual notion of a theory morphisms as a signature morphism has to be qualified in two important ways: (i) "signature morphims" are generalized, so that they need not map basic operations to basic operations, but can map basic operations to *terms*; and (ii) a theory morphism is *not* a signature morphism, but instead an *equivalence class* of signature morphisms. We give also some useful results about definitional extensions that we have found very helpful in cutting down the amount of things to be cheked, and that we will make use of later in the paper. Although *nihil novum sub sole*, the reader may find this background section of some independent interest, besides its use in subsequent sections.

Since all the Boolean theories we shall consider are unsorted, we give the whole treatment in the, simpler, unsorted setting. All ideas, however, extend naturally to typed settings. A *signature* Σ, therefore, is a countable family of sets of function symbols $\Sigma = \{\Sigma_n\}_{n\in\mathbb{N}}$. An equational *theory*[1] is a pair (Σ, E), with Σ a signature, and E a set of Σ-equations, that is, formal equalities of the form $t = t'$, with $t, t' \in T_\Sigma(X)$, where $T_\Sigma(X)$ denotes the free Σ-algebra on the set X of variables, which we assume throughout to be the countable set $X = \{x_n \mid n \in \mathbb{N} \ \wedge \ n > 0\}$. An equational theory (Σ, E) defines the full subcategory $\mathbf{Alg}_{(\Sigma,E)}$ of the category $\mathbf{Alg}_\Sigma$ of all Σ-algebras determined by all those algebras that satisfy the equations E.

Definition 1. *A* signature morphism $H : \Sigma \longrightarrow \Sigma'$ *is an assignment, for each* $n \in \mathbb{N}$*, to each* $f \in \Sigma_n$ *of a term* $H(f) \in T_{\Sigma'}(X)$ *with* $vars(H(f)) \subseteq \{x_1, \ldots, x_n\}$*, where* $vars(t)$ *denotes the set of variables occurring in term* t*.*

[1] More precisely, this should be called a theory *presentation*. However, since the notion of isomorphism we define will make isomorphic not only all equivalent presentations for the same Σ, but also all equivalent presentations with different signatures, this *abus de langage* will hardly matter.

Note that H gives as a "view" of each Σ'-algebra A as a Σ-algebra $A|_H$, just by interpreting on A each operation $f \in \Sigma_n$ by means of the "derived operation" $H(f)$, where there is no ambiguity about the order of the arguments thanks to the linear order in X. Indeed, any signature morphism defines a functor $\_|_H : \mathbf{Alg}_{\Sigma'} \longrightarrow \mathbf{Alg}_{\Sigma}$.

Definition 2. *Given signature morphisms $H : \Sigma \longrightarrow \Sigma'$ and $G : \Sigma' \longrightarrow \Sigma''$ we can compose them to obtain a signature morphism $G \circ H : \Sigma \longrightarrow \Sigma''$ as follows: for each $f \in \Sigma_n$ we have $(G \circ H)(f) = \widehat{G}(H(f))$, where $\widehat{G} : T_{\Sigma'}(X) \longrightarrow T_{\Sigma''}(X)|_G$ denotes the unique Σ'-homomorphism leaving the variables X unchanged.*

It is then easy to check that composition is associative, the assignment $f \mapsto f(x_1, \ldots, x_n)$ is the identity morphism for Σ, and we have a category **Sign** of signatures and signature morphisms.

Definition 3. *A* pre-theory morphism *$H : (\Sigma, E) \longrightarrow (\Sigma', E')$ is a signature morphism $H : \Sigma \longrightarrow \Sigma'$ such that $E' \vdash \widehat{H}(E)$, where $\vdash$ denotes the equational provability relation, and $\widehat{H}$ is extended to equations in the obvious way. We say that two pre-theory morphisms $H, H' : (\Sigma, E) \longrightarrow (\Sigma', E')$ are* equivalent*, denoted $H \equiv H'$, iff for each $n \in \mathbb{N}$, and each $f \in \Sigma_n$ we have, $E' \vdash H(f) = H'(f)$. It is easy to check that this is indeed an equivalence relation. We denote each equivalence class by $[H]$, and call such an equivalence class a* theory morphism *from (Σ, E) to (Σ', E').*

Such equivalence relation is clearly a *congruence* for composition of pre-theory morphisms as signature morphisms. That is, if we have pre-theory morphisms $H, H' : (\Sigma, E) \longrightarrow (\Sigma', E')$, and $G, G' : (\Sigma', E') \longrightarrow (\Sigma'', E'')$, with $H \equiv H'$, and $G \equiv G'$, then $G \circ H \equiv G' \circ H'$. Therefore, we can *compose* theory morphisms by the rule, $[G] \circ [H] = [G \circ H]$. This defines a category **Th**, with theories as *objects* and theory morphisms as *morphisms*. It can, furthermore, be shown that **Th** is equivalent to the category of Lawvere theories [17]. We will be particularly interested in theory *isomorphisms*, so it may be worthwhile to "unpack" what they are. $[H] : (\Sigma, E) \longrightarrow (\Sigma', E')$ will be an isomorphism iff there is a $[H^{-1}] : (\Sigma', E') \longrightarrow (\Sigma, E)$ such that $[H^{-1}] \circ [H] = 1_{(\Sigma,E)}$ and $[H] \circ [H^{-1}] = 1_{(\Sigma',E')}$; iff for each $n \in \mathbb{N}$ and each $f \in \Sigma_n$ and $f' \in \Sigma'_n$ we have: (i) $E \vdash f(x_1, \ldots, x_n) = (H^{-1} \circ H)(f)$; and (ii) $E' \vdash f'(x_1, \ldots, x_n) = (H \circ H^{-1})(f')$.

One important, model-theoretic point to notice is that a theory morphism $[H] : (\Sigma, E) \longrightarrow (\Sigma', E')$ induces a functor $\_|_{[H]} : \mathbf{Alg}_{(\Sigma',E')} \longrightarrow \mathbf{Alg}_{(\Sigma,E)}$, which is just the restiction of the functor $\_|_H : \mathbf{Alg}_{\Sigma'} \longrightarrow \mathbf{Alg}_{\Sigma}$. Furthermore, since the assignment $[H] \mapsto \_|_{[H]}$ is itself a contravariant functor (see [17]), if $[H]$ is a theory isomorphism, then $\_|_{[H]} : \mathbf{Alg}_{(\Sigma',E')} \longrightarrow \mathbf{Alg}_{(\Sigma,E)}$ is an isomorphism of categories that preserves the sets and functions underlying the algebras an homomorphisms. Lawvere's beautiful *Strucuture-Semantics Adjointness Theorem* [17] proves also the opposite direction: any isomorphism of categories $\alpha : \mathbf{Alg}_{(\Sigma',E')} \cong \mathbf{Alg}_{(\Sigma,E)}$ that preserves the sets and functions underlying the

algebras an homomorphisms is of the form $\alpha = \_|_{[H]}$ for some theory isomorphism $[H]$.

Therefore, theory isomorphism give us a *presentation-independent* view of axiomatic classes of algebras. For example, the theory of groups can be presented with many different signatures and sets of axioms. What all these presentations have in common is precisely that they are *isomorphic* theories in the precise sense defined above. In this paper we will be interested in theory isomorphisms for the theory of Boolean algebra. A paradigmatic example of a theory isomorphism in this case, in fact one of the isomorphisms we shall consider, is the Stone isomorphism between the theory of Boolean algebras and that of Boolean *rings*.

2.1 Definitional Extensions

Given two signatures Σ and Σ', we define their union $\Sigma \cup \Sigma'$, resp. intersection $\Sigma \cap \Sigma'$, resp. difference $\Sigma - \Sigma'$, in the obvious way: for each $n \in \mathbb{N}$ $(\Sigma \cup \Sigma')_n = \Sigma_n \cup \Sigma'_n$, $(\Sigma \cap \Sigma')_n = \Sigma_n \cap \Sigma'_n$, and $(\Sigma - \Sigma')_n = \Sigma_n - \Sigma'_n$.

Definition 4. *We call a theory morphism $[H] : (\Sigma, E) \longrightarrow (\Sigma', E')$* unambiguous *iff $[H]$ restricted to $\Sigma \cap \Sigma'$ is the identity.*

This captures the intuitive, and frequently occurring situation where $[H]$ does not change the meaning of *shared symbols*: only the function symbols that Σ does not share with Σ' are given a new interpretation by $[H]$.

Lemma 1. *If $(\Sigma, E) \xrightarrow{[H]} (\Sigma', E') \xrightarrow{[G]} (\Sigma'', E'')$ are unambiguous theory morphisms such that $\Sigma \cap \Sigma'' \subseteq \Sigma'$, then $[G \circ H]$ is unambiguous.*

Proof. Let $f \in \Sigma$. If $f \notin \Sigma \cap \Sigma''$, then the Lemma trivially holds. Assume $f \in \Sigma \cap \Sigma''$. Since $\Sigma \cap \Sigma'' \subseteq \Sigma''$, $f \in \Sigma \cap \Sigma' \cap \Sigma''$. Moreover, because $[H]$ and $[G]$ are unambiguous, $[G]([H](f)) = [G](f) = f$. Hence, $[G] \circ [H] = [G \circ H]$ is unambiguous.

Definition 5. *Given an unambiguous theory morphism $[H] : (\Sigma, E) \longrightarrow (\Sigma', E')$, the* definitional extension *of (Σ', E')* along *$[H]$, denoted $(\Sigma', E')^{[H]}$, is the theory $(\Sigma', E')^{[H]} = (\Sigma \cup \Sigma', E' \cup \Delta_{[H]})$, where $\Delta_{[H]} = \{f(x_1, \ldots, x_n) = H(f) \mid f \in \Sigma - \Sigma'\}$. It is trivial to check that the obvious identity inclusion $(\Sigma', E') \hookrightarrow (\Sigma', E')^{[H]}$ is a theory isomorphism with inverse the identity on Σ', and mapping each $f \in \Sigma - \Sigma'$ to $H(f)$.*

If we have two unambiguous theory morphisms

$$(\Sigma, E) \xrightarrow{[H]} (\Sigma', E') \xrightarrow{[G]} (\Sigma'', E'')$$

such that $\Sigma \cap \Sigma'' \subseteq \Sigma'$, then it is easy to prove using the above lemma that we can iterate the definitional extension process to form a "tower" of definitional extensions

$$(\Sigma'', E'') \hookrightarrow (\Sigma'', E'')^{[G]} \hookrightarrow (\Sigma', E')^{[G] \otimes [H]}$$

where $(\Sigma', E')^{[G]\otimes[H]} = (\Sigma \cup \Sigma' \cup \Sigma'', E'' \cup \Delta_{[G]} \cup \widehat{G}(\Delta_{[H]}))$. In particular we get in this way a theory isomorphism $(\Sigma'', E'') \cong (\Sigma', E')^{[G]\otimes[H]}$. This construction will be technically useful for the Boolean decision procedures we will present, based on theory isomorphisms; because it will automatically justify the correctness of each decision procedure simultaneously supporting *all* the Boolean operations of the different signatures involved.

3 Five Isomorphic Boolean Theories

In this section we present five equational theories, one of them the traditional Boolean theory. We prove that the other four are isomorphic to it. We structure each of these theories in the form $(\Sigma, E \uplus A)$, where we decompose the equations into a set A of commonly occurring axioms, such as associativity and commutativity of some operators, and a remaining set of equations E.

The axiomatization of the traditional Boolean theory T_{BOOL} is that of a complemented distributive lattice.

Definition 6. *The equational theory* $T_{\mathrm{BOOL}} = (\Sigma_{\mathrm{BOOL}}, E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}})$ *is given by:*

$$
\begin{aligned}
\Sigma_{\mathrm{BOOL}} &= \{\mathsf{T}^{(0)}, \mathsf{F}^{(0)}, \neg^{(1)}, \wedge^{(2)}, \vee^{(2)}\} \\
A_{\mathrm{BOOL}} &= \{P \wedge (Q \wedge R) = (P \wedge Q) \wedge R\ ,\ P \wedge Q = Q \wedge P\ , \\
&\qquad P \vee (Q \vee R) = (P \vee Q) \vee R\ ,\ P \vee Q = Q \vee P\} \\
E_{\mathrm{BOOL}} &= \{P \wedge P = P\ ,\ P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)\ , \\
&\qquad P \vee P = P\ ,\ P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)\ , \\
&\qquad P \wedge (P \vee Q) = P\ ,\ P \vee (P \wedge Q) = P\ , \\
&\qquad P \wedge \neg P = \mathsf{F}\ ,\ P \vee \neg P = \mathsf{T}\ \}\ .
\end{aligned}
$$

The constants T and F denote the obvious true and false constant symbols; the other function symbols in Σ_{BOOL} have the following intended meaning and are listed by decreasing binding power: $\neg$ denotes negation while $\vee$ and $\wedge$ denote disjunction and conjunction, respectively. The axioms in A_{BOOL} express the associativity and commutativity properties (AC) of the binary operators in Σ_{BOOL}. The set of axioms E_{BOOL} define both $\wedge$ and $\vee$ to be idempotent, to distribute over each other and to follow the absorption laws. The last two equations in E_{BOOL} are the well-known laws of complements, the first being the definition of contradiction and the second that of the excluded middle.

In the rest of this paper we rely on AC properties and precedence of operators to omit unnecessary parentheses. For example, thanks to this convention, the expression $P \vee (Q \vee (R \wedge S))$ can be written as $P \vee Q \vee (R \wedge S)$. Furthermore, due to the commutativity property of $\wedge$ and $\vee$, the latter expression can as well be written as $Q \vee (S \wedge R) \vee P$.

We introduce the remaining four equational theories, namely T_{DS}, T_{BR}, $T_{\wedge/\equiv}$ and $T_{\vee/\oplus}$, respectively. The theory T_{DS} is our axiomatization as a set of confluent and terminating equations modulo AC of the Dijkstra-Scholten propositional logic [5]. The theory T_{BR} is the theory of Boolean rings and is based on the isomorphism between Boolean algebras and Boolean rings discovered by

M.H.Stone [14, 22]. As a rewrite system, T_{BR} was proposed by J. Hsiang [13] in the 1980's as a decision procedure for propositional logic. We are not aware of earlier equational presentations of $T_{\wedge/\equiv}$ and $T_{\vee/\oplus}$, so we use their main function symbols as acronyms.

Definition 7. *The equational theories* $T_{\mathrm{DS}} = (\Sigma_{\mathrm{DS}}, E_{\mathrm{DS}} \uplus A_{\mathrm{DS}})$, $T_{\mathrm{BR}} = (\Sigma_{\mathrm{BR}}, E_{\mathrm{BR}} \uplus A_{\mathrm{BR}})$, $T_{\wedge/\equiv} = (\Sigma_{\wedge/\equiv}, E_{\wedge/\equiv} \uplus A_{\wedge/\equiv})$ *and* $T_{\vee/\oplus} = (\Sigma_{\vee/\oplus}, E_{\vee/\oplus} \uplus A_{\vee/\oplus})$ *are defined as follows:*

$$
\begin{aligned}
\Sigma_{\mathrm{DS}} &= \{\mathsf{T}^{(0)}, \mathsf{F}^{(0)}, \vee^{(2)}, \equiv^{(2)}\} \\
A_{\mathrm{DS}} &= \{P \equiv (Q \equiv R) = (P \equiv Q) \equiv R\,,\ P \equiv Q = Q \equiv P\,, \\
&\qquad P \vee (Q \vee R) = (P \vee Q) \vee R\,,\ P \vee Q = Q \vee P\} \\
E_{\mathrm{DS}} &= \{P \equiv \mathsf{T} = P\,,\ P \equiv P = \mathsf{T}\,,\ P \vee \mathsf{T} = \mathsf{T}\,,\ P \vee \mathsf{F} = P\,,\ P \vee P = P\,, \\
&\qquad P \vee (Q \equiv R) = (P \vee Q) \equiv (P \vee R)\}\,,
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_{\mathrm{BR}} &= \{\mathsf{T}^{(0)}, \mathsf{F}^{(0)}, \wedge^{(2)}, \oplus^{(2)}\} \\
A_{\mathrm{BR}} &= \{P \oplus (Q \oplus R) = (P \oplus Q) \oplus R\,,\ P \oplus Q = Q \oplus P\} \\
&\qquad P \wedge (Q \wedge R) = (P \wedge Q) \wedge R\,,\ P \wedge Q = Q \wedge P\} \\
E_{\mathrm{BR}} &= \{P \oplus \mathsf{F} = P\,,\ P \oplus P = \mathsf{F}\,,\ P \wedge \mathsf{F} = \mathsf{F}\,,\ P \wedge \mathsf{T} = P\,,\ P \wedge P = P\,, \\
&\qquad P \wedge (Q \oplus R) = (P \wedge Q) \oplus (P \wedge R)\}\,,
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_{\wedge/\equiv} &= \{\mathsf{T}^{(0)}, \mathsf{F}^{(0)}, \wedge^{(2)}, \equiv^{(2)}\} \\
A_{\wedge/\equiv} &= \{P \equiv (Q \equiv R) = (P \equiv Q) \equiv R\,,\ P \equiv Q = Q \equiv P\,, \\
&\qquad P \wedge (Q \wedge R) = (P \wedge Q) \wedge R\,,\ P \wedge Q = Q \wedge P\} \\
E_{\wedge/\equiv} &= \{P \equiv \mathsf{T} = P\,,\ P \equiv P = \mathsf{T}\,,\ P \wedge \mathsf{T} = P\,,\ P \wedge \mathsf{F} = \mathsf{F}\,,\ P \wedge P = P\,, \\
&\qquad P \wedge (Q \equiv R) = (P \wedge Q) \equiv (P \wedge R) \equiv P\}\,,
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_{\vee/\oplus} &= \{\mathsf{T}^{(0)}, \mathsf{F}^{(0)}, \vee^{(2)}, \oplus^{(2)}\} \\
A_{\vee/\oplus} &= \{P \oplus (Q \oplus R) = (P \oplus Q) \oplus R\,,\ P \oplus Q = Q \oplus P\,, \\
&\qquad P \vee (Q \vee R) = (P \vee Q) \vee R\,,\ P \vee Q = Q \vee P\} \\
E_{\vee/\oplus} &= \{P \oplus \mathsf{T} = P\,,\ P \oplus P = \mathsf{T}\,,\ P \vee \mathsf{T} = P\,,\ P \vee \mathsf{F} = \mathsf{F}\,,\ P \vee P = P\,, \\
&\qquad P \vee (Q \oplus R) = (P \vee Q) \oplus (P \vee R) \oplus P\}\,.
\end{aligned}
$$

The function symbols $\equiv$ and $\oplus$ denote equivalence and discrepancy, respectively, and have less binding power than any other function symbol. Both symbols are associative and commutative in the theories where they are defined. The other function symbols correspond to those of Σ_{BOOL}; we have chosen not to change their notation in order to keep the definitions and proofs as compact as possible. The symbol $\oplus$ is sometimes denoted by $\not\equiv$ and it is known as either the symmetric difference operator in algebra or as the exclusive or operator in switching theory.

Definition 8. *The nine morphisms appearing in Fig. 1 are defined as follows:*

- *G maps identically T, F and $\vee$. For $\neg$ and $\wedge$ we have: $G(\neg P) = P \equiv \mathsf{F}$ and $G(P \wedge Q) = P \equiv Q \equiv P \vee Q$.*
- *G^{-1} maps identically T, F and $\vee$. For $\equiv$ we have $G^{-1}(P \equiv Q) = (P \vee \neg Q) \wedge (\neg P \vee Q)$.*

– *H maps identically* T, F *and* $\wedge$. *For* $\neg$ *and* $\vee$ *we have:* $H(\neg P) = P \oplus \mathsf{T}$ *and* $H(P \vee Q) = P \oplus Q \oplus P \wedge Q$.
– *H^{-1} maps identically* T, F *and* $\wedge$. *For* $\oplus$ *we have* $H^{-1}(P \oplus Q) = (P \wedge \neg Q) \vee (\neg P \wedge Q)$.
– *K maps identically* T, F *and* $\wedge$. *For* $\neg$ *and* $\vee$ *we have:* $K(\neg P) = P \equiv \mathsf{F}$ *and* $P \vee Q = P \equiv Q \equiv P \wedge Q$.
– *K^{-1} maps identically* T, F *and* $\wedge$. *For* $\equiv$ *we have* $K^{-1}(P \equiv Q) = (P \vee \neg Q) \wedge (\neg P \vee Q)$.
– *L maps identically* T, F *and* $\vee$. *For* $\neg$ *and* $\wedge$ *we have:* $L(\neg P) = P \oplus \mathsf{T}$ *and* $L(P \wedge Q) = P \oplus Q \oplus P \vee Q$.
– *L^{-1} maps identically* T, F *and* $\vee$. *For* $\oplus$ *we have* $L^{-1}(P \oplus Q) = (P \wedge \neg Q) \vee (\neg P \wedge Q)$.
– *op is the duality morphism for Boolean algebras, mapping* T *to* F, F *to* T, $\neg$ *to* $\neg$, $\wedge$ *to* $\vee$ *and* $\vee$ *to* $\wedge$.

Observe that, except for *op* which is its own inverse, all the other morphisms come in pairs of a morphism and its inverse. The 'translation' of $\wedge$ and $\vee$ in every case is known as the *golden rule* [10]. The translations of $\equiv$ and $\oplus$ coincide in each case and both coincide if properly composed with *op*.

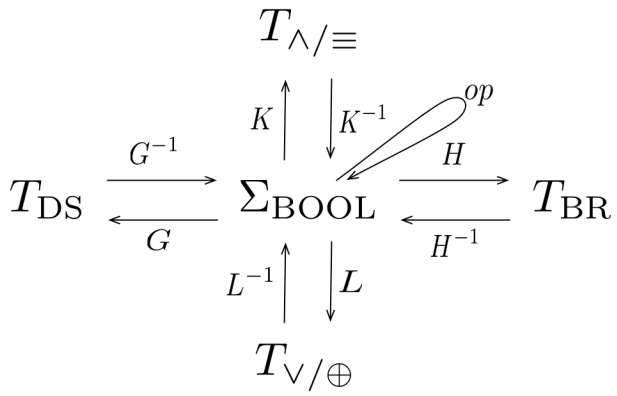

Fig. 1. Isomorphisms between the Boolean theory and the other four theories.

Theorem 1. *The morphisms op, G, H, K and L are theory isomorphisms between the corresponding theories.*

Proof. Since *op* is its own inverse, its being a theory isomorphism is obvious. For the other four we have: for each theory morphism $M : T_{\mathrm{BOOL}} = (\Sigma_{\mathrm{BOOL}}, E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}}) \longrightarrow (\Sigma, E \uplus A)$, we prove:

(a) All the axioms of T_{BOOL} hold in $(\Sigma, E \uplus A)$,
(b) All the axioms of $(\Sigma, E \uplus A)$ hold in $(\Sigma, E \uplus A)T_{\mathrm{BOOL}}$,

(c) $M \circ M^{-1}$ is the identity in $(\Sigma, E \uplus A)$, and
(d) $M^{-1} \circ M$ is the identity in $(\Sigma, E \uplus A)T_{\mathrm{BOOL}}$.

Note that these conditions correspond to the ones an isomorphism between theories should satisfy, as indicated in Section 2.

1. Let us begin with G.
   (a) $E_{\mathrm{DS}} \uplus A_{\mathrm{DS}} \vdash G(\alpha) = G(\beta)$, for $\alpha = \beta \in E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}}$.

   This proof is obtained mechanically by term rewriting in the Maude System [4]. First, we present the executable specification of T_{DS} in Maude:

```
fmod BOOL-DS is
   sort BoolDS .
   ops TRUE FALSE : -> BoolDS .
   op _equ_ : BoolDS BoolDS -> BoolDS [assoc comm prec 80] .
   op _or_ : BoolDS BoolDS -> BoolDS [assoc comm prec 50] .
   vars P Q R : BoolDS .
   eq P equ P = TRUE .
   eq P equ TRUE = P .
   eq P or TRUE = TRUE .
   eq P or FALSE = P .
   eq P or P = P .
   eq P or ( Q equ R ) = P or Q equ P or R .
endfm
```

   Note that only the function symbols in Σ_{DS} appear in the specification: `TRUE` and `FALSE` respectively correspond to T and F, `equ` to $\equiv$ and `or` to $\vee$. The associativity and commutativity properties of $\equiv$ and $\vee$ (i.e. the axioms in A_{DS}) are specified with the operation attributes `assoc` and `comm`, respectively. The precedence of each function symbol is specified with the keyword `prec` followed by a natural number: the higher this value, the lower the binding power. After the declaration of the function symbols, the specification of the equations follows, which is an ad-hoc translation of the axioms in E_{DS}. We refer the reader to Maude's manual [4] for further details.
   To have the translation due to G in an automatic way, we extended the previous specification with the symbols `not` and `and` for the function symbols $\neg$ and $\wedge$ of Σ_{BOOL}, respectively, together with the corresponding equations in the morphism G, namely:

```
   op not_ : BoolDS -> BoolDS [prec 10] .
   op _and_ : BoolDS BoolDS -> BoolDS [prec 50] .
   eq not P = P equ FALSE .
   eq P and Q = P equ Q equ P or Q .
```

   We use Maude's built-in symbol `==` to denote syntactic equality of terms; the execution script of the proofs is show below.

```
reduce in BOOL-DS : (P or Q) or R == P or (Q or R) .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P and (Q and R) == (P and Q) and R .
rewrites: 9 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P or Q == Q or P .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P and Q == Q and P .
rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P or P == P .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P and P == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P or (P and Q) == P .
rewrites: 8 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P and (P or Q) == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P or (Q and R) == (P or Q) and (P or R) .
rewrites: 6 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P and (Q or R) == (P and Q) or (P and R) .
rewrites: 22 in 0ms cpu (2ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P or TRUE == TRUE .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS : P and FALSE == FALSE .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

(b) $E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}} \vdash G^{-1}(\alpha) = G^{-1}(\beta)$, for $\alpha = \beta \in E_{\mathrm{DS}} \uplus A_{\mathrm{DS}}$.

Since $G^{-1}(\vee) = \vee$, associativity and commutativity of $\vee$ in T_{DS} follow directly from the associativity and commutative of $\vee$ in T_{BOOL}.

i. $G^{-1}(P \vee (Q \vee R)) = G^{-1}((P \vee Q) \vee R)$.
Trivial, since $\vee$ is associative in T_{BOOL}.

ii. $G^{-1}(P \vee Q) = G^{-1}(Q \vee P)$.
Trivial, since $\vee$ is commutative in T_{BOOL}.

iii. $G^{-1}(P \equiv (Q \equiv R)) = G^{-1}((P \equiv Q) \equiv R)$. After algebraic simplifications, it is easy to show that both sides of the equality reduce to $(P \wedge Q \wedge R) \vee (P \wedge \neg Q \wedge \neg R) \vee (\neg P \wedge Q \wedge \neg R) \vee (\neg P \wedge \neg Q \wedge R)$.

iv. $G^{-1}(P \equiv Q) = G^{-1}(Q \equiv P)$.
$G^{-1}(P \equiv Q)$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$(P \vee \neg Q) \wedge (\neg P \vee Q)$
$=$ $\langle$ $\vee$ is commutative in T_{BOOL} $\rangle$
$(\neg Q \vee P) \wedge (Q \vee \neg P)$
$=$ $\langle$ $\wedge$ is commutative in T_{BOOL} $\rangle$
$(Q \vee \neg P) \wedge (\neg Q \vee P)$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$G^{-1}(Q \equiv P)$.

v. $G^{-1}(P \vee \mathsf{T}) = G^{-1}(\mathsf{T})$.
$G^{-1}(P \vee \mathsf{T})$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$P \vee \mathsf{T}$
$=$ $\langle$ Excluded middle $\rangle$
$P \vee P \vee \neg P$
$=$ $\langle$ $\vee$ is idempotent in T_{BOOL} $\rangle$
$P \vee \neg P$
$=$ $\langle$ Excluded middle $\rangle$
T
$=$ $\langle$ Def. of G^{-1} $\rangle$
$G^{-1}(\mathsf{T})$

vi. $G^{-1}(P \vee P) = G^{-1}(P)$.
Trivial, since $\vee$ is idempotent in T_{BOOL}.

vii. $G^{-1}(P \vee \mathsf{F}) = G^{-1}(P)$.
$G^{-1}(P \vee \mathsf{F})$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$P \vee \mathsf{F}$
$=$ $\langle$ Contradiction $\rangle$
$P \vee (P \wedge \neg P)$

$=$ $\langle$ Absorption $\rangle$
P
$=$ $\langle$ Def. of G^{-1} $\rangle$
$G^{-1}(P)$.

viii. $G^{-1}(P \equiv \mathsf{T}) = G^{-1}(P)$.
$G^{-1}(P \equiv \mathsf{T})$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$(P \vee \neg\mathsf{T}) \wedge (\neg P \vee \mathsf{T})$
$=$ $\langle$ Excluded middle, $\neg\mathsf{T} = \mathsf{F}$ $\rangle$
$(P \vee \mathsf{F}) \wedge \mathsf{T}$
$=$ $\langle$ F is the module of $\vee$ and T of $\wedge$ in T_{BOOL} $\rangle$
P
$=$ $\langle$ Def. of G^{-1} $\rangle$
$G^{-1}(P)$.

ix. $G^{-1}(P \equiv P) = G^{-1}(\mathsf{T})$.
$G^{-1}(P \equiv P)$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$(P \vee \neg P) \wedge (\neg P \vee P)$
$=$ $\langle$ Excluded middle $\rangle$
$\mathsf{T} \wedge \mathsf{T}$
$=$ $\langle$ T is the module of $\wedge$ $\rangle$
T
$=$ $\langle$ Def. of G^{-1} $\rangle$
$G^{-1}(\mathsf{T})$.

x. $G^{-1}(P \vee (Q \equiv R)) = G^{-1}(P \vee Q \equiv P \vee R)$.
In this proof we use the well-known fact that both DeMorgan's laws hold in any Boolean algebra. We also rely on the identity $P \vee (\neg P \wedge Q) = P \vee Q$:

$$P \vee (\neg P \wedge Q) = (P \vee \neg P) \wedge (P \vee Q) = \mathsf{T} \wedge (P \vee Q) = P \vee Q.$$

$G^{-1}(P \vee Q \equiv P \vee R)$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$(P \vee Q \vee \neg(P \vee R)) \wedge (\neg(P \vee Q) \vee P \vee R)$
$=$ $\langle$ DeMorgan's law $\rangle$
$(P \vee Q \vee (\neg P \wedge \neg R)) \wedge ((\neg P \wedge \neg Q) \vee P \vee R)$
$=$ $\langle$ Given identity $\rangle$
$(P \vee Q \vee \neg R) \wedge (P \vee \neg Q \vee R)$
$=$ $\langle$ Distribution of $\vee$ over $\wedge$ $\rangle$
$P \vee ((Q \vee \neg R) \wedge (\neg Q \vee R))$
$=$ $\langle$ Def. of G^{-1} $\rangle$

$G^{-1}(P \vee (Q \equiv R))$.

(c) $G \circ G^{-1} = 1_{\mathrm{DS}}$.

Since for T, F and $\vee$ both G and G^{-1} are the identity, it is enough to show that $G(G^{-1}(P \equiv Q)) = P \equiv Q$, that is,

$$G((P \vee \neg Q) \wedge (\neg P \vee Q)) = P \equiv Q.$$

This proof has been obtained mechanically by term rewriting using the extended specification of T_{DS} presented above:

```
reduce in BOOL-DS : (P:BoolDS or not Q:BoolDS) and (not P:BoolDS or Q:BoolDS)
    (Q:BoolDS equ P:BoolDS) .
rewrites: 20 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

(d) $G^{-1} \circ G = 1_{\mathrm{BOOL}}$.

Since for T, F and $\vee$ both G and G^{-1} are the identity, the condition is only checked for $\neg$ and $\wedge$.

i. $(G^{-1} \circ G)(\neg P) = \neg P$.
$(G^{-1} \circ G)(\neg P)$
$=$ $\langle$ Def. of $\circ$ and def. G $\rangle$
$G^{-1}(P \equiv \mathsf{F})$
$=$ $\langle$ Def. of G^{-1} $\rangle$
$(P \vee \neg\mathsf{F}) \wedge (\neg P \vee \mathsf{F})$
$=$ $\langle$ $\neg\mathsf{F} = \mathsf{T}$, F is the module of $\vee$ in T_{BOOL} $\rangle$
$(P \vee \mathsf{T}) \wedge \neg P$
$=$ $\langle$ Excluded middle $\rangle$
$\mathsf{T} \wedge \neg P$
$=$ $\langle$ T is the module of $\wedge$ $\rangle$
$\neg P$.

ii. $(G^{-1} \circ G)(P \wedge Q) = P \wedge Q$.
Observe that

$$G^{-1}(Q \equiv P \vee Q) = (Q \vee \neg(P \vee Q)) \wedge (\neg Q \vee P \vee Q) = \neg P \vee Q.$$

We also use the fact that $\neg\neg P = P$ (Double negation) holds in T_{BOOL}. $(G^{-1} \circ G)(P \wedge Q)$
$=$ $\langle$ Def. of $\circ$ and def. G $\rangle$
$G^{-1}(P \equiv Q \equiv P \vee Q)$
$=$ $\langle$ Associativity of $\equiv$, def. of G^{-1}, given identity $\rangle$
$(P \vee \neg(\neg P \vee Q)) \wedge (\neg P \vee \neg P \vee Q)$
$=$ $\langle$ DeMorgan, double negation, idempotency of $\vee$ $\rangle$
$(P \vee (P \wedge Q)) \wedge (\neg P \vee Q)$

$=$ $\langle$ Absorption law $\rangle$
$P \wedge (\neg P \vee Q)$
$=$ $\langle$ Identity of a previous proof $\rangle$
$P \wedge Q$.

Thus, G and G^{-1} are theory isomorphisms for T_{BOOL} and T_{DS}, respectively, and both are inverse morphisms.

2. We continue with the corresponding proof for H.
   (a) $E_{\mathrm{BR}} \uplus A_{\mathrm{BR}} \vdash G(\alpha) = G(\beta)$, for $\alpha = \beta \in E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}}$.

We have mechanized T_{BR} in the Maude system. The specification with the corresponding extension for $\neg$ and $\vee$ is presented below. We follow the same naming convention we had for `BOOL-DS` and include `neq` for $\oplus$.

```
fmod BOOL-BR is
   sort BoolBR .
   ops TRUE FALSE : -> BoolBR .
   op _neq_ : BoolBR BoolBR -> BoolBR [assoc comm prec 80] .
   op _and_ : BoolBR BoolBR -> BoolBR [assoc comm prec 50] .
   op not_ : BoolBR -> BoolBR [prec 10] .
   op _or_ : BoolBR BoolBR -> BoolBR [prec 50] .
   vars P Q R : BoolBR .
   eq P neq P = FALSE .
   eq P neq FALSE = P .
   eq P and TRUE = P .
   eq P and FALSE = FALSE .
   eq P and P = P .
   eq P and ( Q neq R ) = P and Q neq P and R .
   eq not P = P neq TRUE .
   eq P or Q = P neq Q neq P and Q .
endfm
```

The script with the proofs is shown below.

```
reduce in BOOL-BR : P or (Q or R) == (P or Q) or R .
rewrites: 9 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P and (Q and R) == (P and Q) and R .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P or Q == Q or P .
rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P and Q == Q and P .
```

```
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P or P == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P and P == P .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P or Q and P == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P and (P or Q) == P .
rewrites: 8 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P or (Q and R) == (P or Q) and (P or R) .
rewrites: 22 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P and (Q or R) == (P and Q) or (P and R) .
rewrites: 6 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P or TRUE == TRUE .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-BR : P and FALSE == FALSE .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

(b) $E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}} \vdash H^{-1}(\alpha) = H^{-1}(\beta)$, for $\alpha = \beta \in E_{\mathrm{BR}} \uplus A_{\mathrm{BR}}$.

i. $H^{-1}(P \wedge (Q \wedge R)) = H^{-1}((P \wedge Q) \wedge R)$.
Trivial, since $\wedge$ is associative in T_{BOOL}.

ii. $H^{-1}(P \wedge Q) = H^{-1}(Q \wedge P)$.
Trivial, since $\wedge$ is commutative in T_{BOOL}.

iii. $H^{-1}(P \oplus (Q \oplus R)) = H^{-1}((P \oplus Q) \oplus R)$.
Observe that:

$op(H^{-1}(P \oplus Q)) = op((P \wedge \neg Q) \vee (\neg P \wedge Q)) = (P \vee \neg Q) \wedge (\neg P \vee Q) = G(P \equiv Q)$.

Since $\equiv$ is associative, $\oplus$ is necessarily associative.

iv. $H^{-1}(P \oplus Q) = H^{-1}(Q \oplus P)$. Accordingly, as in the previous proof, $\oplus$ is commutative since $\equiv$ is commutative.

The remaining proofs for this requirement are similar in structure to those proofs in the same numeral for T_{DS}.

(c) $H \circ H^{-1} = 1_{\mathrm{BR}}$.

Since for T, F and $\wedge$ both H and H^{-1} are the identity, it is enough to show that $H(H^{-1}(P \oplus Q)) = P \oplus Q$, that is,

$$H((P \wedge \neg Q) \vee (\neg P \wedge Q)) = P \oplus Q.$$

This proof has been obtained mechanically by term rewriting using the extended specification of T_{BR} presented above:

```
reduce in BOOL-BR : (P and not Q) or (not P and Q) == (P neq Q) .
rewrites: 20 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

(d) $H^{-1} \circ H = 1_{\mathrm{BOOL}}$.

Since for T, F and $\wedge$ both H and H^{-1} are the identity, the condition is only checked for $\neg$ and $\vee$.

i. $(H^{-1} \circ H)(\neg P) = \neg P$.
$(H^{-1} \circ H)(\neg P)$
$=$ $\langle$ Def. of H $\rangle$
$H^{-1}(P \oplus \mathsf{T})$
$=$ $\langle$ Def. of H^{-1} $\rangle$
$(P \wedge \neg\mathsf{T}) \vee (\neg P \wedge \mathsf{T})$
$=$ $\langle$ $\neg\mathsf{T} = \mathsf{F}$, T is the module of $\wedge$ $\rangle$
$(P \wedge \mathsf{F}) \vee \neg P$
$=$ $\langle$ Contradiction $\rangle$
$\mathsf{F} \vee \neg P$
$=$ $\langle$ F is the module of $\vee$ $\rangle$
$\neg P$.

ii. $(H^{-1} \circ H)(P \vee Q) = P \vee Q$.
Observe that

$$H^{-1}(Q \oplus P \wedge Q = (Q \wedge \neg(P \wedge Q)) \vee (\neg Q \wedge P \wedge Q) = \neg P \wedge Q.$$

$(H^{-1} \circ H)(P \vee Q)$
$=$ $\langle$ Def. of H $\rangle$
$H^{-1}(P \oplus Q \oplus P \wedge Q)$
$=$ $\langle$ Associativity of $\oplus$, def. of H^{-1}, given identity $\rangle$

$(P \wedge \neg(\neg P and Q)) \vee (\neg P \wedge \neg P \wedge Q)$
$=$ $\langle$ DeMorgan, double negation, idempotency of $\wedge$ $\rangle$
$(P \wedge (P \wedge \neg Q)) \vee (\neg P \wedge Q)$
$=$ $\langle$ Absorption law $\rangle$
$P \vee (\neg P \wedge Q)$
$=$ $\langle$ $P \vee (\neg P \wedge Q) = P \vee Q$ $\rangle$
$P \vee Q$.

This concludes the proof for H and H^{-1}.

3. For K we have:

   (a) $E_{\wedge/\equiv} \uplus A_{\wedge/\equiv} \vdash K(\alpha) = K(\beta)$, for $\alpha = \beta \in E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}}$.

   We have mechanized $T_{\wedge/\equiv}$ in the Maude system. The specification with the corresponding extension for $\neg$ and $\vee$ is presented below. We follow the same naming convention we had for `BOOL-DS` and `BOOL-BR`.

```
fmod BOOL-AE is
   sort BoolAE .
   ops TRUE FALSE : -> BoolAE .
   op _equ_ : BoolAE BoolAE -> BoolAE [assoc comm prec 80] .
   op _and_ : BoolAE BoolAE -> BoolAE [assoc comm prec 50] .
   op not_ : BoolAE -> BoolAE [prec 10] .
   op _or_ : BoolAE BoolAE -> BoolAE [prec 50] .
   vars P Q R : BoolAE .
   eq P equ P = TRUE .
   eq P equ TRUE = P .
   eq P and TRUE = P .
   eq P and FALSE = FALSE .
   eq P and P = P .
   eq P and ( Q equ R ) = P equ P and Q equ P and R .
   eq P or Q = P equ Q equ P and Q .
   eq not P = P equ FALSE .
endfm
```

   The script with the proofs is shown below.

```
reduce in BOOL-AE : P or (Q or R) == (P or Q) or R .
rewrites: 13 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P and (Q and R) == (P and Q) and R .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P or Q == Q or P .
rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)
```

```
result Bool: true
==========================================
reduce in BOOL-AE : P and Q == Q and P .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P or P == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P and P == P .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P or (P and Q) == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P and (P or Q) == P .
rewrites: 10 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P or (Q and R) == (P or Q) and (P or R) .
rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P and (Q or R) == (P and Q) or (P and R) .
rewrites: 8 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P or TRUE == TRUE .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-AE : P and FALSE == FALSE .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

(b) $E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}} \vdash K^{-1}(\alpha) = K^{-1}(\beta)$, for $\alpha = \beta \in E_{\wedge/\equiv} \uplus A_{\wedge/\equiv}$.

i. $K^{-1}(P \wedge (Q \wedge R)) = K^{-1}((P \wedge Q) \wedge R)$.
Trivial, since $\wedge$ is associative in T_{BOOL}.

ii. $K^{-1}(P \wedge Q) = K^{-1}(Q \wedge P)$.
Trivial, since $\wedge$ is commutative in T_{BOOL}.

iii. $K^{-1}(P \equiv (Q \equiv R)) = K^{-1}((P \equiv Q) \equiv R)$.

Observe that $G^{-1}(\equiv) = K^{-1}(\equiv)$, and we have already proved that for G^{-1} this proposition holds. Hence, it holds for K^{-1}.

iv. $K^{-1}(P \equiv Q) = K^{-1}(Q \equiv P)$. Accordingly, as in the previous proof, $\equiv$ is commutative.

v. $K^{-1}(P \wedge \mathsf{T}) = K^{-1}(P)$.
$K^{-1}(P \wedge \mathsf{T})$
$=$ $\langle$ Def. of K^{-1} $\rangle$
$P \wedge \mathsf{T}$
$=$ $\langle$ Excluded middle $\rangle$
$P \wedge (P \vee \neg P)$
$=$ $\langle$ Distribution of $\wedge$ over $\vee$ $\rangle$
$(P \wedge P) \vee (P \wedge \neg P)$
$=$ $\langle$ Idempotency of $\wedge$, contradiction $\rangle$
$P \vee \mathsf{F}$
$=$ $\langle$ F is the module of $\vee$ $\rangle$
P
$=$ $\langle$ Def. of K^{-1} $\rangle$
$K^{-1}(P)$.

vi. $K^{-1}(P \wedge P) = K^{-1}(P)$.
Trivial, since $\wedge$ is idempotent in T_{BOOL}.

vii. $K^{-1}(P \wedge \mathsf{F}) = K^{-1}(\mathsf{F})$.
$K^{-1}(P \wedge \mathsf{F})$
$=$ $\langle$ Def. of K^{-1} $\rangle$
$P \wedge \mathsf{F}$
$=$ $\langle$ Contradiction $\rangle$
$P \wedge (P \wedge \neg P)$
$=$ $\langle$ $\wedge$ is associative $\rangle$
$(P \wedge P) \wedge \neg P$
$=$ $\langle$ $\wedge$ is idempotent $\rangle$
$P \wedge \neg P$
$=$ $\langle$ Contradiction $\rangle$
F
$=$ $\langle$ Def. of K^{-1} $\rangle$
$K^{-1}(\mathsf{F})$.

viii. $K^{-1}(P \equiv \mathsf{T}) = K^{-1}(P)$
Follows from the observation that $G^{-1}(\equiv)$ and $K^{-1}(\equiv)$ are equivalent, and from the proof for the same proposition for G.

ix. $K^{-1}(P \equiv P) = K^{-1}(\mathsf{T})$.

Follows from the observation that $G^{-1}(\equiv)$ and $K^{-1}(\equiv)$ are equivalent, and from the proof for the same proposition for G.

x. $K^{-1}(P \wedge (Q \equiv R)) = K^{-1}(P \equiv P \wedge Q \equiv P \wedge R)$.
First, observe that we have the following identity in T_{BOOL}:

$$K^{-1}(P \equiv P \wedge Q) = (P \vee \neg(P \wedge Q)) \vee (\neg P \vee (P \wedge Q)) = \neg P \vee Q$$

Then, we have:
$K^{-1}(P \equiv P \wedge Q \equiv P \wedge R)$
$=$ $\langle$ Def. of K^{-1}, previous observation $\rangle$
$((\neg P \vee Q) \vee \neg(P \wedge R)) \wedge (\neg(\neg P \vee Q) \vee (P \wedge R))$
$=$ $\langle$ DeMorgan, double negation $\rangle$
$(\neg P \vee Q \vee \neg P \vee \neg R) \wedge ((P \wedge \neg Q) \vee (P \wedge R))$
$=$ $\langle$ $\vee$ is idempotent, $\wedge$ distributes over $\vee$ $\rangle$
$(\neg P \vee Q \vee \neg R) \wedge P \wedge (\neg Q \vee R)$
$=$ $\langle$ $P \wedge (\neg P \vee Q) = P \wedge Q$ $\rangle$
$(Q \vee \neg R) \wedge P \wedge (\neg Q \vee R)$
$=$ $\langle$ $\wedge$ is commutative and associative $\rangle$
$P \wedge ((Q \vee \neg R) \wedge (\neg Q \vee R))$
$=$ $\langle$ Def. of K^{-1}, previous observation $\rangle$
$K^{-1}(P \wedge (Q \equiv R))$.

(c) $K \circ K^{-1} = 1_{\wedge/\equiv}$.

Since for T, F and $\wedge$ both K and K^{-1} are the identity, it is enough to show that $K(K^{-1}(P \equiv Q)) = P \equiv Q$, which follows directly from the observation of $K^{-1}(\equiv) = G^{-1}(\equiv)$.

(d) $K^{-1} \circ K = 1_{\mathrm{BOOL}}$.

Since for T, F and $\wedge$ both K and K^{-1} are the identity, the condition is only checked for $\neg$ and $\vee$.

i. $(K^{-1} \circ K)(\neg P) = \neg P$.
The observation of $G^{-1}(\equiv) = K^{-1}(\equiv)$, and the proof of this same statement for G, gives us a direct proof.

ii. $(K^{-1} \circ K)(P \vee Q) = P \vee Q$.
The observation of $G^{-1}(\equiv) = K^{-1}(\equiv)$, and the proof of this same statement for G, gives us a direct proof.

This concludes the proof for K and K^{-1}.

4. Finally, for L we have:

(a) $E_{\vee/\oplus} \uplus A_{\vee/\oplus} \vdash L(\alpha) = L(\beta)$, for $\alpha = \beta \in E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}}$.

We have mechanized $T_{\vee/\oplus}$ in the Maude system. The specification with the corresponding extension for $\neg$ and $\vee$ is presented below. We follow the same naming convention we had for `BOOL-DS` and `BOOL-BR`.

```
fmod BOOL-OX is
   sort BoolOX .
   ops TRUE FALSE : -> BoolOX .
   op _or_ : BoolOX BoolOX -> BoolOX [assoc comm prec 50] .
   op _neq_ : BoolOX BoolOX -> BoolOX [assoc comm prec 80] .
   op not_ : BoolOX -> BoolOX [prec 10] .
   op _and_ : BoolOX BoolOX -> BoolOX [prec 50] .
   vars P Q R : BoolOX .
   eq P neq P = FALSE .
   eq P neq FALSE = P .
   eq P or TRUE = TRUE .
   eq P or FALSE = P .
   eq P or P = P .
   eq P or ( Q neq R ) = P neq P or Q neq P or R .
   eq P and Q = P neq Q neq P or Q .
   eq not P = P neq TRUE .
endfm
```

The script with the proofs is shown below.

```
reduce in BOOL-OX : P or (Q or R) == (P or Q) or R .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P and (Q and R) == (P and Q) and R .
rewrites: 13 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P or Q == Q or P .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P and Q == Q and P .
rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P or P == P .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P and P == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

```
==========================================
reduce in BOOL-OX : P or (P and Q) == P .
rewrites: 10 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P and (P or Q) == P .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P or (Q and R) == (P or Q) and (P or R) .
rewrites: 8 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P and (Q or R) == (P and Q) or (P and R) .
rewrites: 32 in 10ms cpu (0ms real) (3200 rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P or TRUE == TRUE .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-OX : P and FALSE == FALSE .
rewrites: 5 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

(b) $E_{\mathrm{BOOL}} \uplus A_{\mathrm{BOOL}} \vdash L^{-1}(\alpha) = L^{-1}(\beta)$, for $\alpha = \beta \in E_{\vee/\oplus} \uplus A_{\vee/\oplus}$.

i. $L^{-1}(P \vee (Q \vee R)) = L^{-1}((P \vee Q) \vee R)$.
Trivial, since $\vee$ is associative in T_{BOOL}.

ii. $L^{-1}(P \vee Q) = L^{-1}(Q \vee P)$.
Trivial, since $\vee$ is commutative in T_{BOOL}.

iii. $H^{-1}(P \oplus (Q \oplus R)) = H^{-1}((P \oplus Q) \oplus R)$.
This follows from $H^{-1}(\oplus) = L^{-1}(\oplus)$ and the proof given before for H^{-1}.

iv. $H^{-1}(P \oplus Q) = H^{-1}(Q \oplus P)$. Accordingly, as in the previous proof, $\oplus$ is commutative.

The remaining proofs for this requirement are similar in structure to those proofs in the same numeral for T_{BR}.

(c) $L \circ L^{-1} = 1_{\vee/\oplus}$.

Since for T, F and $\vee$ both L and L^{-1} are the identity, it is enough to show that $L(L^{-1}(P \oplus Q)) = P \oplus Q$, which follows directly from the ob-

servation of $L^{-1}(\oplus) = H^{-1}(\oplus)$ and the corresponding proof for H^{-1}.

(d) $L^{-1} \circ L = 1_{\mathrm{BOOL}}$.

Since for T, F and $\vee$ both L and L^{-1} are the identity, the condition is only checked for $\neg$ and $\wedge$.

i. $(L^{-1} \circ L)(\neg P) = \neg P$.
The observation of $H^{-1}(\equiv) = L^{-1}(\equiv)$, and the proof of this same statement for H, gives us a direct proof.

ii. $(L^{-1} \circ L)(P \vee Q) = P \vee Q$.
The observation of $H^{-1}(\equiv) = L^{-1}(\equiv)$, and the proof of this same statement for L, gives us a direct proof.

This concludes the proof for L and L^{-1}.

This concludes the proof of Theorem 1.

We call these isomorphisms *Boolean isomorphisms*. They give rise to new Boolean isomorphisms by composition among them.

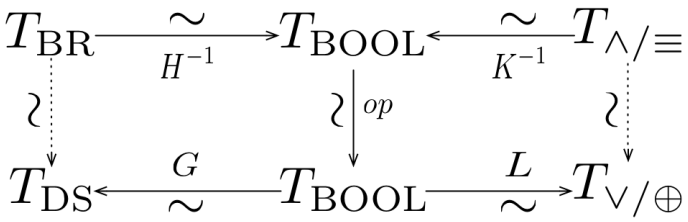

Fig. 2. Commutation and composition of Boolean isomorphisms.

Figure 2 highlights two particular ones, namely $G \circ op \circ H^{-1}$ and $L \circ op \circ K^{-1}$, which show that the theories T_{BR} and T_{DS}, and the theories $T_{\wedge/\equiv}$ and $T_{\vee/\oplus}$ are pairs of *dual theories*. These morphisms are used in the next section to build decision procedures for propositional logic by term rewriting using the non-traditional Boolean theories T_{DS}, T_{BR}, $T_{\wedge/\equiv}$ and $T_{\vee/\oplus}$. It is important to point out that the relation between the work of Dijkstra-Scholten and that of J.Hsiang has passed unnoticed until now.

4 Four Equational Decision Procedures

Equational theories can be used as programs when appropriate executability requirements are satisfied, so that their equations can be used as *simplification rules* from left to right. Typical requirements are that the rules should be

terminating (i.e., that there are no infinite simplification sequences) and *confluent*, which under the termination assumption is equivalent to the fact that each expression can be fully simplified to a *unique* equivalent expression, called its *canonical form*, that cannot be further simplified. Termination and confluence may be defined *modulo* a set A of axioms, as it will be the case for all our decision procedures. If we have such simplification system, we then have a decision procedure for the word problem in the theory.

In this section we construct a decision procedure for propositional logic from the equational theory T_{DS} by means of *definitional extensions* (Section 2.1). Same constructions apply to T_{BR} (where it is well-known since [13]), to $T_{\wedge/\equiv}$ and to $T_{\vee/\oplus}$. Decision procedures for propositional logic have been a matter of interest in the rewriting community. The first decision procedure for propositional logic based on rewriting was developed by J. Hsiang in the early 1980s [13].

Consider the natural extension $T_{\mathrm{BOOL}'}$ of T_{BOOL} with the function symbols $\Rightarrow^{(2)}$ (implication) and $\Leftarrow^{(2)}$ (consequence) and with corresponding equations $P \Rightarrow Q = \neg P \vee Q$ and $P \Leftarrow Q = P \vee \neg Q$, and let M be the theory isomorphism from $T_{\mathrm{BOOL}'}$ to T_{BOOL}. Since both G and M are unambiguous, by Lemma 1, we obtaing the theory $T_{\mathrm{DS}}^{G\otimes M}$ as a definitional extension of T_{DS} in which negation, conjunction, implication and consequence are included:

$$
\begin{array}{ll}
\Sigma_{\mathrm{DS}}^{G\otimes M} & = \{\mathsf{T}^{(0)}, \mathsf{F}^{(0)}, \vee^{(2)}, \wedge^{(2)}, \equiv^{(2)}, \Rightarrow^{(2)}, \Leftarrow^{(2)}\} \\
A_{\mathrm{DS}}^{G\otimes M} & = \{P \equiv (Q \equiv R) = (P \equiv Q) \equiv R\,,\ P \equiv Q = Q \equiv P\,, \\
 & \quad P \vee (Q \vee R) = (P \vee Q) \vee R\,,\ P \vee Q = Q \vee P\,, \\
 & \quad P \wedge (Q \wedge R) = (P \wedge Q) \wedge R\,,\ P \wedge Q = Q \wedge P\} \\
E_{\mathrm{DS}}^{G\otimes M} & = \{P \equiv \mathsf{T} = P\,,\ P \equiv P = \mathsf{T}\,,\ P \vee \mathsf{T} = \mathsf{T}\,,\ P \vee \mathsf{F} = P\,,\ P \vee P = P \\
 & \quad P \vee (Q \equiv R) = (P \vee Q) \equiv (P \vee R)\,,\ P \wedge Q = P \equiv Q \equiv P \vee Q\,, \\
 & \quad P \Rightarrow Q = P \vee Q \equiv Q\,,\ P \Leftarrow Q = P \vee Q \equiv P\,\}\,.
\end{array}
$$

Furthermore, we can extend this theory to include the $\oplus$ operator in the following way:

$$T_{\mathrm{DS}}^{\mathrm{DEC}} = (T_{\mathrm{DS}}^{G\otimes M})^{J\circ H^{-1}},$$

where J is the inclusion morphism from T_{BOOL} to $T_{\mathrm{DS}}^{G\otimes M}$. The resulting theory is shown below:

$$
\begin{array}{ll}
\Sigma_{\mathrm{DS}}^{\mathrm{DEC}} & = \Sigma_{\mathrm{DS}}^{G\otimes M} \cup \{\oplus^{(2)}\} \\
A_{\mathrm{DS}}^{\mathrm{DEC}} & = A_{\mathrm{DS}}^{G\otimes M} \cup \{P \oplus (Q \oplus R) = (P \oplus Q) \oplus R\,,\ P \oplus Q = Q \oplus P\,\} \\
E_{\mathrm{DS}}^{\mathrm{DEC}} & = E_{\mathrm{DS}}^{G\otimes M} \cup \{P \oplus Q = P \equiv Q \equiv \mathsf{F}\,\}\,.
\end{array}
$$

$T_{\mathrm{DS}}^{\mathrm{DEC}}$ contains *all* the Boolean function symbols we have defined so far. We prove that $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is a decision procedure for Boolean algebras, and hence, for propositional logic.

Theorem 2. *The equations $E_{\mathrm{DS}}^{\mathrm{DEC}}$ in $T_{\mathrm{DS}}^{\mathrm{DEC}}$ are confluent and terminating modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$.*

Proof. Termination and confluence modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$ can be established mechanically by using formal tools that: (i) find a well-founded ordering $\succ$ on $A_{\mathrm{DS}}^{\mathrm{DEC}}$-

equivalence classes of terms such that $[t]_{A_{\mathrm{DS}}^{\mathrm{DEC}}} \rightarrow_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}} [t']_{A_{\mathrm{DS}}^{\mathrm{DEC}}}$ implies $[t]_{A_{\mathrm{DS}}^{\mathrm{DEC}}} \succ [t']_{A_{\mathrm{DS}}^{\mathrm{DEC}}}$, and (ii) check confluence of $E_{\mathrm{DS}}^{\mathrm{DEC}}$ modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$ by computing all so-called 'critical-pairs' modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$ and showing they are all confluent. We have used the CiME tool [16] to check termination and confluence of $E_{\mathrm{DS}}^{\mathrm{DEC}}$ modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$. In CiME, $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is specified as follows:

```
let S_BoolDS =
   signature
   "
      TRUE,FALSE : constant ;
      not : unary ;
      equ,neq,or,and : AC ;
      imp,cos : infix binary ;
   ";

let V = vars "P Q R";

let R_BoolDS =
   TRS S_BoolDS V
   "
      P equ TRUE -> P ;
      P equ P -> TRUE ;
      not(P) -> P equ FALSE ;
      P neq Q -> not (P equ Q) ;
      P or P -> P ;
      P or FALSE -> P ;
      P or TRUE -> TRUE ;
      P or (Q equ R) -> (P or Q) equ (P or R) ;
      P and Q -> P equ Q equ (P or Q) ;
      P imp Q -> ( P or Q ) equ Q ;
      P cos Q -> ( P or Q ) equ P ;
   ";
```

`S_BoolDS` specifies $\Sigma_{\mathrm{DS}}^{\mathrm{DEC}}$ with the arities and AC properties of the operators, `V` the variables and `R_BoolDS` the equations $E_{\mathrm{DS}}^{\mathrm{DEC}}$, defining a Term Rewrite System (TRS).

Confluence of the TRS modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$ is checked with the following command:

```
CiME> confluence R_BoolDS ;
Computing self critical pairs
...
System is confluent (127 critical pair(s) tested).
```

Termination of the TRS modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$ requires *dependency pairs with marks* and *dependency graphs*:

```
CiME> polyinterpkind {("simple",10)};
CiME> termcrit "dp";
CiME> termcrit "marks";
```

```
CiME> termcrit "graph";
CiME> termination R_BoolDS;
Entering the termination expert. Verbose level = 0
...
(12 termination constraints)
Search parameters: simple polynomials, coefficient bound is 10.
Solution found for these constraints:
[TRUE] = 0;
[FALSE] = 0;
[not](X0) = X0 + 1;
[equ](X0,X1) = X1 + X0 + 1;
[neq](X0,X1) = X1 + X0 + 2;
[or](X0,X1) = X1*X0 + X1 + X0;
[and](X0,X1) = X1*X0 + 2*X1 + 2*X0 + 2;
[imp](X0,X1) = X1*X0 + 2*X1 + X0 + 1;
[cos](X0,X1) = X1*X0 + X1 + 2*X0 + 1;

Termination proof found.
```

Hence, $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is confluent and terminating modulo $A_{\mathrm{DS}}^{\mathrm{DEC}}$.

The executable specification of $T_{\mathrm{DS}}^{\mathrm{DEC}}$ in Maude is presented below. We use `imp` and `cos` for $\Rightarrow$ and $\Leftarrow$, respectively. It can be shown with help of Maude's Sufficient Completeness Checker [12] that the canonical form of any term is either T, F or $t_0 \equiv \ldots \equiv t_n$, where all t_i are distinct disjunctions (modulo AC) of propositional variables.

```
fmod BOOL-DS is
   sort BoolDS .
   ops TRUE FALSE : -> BoolDS [ctor] .
   op _equ_ : BoolDS BoolDS -> BoolDS [ctor assoc comm prec 80] .
   op _or_ : BoolDS BoolDS -> BoolDS [ctor assoc comm prec 50] .
   op not_ : BoolDS -> BoolDS [prec 10] .
   op _and_ : BoolDS BoolDS -> BoolDS [assoc comm prec 50] .
   op _imp_ : BoolDS BoolDS -> BoolDS [prec 60] .
   op _cos_ : BoolDS BoolDS -> BoolDS [prec 60] .
   vars P Q R : BoolDS .
   eq P equ P = TRUE .
   eq P equ TRUE = P .
   eq P or TRUE = TRUE .
   eq P or FALSE = P .
   eq P or P = P .
   eq P or ( Q equ R ) = P or Q equ P or R .
   eq P and Q = P equ Q equ P or Q .
   eq not P = P equ FALSE .
   eq P imp Q = P or Q equ Q .
   eq P cos Q = P or Q equ P .
endfm
```

Lemma 2. *The canonical form* $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t]$ *of any term* t *is either* T, F *or* $t_0 \equiv \ldots \equiv t_n$*, where all* t_i *are distinct disjunctions (modulo AC) of propositional variables.*

Proof. First, we check with Maude's Sufficient Completeness Checker that the canonical form of any term can contain only the function symbols T, F, $\vee$ and $\equiv$. For this purpose, we have annotated the definition of the function symbols `TRUE`, `FALSE`, `or` and `equ`, with the `ctor` (i.e. constructor) attribute, in the specification given above.

```
Maude> load scc
Maude> select CC-LOOP .
Maude> loop init-cc .
Starting the Maude Sufficient and Canonical Completeness Checker.
Maude> (scc BOOL-DS .)
Checking sufficient completeness of BOOL-DS ...
Warning: This module has equations that are not left-linear which will be
    ignored when checking.
Success: BOOL-DS is sufficiently complete under the assumption that it is
    ground weakly-normalizing, confluent, and ground sort-decreasing.
```

Now, we prove that the canonical form of any term is either T, F or $t_0 \equiv \ldots \equiv t_n$, where all t_i are distinct disjunctions (modulo AC) of propositional variables. Assume it is not. By Sufficient Completeness, $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t]$ can only contain T, F, $\vee$ and $\equiv$ as function symbols. Since $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t]$ is not of the form T, F or $t_0 \equiv \ldots \equiv t_n$, where all t_i are distinct disjunctions (modulo AC) of propositional variables, the equation `P or ( Q equ R ) = P or Q equ P or R` could be applied, and therefore, $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t]$ is not the canonical form of t. Since $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is confluent and terminating, this contradicts our assumption. Hence, the canonical form of any term in $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is either T, F or $t_0 \equiv \ldots \equiv t_n$, where all t_i are distinct disjunctions (modulo AC) of propositional variables.

Since $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is both confluent and terminating, we can use it as a *decision procedure for propositional logic*. For any propositional expressions t and t':

$$T_{\mathrm{DS}}^{\mathrm{DEC}} \vdash t = t' \;\Leftrightarrow\; T_{\mathrm{DS}}^{\mathrm{DEC}} \vdash t \equiv t' = \mathsf{T} \;\Leftrightarrow\; \mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t] = \mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t'].$$

In particular, since T and F are both in $E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}$-canonical form, we have:

$$T_{\mathrm{DS}}^{\mathrm{DEC}} \vdash t \equiv t' = \mathsf{T} \;\Leftrightarrow\; \mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t \equiv t'] = [\mathsf{T}]$$

and

$$T_{\mathrm{DS}}^{\mathrm{DEC}} \vdash t \equiv t' = \mathsf{F} \;\Leftrightarrow\; \mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t \equiv t'] = [\mathsf{F}].$$

Definition 9. *We call a proposition t a* tautology *iff* $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t] = [\mathsf{T}]$ *and a* falsity *iff* $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t] = [\mathsf{F}]$. *We call* t satisfiable *iff* $\mathsf{can}_{E_{\mathrm{DS}}^{\mathrm{DEC}}/A_{\mathrm{DS}}^{\mathrm{DEC}}}[t] \neq [\mathsf{F}]$.

Therefore, our decision procedure gives also a decision procedure for checking satisfiability of any proposition t, capability we used in experiments described in Section 6.3.

Theorem 3. *The theories* $T_{\mathrm{BR}}^{\mathrm{DEC}} = (T_{\mathrm{BR}}^{H\otimes M})^{F\circ G^{-1}}$, $T_{\wedge/\equiv}^{\mathrm{DEC}} = (T_{\wedge/\equiv}^{K\otimes M})^{I\circ L^{-1}}$ *and* $T_{\vee/\oplus}^{\mathrm{DEC}} = (T_{\vee/\oplus}^{L\circ M})^{N\circ K-1}$ *are confluent and terminating modulo* $A_{\mathrm{BR}}^{\mathrm{DEC}}$, $A_{\wedge/\equiv}^{\mathrm{DEC}}$ *and* $A_{\vee/\oplus}^{\mathrm{DEC}}$, *respectively, where* F, I *and* N *are the inclusion morphisms from* T_{BOOL} *to the corresponding theories.*

Proof. One can mechanically checked using the CiME tool and following the same procedure used in the proof of Theorem 2 that all these theories, including their definitional extensions, are confluent and terminating modulo the corresponding sets of AC axioms.

The executable specifications of these theories in Maude, and therefore of the decision procedures, can be found in Appendix A.

Remark 1. Following the same schema used in Lemma 2 is easy to check:

- The canonical form $\mathsf{can}_{E_{\mathrm{BR}}^{\mathrm{DEC}}/A_{\mathrm{BR}}^{\mathrm{DEC}}}[t]$ of any term t is either T, F or $t_0 \oplus \ldots \oplus t_n$, where all t_i are distinct conjunctions (modulo AC) of propositional variables.
- The canonical form $\mathsf{can}_{E_{\wedge/\equiv}^{\mathrm{DEC}}/A_{\wedge/\equiv}^{\mathrm{DEC}}}[t]$ of any term t is either T, F or $t_0 \equiv \ldots \equiv t_n$, where all t_i are distinct conjunctions (modulo AC) of propositional variables.
- The canonical form $\mathsf{can}_{E_{\vee/\oplus}^{\mathrm{DEC}}/A_{\vee/\oplus}^{\mathrm{DEC}}}[t]$ of any term t is either T, F or $t_0 \oplus \ldots \oplus t_n$, where all t_i are distinct disjunctions (modulo AC) of propositional variables.

Hence, together with $T_{\mathrm{DS}}^{\mathrm{DEC}}$, these equational theories serve as decision procedures for propositional logic by term rewriting and also solve the satisfiability problem for propositional logic.

Because of the duality isomorphism $G \circ op \circ H^{-1}$, $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is *dual* to Hsiang's canonical system $T_{\mathrm{BR}}^{\mathrm{DEC}}$. The theories $T_{\wedge/\equiv}$ and $T_{\vee/\oplus}$, as well as their definitional extensions, form another pair of dual theories thanks to the duality isomorphism $K^{-1} \circ op \circ L$ mentioned in Section 3.

Decision procedures for propositional logic have been a matter of persistent interest in the rewriting community. The first decision procedure for propositional logic based on rewriting was developed by J.Hsiang in the early 1980s [13], as mentioned before. Various first-order and modal propositional theorem proving systems have used canonical systems for propositional logic. A refutation rewrite-based method, the *N-strategy*, was designed by Hsiang for the word problem in first-order predicate logic in 1985 [15]. The N-strategy used Hsiang's canonical system for Boolean algebras to reduce a set of clauses to canonical form and then converting the clauses into a set of simplifying equations. Following a similar approach, L. Bachmair and N. Dershowitz developed a complete

inference system for first-order theorem proving based on Hsiang's rewrite system for Boolean algebras [1]. D. Kapur and P. Narendran in 1985, following a different approach, developed another refutation method for first order theorem proving. In 1992 A. Foret extended Hsiang's system to give a complete term rewriting system for the modal propositional logics known as K, Q, T and S5 [8]. A generic extension of the four decision procedures presented in this paper has been studied in [21] as an equation-based decision procedure for Monadic First Order Logic's word problem.

5 Optimizing Equations

In Section 3 we have presented five isomorphic Boolean theories. In Section 4 we established that four of them give rise to decision procedures for propositional logic by term rewriting. This section introduces *optimizing equations* as a technique to improve the efficiency of these decision procedures. The goal is to use such optimizing equations as a sound and efficient mechanism to simplify propositional expressions to semantically equivalent ones *before* the decision procedures are applied. We use $T_{\mathrm{DS}}^{\mathrm{DEC}}$ to mechanically prove some propositions in this section. Since the other three decision procedures are isomorphic to this one, the results obtained in this section are generic for all the decision procedures.

Behind optimizing equations lies the idea of exploiting annihilators of Boolean operators to simplify propositions in a generic way. One can observe, for example, that in T_{DS} the constant symbol T is the annihilator of $\vee$, since $P \vee \mathsf{T} = \mathsf{T}$ is one of the equations. On the other hand, it is not clear that F is the annihilator of $\wedge$ in T_{DS}, since there is no equation explicitly describing such property for conjunction. Of course, $P \wedge \mathsf{F}$ simplifies to F no matter what the canonical form of P is, but the T_{DS} simplification process may require substantial amounts of computational resources, since futile simplifications over P could be performed. If P happens to be a large expression, the efficiency of the simplification process can be highly decreased, due to the non-deterministic application of the equations in the decision procedure.

The *optimizing equations* we propose are a set of *generic simplifying equations*, together with other equations that implement a syntactic process of *replacement of atomic propositions with constants*. These generic equations simplify expressions in the presence of an annihilator of a Boolean operator. The replacement of atomic propositions with constants introduces constants in a proposition under certain conditions.

Definition 10. *The set E_{OPT} of generic simplifying equations is defined as follows:*

$$\begin{aligned} E_{\mathrm{OPT}} = \{ & \neg\mathsf{T} = \mathsf{F}\ ,\ \neg\mathsf{F} = \mathsf{T}\ , \neg\neg P = P\ ,\ P \vee \mathsf{T} = \mathsf{T}\ ,\ P \wedge \mathsf{F} = \mathsf{F}\ , \\ & \mathsf{F} \Rightarrow P = \mathsf{T}\ ,\ P \Rightarrow \mathsf{T} = \mathsf{T}\ ,\ P \Leftarrow \mathsf{F} = \mathsf{T},\ \mathsf{T} \Leftarrow P = \mathsf{T}\ \}\ . \end{aligned}$$

Lemma 3. *The set of equations E_{OPT} is confluent and terminating modulo A_{OPT}, where A_{OPT} consists of the AC equations for $\wedge$ and $\vee$.*

Proof. We have mechanically proved, using again the CiME System [16], that the set of equations E_{OPT} is confluent and terminating modulo A_{OPT}. Consider the specification of this set of equations in CiME:

```
let S_BoolOPT =
   signature
   "
      TRUE,FALSE : constant ;
      not : unary ;
      or,and : AC ;
      imp,cos : infix binary ;
   ";

let V = vars "P";

let R_BoolOPT =
   TRS S_BoolOPT V
   "
      not TRUE -> FALSE ;
      not FALSE -> TRUE ;
  not not P -> P ;
  P or TRUE ->  TRUE ;
  P and FALSE -> FALSE ;
  P imp TRUE -> TRUE ;
  FALSE imp P -> TRUE ;
  P cos FALSE -> TRUE ;
  TRUE cos P -> TRUE ;
   ";
```

Confluence of E_{OPT} modulo A_{OPT} is checked as follows:

```
CiME> confluence R_BoolOPT ;
Computing self critical pairs
...
System is confluent (30 critical pair(s) tested).
```

Accordingly, termination is checked with the following script of commands:

```
CiME> polyinterpkind {("simple",5)};
- : unit = ()
CiME> termcrit "dp";
Termination now uses dependency pair criterion
- : unit = ()
CiME> termination R_BoolOPT ;
Entering the termination expert. Verbose level = 0
...
(10 termination constraints)
Search parameters: simple polynomials, coefficient bound is 5.
Solution found for these constraints:
```

```
[TRUE] = 0;
[FALSE] = 0;
[not](X0) = X0;
[or](X0,X1) = 0;
[and](X0,X1) = X1 + X0 + 1;
[imp](X0,X1) = 0;
[cos](X0,X1) = 0;

Termination proof found.
```

Hence, the set of equations E_{OPT} is confluent and terminating modulo A_{OPT}.

Note that the previous lemma does not make any assumption on the underlying Boolean theory, that is, this simplifying equations are generic for the four decision procedures defined in Section 4.

Under certain conditions, some atomic propositions occurring in a propositional expression can be replaced with the constants T or F without altering the truth value of the propositional expression. The *replacement of atomic propositions with constants* relies on a specialization of Shannon's law [10]:

Theorem 4. *Let P and Q be Boolean expressions and p be an atomic proposition. The textual substitution $P(Q/p)$ is the expression obtained from P by respectively replacing each occurrence of p by Q in P.* Shannon's law *is stated as follows:*

$$P = P(p/p) = (p \Rightarrow P(\mathsf{T}/p)) \wedge (\neg p \Rightarrow P(\mathsf{F}/p))$$

We have given a sufficiently complete, confluent and terminating equational specification of Shannon's law in the Maude system by extending $T_{\mathrm{DS}}^{\mathrm{DEC}}$ and then proving by structural induction that it holds as a theorem. Since in Shannon's law, and in the equations given below in Theorem 5, p ranges over atomic propositions, whereas P ranges over propositional expressions, the extension of $T_{\mathrm{DS}}^{\mathrm{DEC}}$ is *order sorted*, with a subsort $\mathsf{AtomDS} < \mathsf{BoolDS}$, so that $p : \mathsf{AtomDS}$ and $P : \mathsf{BoolDS}$. Furthermore, we have define the subsort $\mathsf{ConDS} < \mathsf{BoolDS}$ (for constant symbols), such that $\mathsf{T} : \mathsf{ConDS}$ and $\mathsf{F} : \mathsf{ConDS}$. The extension of $T_{\mathrm{DS}}^{\mathrm{DEC}}$ with Shannon's law as an equation, and with the needed infrastructure, is shown below:

```
fmod BOOL-DS-EXT is
   sort ConDS .
   sort AtomDS .
   sort BoolDS .
   subsorts ConDS AtomDS < BoolDS .
   ops TRUE FALSE : -> ConDS [ctor] .
   op _equ_ : BoolDS BoolDS -> BoolDS [ctor assoc comm prec 80] .
   op _or_ : BoolDS BoolDS -> BoolDS [ctor assoc comm prec 50] .
   op not_ : BoolDS -> BoolDS [prec 10] .
   op _and_ : BoolDS BoolDS -> BoolDS [assoc comm prec 50] .
   op _imp_ : BoolDS BoolDS -> BoolDS [prec 60] .
```

```
   op _cos_ : BoolDS BoolDS -> BoolDS [prec 60] .
   vars P Q R : BoolDS .
   eq P equ P = TRUE .
   eq P equ TRUE = P .
   eq P or TRUE = TRUE .
   eq P or FALSE = P .
   eq P or P = P .
   eq P or ( Q equ R ) = P or Q equ P or R .
   eq P and Q = P equ Q equ P or Q .
   eq not P = P equ FALSE .
   eq P imp Q = P or Q equ Q .
   eq P cos Q = P or Q equ P .
   op _`(_/_`) : BoolDS AtomDS BoolDS -> BoolDS .
   op _`(_/_`) : BoolDS ConDS BoolDS -> BoolDS .
   var Av : AtomDS .
   eq P( Av / Q ) = ( Av imp P( TRUE / Q ) ) and ( not Av imp P( FALSE / Q ) ) .
endfm
```

Observe that:

1. There are two specifications of *textual substitution* which only differ in the second sort of the corresponding specifications. The one appearing first, constraints it to be an atomic variable, as in $P(p/Q)$, while the second constraints it to be a *Boolean constant*, as in $P(\mathsf{T}/Q)$. This, is in turn, exploited by the equation corresponding to Shannon's law in `BOOL-DS-EXT`, so the substitution process is well founded.
2. There is no explicit mention to how the textual substitution is computed. We will soon see that such detailed description is not needed.
3. We are going to use `BOOL-DS-EXT` as a mechanical tool to prove automatically some propositions involving Shannon's law.

The method suggested by Shannon's law, when used without any 'context' is of course of little help, since it gives rise to an exponential explosion of the original proposition in terms of its number of variables. The *replacement of atomic propositions with constants* as a specialization of Shannon's law to various useful contexts is stated as follows:

Theorem 5. *Let P be a propositional expression and p an atomic proposition. The following are theorems of any of the five Boolean theories:*

1.	$p \wedge P(p/p) = p \wedge P(\mathsf{T}/p)$	*5.*	$p \vee P(p/p) = p \vee P(\mathsf{F}/p)$
2.	$\neg p \wedge P(p/p) = \neg p \wedge P(\mathsf{F}/p)$	*6.*	$\neg p \vee P(p/p) = \neg p \vee P(\mathsf{T}/p)$
3.	$p \Rightarrow P(p/p) = p \Rightarrow P(\mathsf{T}/q)$	*7.*	$p \Leftarrow P(p/p) = p \Leftarrow P(\mathsf{F}/p)$
4.	$\neg p \Rightarrow P(p/p) = \neg p \Rightarrow P(\mathsf{F}/p)$	*8.*	$\neg p \Leftarrow P(p/p) = \neg p \Leftarrow P(\mathsf{T}/p)$.

Proof. We use the executable specification `BOOL-DS-EXT` to mechanically prove all eight propositions.

```
reduce in BOOL-DS-EXT : p:AtomDS and P(p:AtomDS / p:AtomDS) ==
```

```
                        p:AtomDS and P(TRUE /  p:AtomDS) .
rewrites: 24 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : not p:AtomDS and P(p:AtomDS / p:AtomDS) ==
                        not p:AtomDS and P(FALSE / p:AtomDS) .
rewrites: 33 in 0ms cpu (2ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : p:AtomDS imp P(p:AtomDS / p:AtomDS) ==
                        p:AtomDS imp P(TRUE / p:AtomDS) .
rewrites: 24 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : not p:AtomDS imp P(p:AtomDS / p:AtomDS) ==
                        not p:AtomDS imp P(FALSE / p:AtomDS) .
rewrites: 33 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : p:AtomDS or P(p:AtomDS / p:AtomDS) ==
                        p:AtomDS or P(FALSE / p:AtomDS) .
rewrites: 20 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : not p:AtomDS or P(p:AtomDS / p:AtomDS) ==
                        not p:AtomDS or P(TRUE / p:AtomDS) .
rewrites: 27 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : p:AtomDS cos P(p:AtomDS / p:AtomDS) ==
                        p:AtomDS cos P(FALSE / p:AtomDS) .
rewrites: 22 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
==========================================
reduce in BOOL-DS-EXT : not p:AtomDS cos P(p:AtomDS / p:AtomDS) ==
                        not p:AtomDS cos P(TRUE / p:AtomDS) .
rewrites: 29 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true
```

We have shown that the equalities hold for any atomic proposition p and any Boolean expression P. Observe that we have indirectly used $T_{\mathrm{DS}}^{\mathrm{DEC}}$ to calculate at its meta-level.

Theorem 5 provides the foundation for a syntactic replacement of atomic propositions with constants in any Boolean expression without altering its semantic value. The optimizing equations (i.e., the generic simplifying equations in Definition 10 together with the equations replacing atomic propositions with constants in Theorem 5) work by inspecting a given proposition to find whether a replacement of an atomic proposition with a constant can be performed and then, if possible, applying the generic equations to simplify it.

To illustrate how the process is conducted, let us consider the example depicted in Fig. 3, where p, q, r and s are atomic propositions. The process has as input the proposition $q \wedge (((r \vee \mathsf{T}) \Leftarrow s \equiv \neg\mathsf{T}) \vee \neg p)$; a 'syntax tree' of this expression, with the expression itself at the top appears in column (a). In column (b), the replacement of q by T has taken place as a result of the first proposition in Theorem 5. Finally, in column (c), the result after applying various equations of E_{OPT} is shown. In this way, the proposition $q \wedge (((r \vee \mathsf{T}) \Leftarrow s \equiv \neg\mathsf{T}) \vee \neg p)$ has been simplified to a semantically equivalent one, namely $q \wedge ((\mathsf{T} \equiv \mathsf{F}) \vee \neg p)$, before using any decision procedure.

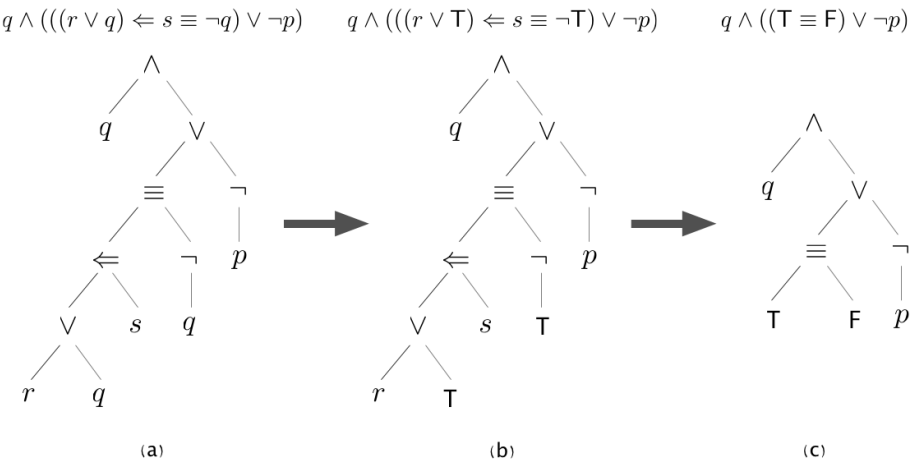

Fig. 3. (a) The expression $q \wedge (((r \vee \mathsf{T}) \Leftarrow s \equiv \neg\mathsf{T}) \vee \neg p)$ is depicted with a corresponding 'syntax tree'. (b) An equivalent expression to that of (a), but with replacement of q by T applied. (c) A semantically equivalent proposition to the one in (b) further simplified with the optimizing equations.

Observe that the 'inspection' process can be guaranteed to terminate, since any Boolean expression has finitely many symbols. However, the integrated process does not need to be confluent, since it preserves validity and the resulting proposition will be fed to a decision procedure which is both confluent and terminating.

We have developed an efficient inspection and substitution algorithm for Boolean expressions, characterized by binary function symbols having either an atomic proposition or the negation of an atomic proposition as one of its operands. Currently, we are studying how to extend the algorithm to the general case without compromising the efficiency of the technique. In Section 6 we present experiments in which the current specification of the algorithm is put under test.

6 Experiments

The objective of this section is to study experimentally the computational behavior of the four decision procedures as automatic decision tools when implemented in a high-performance rewrite engine with efficient support for AC rewriting such as Maude's [4]. We present the results of three experiments in which executable specifications of the four decision procedures in the Maude system were exercised against sets of propositional expressions. Each propositional expression was randomly generated, with a normal distribution for function symbols and variables, and then each propositional expression and its dual were automatically fed to each decision procedure[2]. Both, the *number of rewrites* (number of times an equation was used to simplify the input expression) and the *time* needed to simplify each input expression to its canonical form were recorded. Furthermore, each set of propositional expressions was parameterized by the number of variables and the height of the *syntax trees* of the propositions. The internal nodes of a syntax tree are function symbols with positive arity, whereas its leaves are atomic propositions or constant symbols. In the three experiments the *size* of a Boolean expression is the number of *nodes* of its syntax tree. In all three experiments, we used the *median* value instead of the *mean* value to analyze the gathered data.

6.1 Comparing the Four Decision Procedures

This experiment compares the performance of the four decision procedures with each other. One expects a decision procedure for propositional logic to be exponential in the size of its input, since SAT, which is one of the applications of these procedures, is an NP-Complete problem. Figure 4 depicts the average time required for each decision procedure to simplify sample propositions to their corresponding canonical forms as a function of their size. As already mentioned, the set of propositions used in this experiment was 'fair' in the sense that each proposition and its dual were given as inputs.

On average (using the median value), the pair of dual decision procedures for $T^{\mathrm{DEC}}_{\mathrm{DS}}$ and $T^{\mathrm{DEC}}_{\mathrm{BR}}$ performed 40% faster than the other pair of dual decision procedures for $T^{\mathrm{DEC}}_{\wedge/\equiv}$ and $T^{\mathrm{DEC}}_{\vee/\oplus}$. Moreover, the efficiency within each pair of dual decision procedures did not exceed the 1% difference.

6.2 Using Optimizing Equations

As described in Section 5, optimizing equations can be used to simplify each propositional expression prior to the execution of the decision procedures. This

[2] As seen in [20], due to the random nature of the inputs and the quite different performance of each procedure for each input, when we did not dualize the inputs, the average performance of dual procedures was substantially different. However, when we did feed each propositional expression and its dual to each of the procedures, the performance of each pair of dual procedures became almost identical.

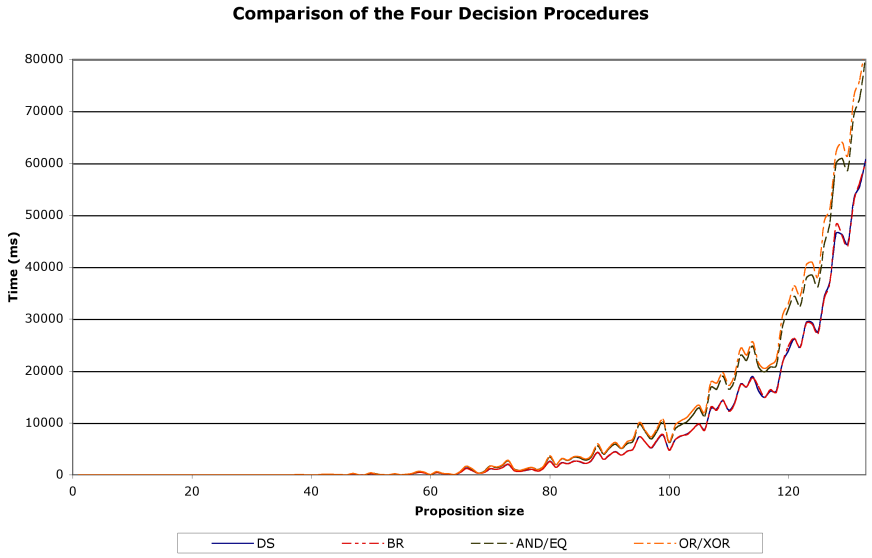

Fig. 4. Time taken by the decision procedures to simplify propositional expressions to their respective canonical forms as a function of the size of the propositions.

second experiment shows how optimizing equations improve the overall performance of the decision procedures. We have implemented an efficient executable specification of this technique for propositional expressions in which every binary function symbol has at least one atomic proposition or a negated atomic proposition as one of its operands. Figure 5 presents a comparison of the decision procedure $T_{\mathrm{DS}}^{\mathrm{DEC}}$ with and without optimizing equations enabled, in terms of the average number of rewrites as a function of the expression's size.

In this case, the dashed line is used to present the results for the execution of the decision procedure with the optimization technique disabled, while the continuous line describes those of the decision procedure with the technique enabled. The number of rewrites of the optimization phase are also included in the average in the case were the optimizing equations were enabled. In this experiment, the number of rewrites required to simplify the propositions was approximately 24 times less when using the technique of optimizing equations. In another experiment, in the same spirit, the optimizing equations with Hsiang's $T_{\mathrm{BR}}^{\mathrm{DEC}}$ decision procedure performed up to 29 times faster than the same procedure with the optimizing equations disabled [20].

6.3 Comparison with SAT-Solving

In this third experiment, the performance of $T_{\mathrm{DS}}^{\mathrm{DEC}}$ with optimizing equations enabled is compared against that of a DPLL(T) SAT-Solver implemented in Maude by Joe Hendrix. As the latter tool is optimized for SAT-Solving, this

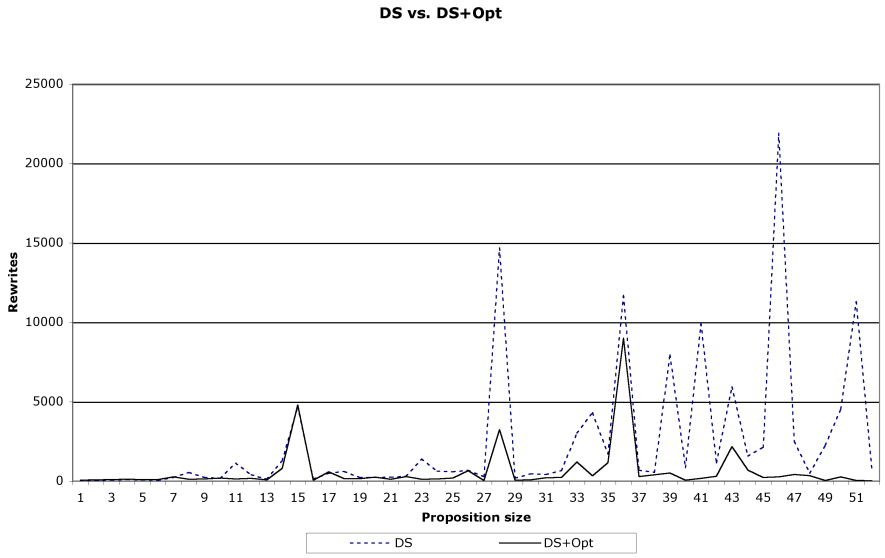

Fig. 5. Comparison of $T_{\mathrm{DS}}^{\mathrm{DEC}}$ with and without using optimizing equations in terms of the number of rewrites as a function of the proposition's size.

experiment uses the decision procedure to solve the satisfiability problem, that is, to determine if the canonical form of a given expression is different from F.

Similar to the previous experiments, the input data consisted of randomly generated propositional expressions in which every binary function symbol had at least one variable, or constant, or either a negated variable or negated constant, as one of its operands. The DPLL(T) SAT-Solver was equipped with a utility to convert arbitrary propositional expressions into semantically equivalent ones in Conjunctive Normal Form (CNF) format, since this is required as part of DPLL's input format.

Figure 6 presents a comparison of the tools in terms of the average number of rewrites needed to solve the satisfiability problem. In both cases, the average number of rewrites includes those rewrites needed to pre-process each proposition. That is, we include both the number of rewrites employed by the optimizing equations in the case of the $T_{\mathrm{DS}}^{\mathrm{DEC}}$, and the number of rewrites to convert to CNF in the case of the SAT-Solver. Note that the average number of rewrites is plotted within a logarithmic scale. Overall, $T_{\mathrm{DS}}^{\mathrm{DEC}}$ with optimizing equations enabled performed 125 times faster that the DPLL(T) SAT-Solver.

7 Conclusions

We have presented four Boolean equational theories giving rise to four rewriting-based decision procedures for propositional logic, have proved their mathematical

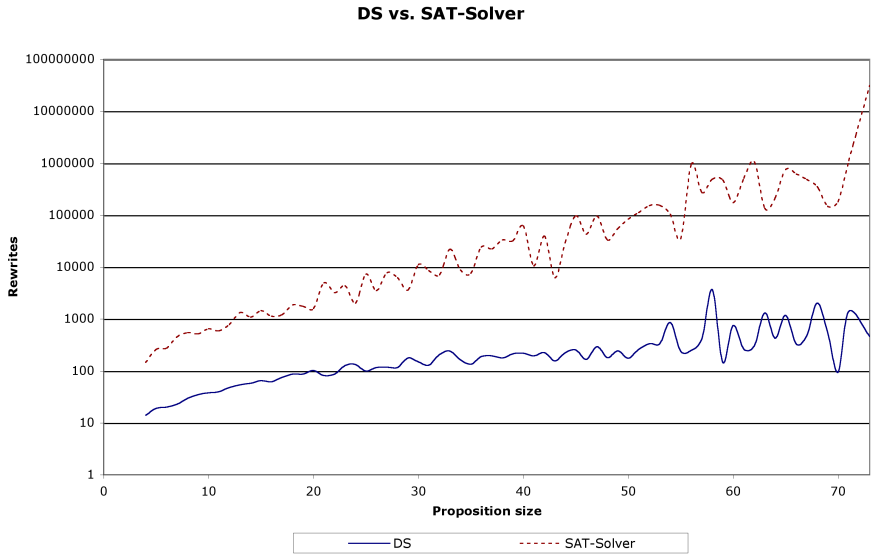

Fig. 6. Comparison of $T_{\mathrm{DS}}^{\mathrm{DEC}}$ with optimizing equations enabled against a DPLL(T) SAT-Solver implemented in Maude.

and operational correctness, and have shown how they can be further sped up by the use of optimizing equations. We have also presented experimental results suggesting that these procedures, when implemented on a high-performance rewrite engine, have very good efficiency and outperform a DPLL(T)-based SAT-solver. Some research directions that seem worth following include: (i) exploiting these decision procedures as components in "deduction modulo" inference systems; (ii) studying if the use of context-sensitive rewriting such as in Maude's functional evaluation strategies [4] can further increase procedure performance; and (iii) carrying out a more extensive comparison with other SAT-solvers to gain a more comprehensive performance evaluation.

References

1. L. Bachmair and N. Dershowitz. Inference rules for rewrite-based first-order theorem proving. In *LICS*, pages 331–337. IEEE Computer Society, 1987.
2. H. Barendregt and E. Barendsen. Autarkik computations and formal proofs. *Journal of Automated Reasoning*, 28(3):321–336, 2002.
3. D. Benanav, D. Kapur, and P. Narendran. Complexity of matching problems. *J. Symb. Comput.*, 3(1/2):203–216, 1987.
4. M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude*. Springer LNCS Vol. 4350, 2007. To appear.
5. E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

6. G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *J. Autom. Reasoning*, 31(1):33–72, 2003.
7. S. M. Eker. Associative-commutative rewriting on large terms. In *Rewriting Techniques and Applications (RTA'03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 14–29. Springer-Verlag, 2003.
8. A. Foret. Rewrite rule systems for modal propositional logic. *J. Log. Program.*, 12(3&4):281–298, 1992.
9. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer, 2000.
10. D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Springer Verlag, 1993.
11. D. Gries and F. B. Schneider. Equational propositional logic. *Inf. Process. Lett.*, 53(3):145–152, 1995.
12. J. Hendrix, H. Ohsaki, and J. Meseguer. Sufficient completeness checking with propositional tree automata. Technical Report UIUCDCS-R-2005-2635, University of Illinois Urbana-Champaign, 2005.
13. J. Hsiang. *Topics in automated theorem proving and program generation*. PhD thesis, University of Illinois at Urbana-Champaign, 1982.
14. N. Jacobson. *Basic algebra. I*. W. H. Freeman and Co., San Francisco, Calif., 1974.
15. J.-P. Jouannaud, editor. *Rewriting Techniques and Applications, First International Conference, RTA-85, Dijon, France, May 20-22, 1985, Proceedings*, volume 202 of *Lecture Notes in Computer Science*. Springer, 1985.
16. Laboratoire de Recherche en Informatique. The CiME 2.0 System (http://cime.lri.fr/).
17. F. W. Lawvere. Functorial semantics of algebraic theories. *Proceedings, National Academy of Sciences*, 50:869–873, 1963. Summary of Ph.D. Thesis, Columbia University.
18. V. Lifschitz. On calculational proofs. Unpublished, 1998.
19. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd. Edition*, pages 1–87. Kluwer Academic Publishers, 2002. First published as SRI Tech. Report SRI-CSL-93-05, August 1993.
20. C. Rocha and J. Meseguer. Five isomorphic boolean theories and four equational decision procedures. Technical Report (to be indexed), University of Illinois at Urbana-Champaign, 2007.
21. C. Rocha and J. Meseguer. A rewriting decision procedure for Dijkstra-Scholten's monadic first-order logic. In *Submited to Segundo Congreso Colombiano de Computación*, 2007.
22. G. F. Simmons. *Introduction to topology and modern analysis*. McGraw-Hill Book Co., Inc., New York, 1963.
23. P. Viry. Equational rules for rewriting logic. *Theoretical Computer Science*, 285:487–517, 2002.

A Executable Specification in Maude of the Four Decision Procedures

A.1 $T_{\mathbf{DS}}^{\mathbf{DEC}}$

```
fmod BOOL-DS is
```

```
   sort BoolDS .
   ops TRUE FALSE : -> BoolDS [ctor] .
   op _equ_ : BoolDS BoolDS -> BoolDS [ctor assoc comm prec 80] .
   op _or_ : BoolDS BoolDS -> BoolDS [ctor assoc comm prec 50] .
   op not_ : BoolDS -> BoolDS [prec 10] .
   op _and_ : BoolDS BoolDS -> BoolDS [assoc comm prec 50] .
   op _imp_ : BoolDS BoolDS -> BoolDS [prec 60] .
   op _cos_ : BoolDS BoolDS -> BoolDS [prec 60] .
   vars P Q R : BoolDS .
   eq P equ P = TRUE .
   eq P equ TRUE = P .
   eq P or TRUE = TRUE .
   eq P or FALSE = P .
   eq P or P = P .
   eq P or ( Q equ R ) = P or Q equ P or R .
   eq P and Q = P equ Q equ P or Q .
   eq not P = P equ FALSE .
   eq P imp Q = P or Q equ Q .
   eq P cos Q = P or Q equ P .
endfm
```

A.2 $T_{\mathbf{BR}}^{\mathbf{DEC}}$

```
fmod BOOL-BR is
   sort BoolBR .
   ops TRUE FALSE : -> BoolBR [ctor] .
   op _neq_ : BoolBR BoolBR -> BoolBR [ctor assoc comm prec 80] .
   op _and_ : BoolBR BoolBR -> BoolBR [ctor assoc comm prec 50] .
   op not_ : BoolBR -> BoolBR [prec 10] .
   op _or_ : BoolBR BoolBR -> BoolBR [assoc comm prec 50] .
   op _imp_ : BoolBR BoolBR -> BoolBR [prec 60] .
   op _cos_ : BoolBR BoolBR -> BoolBR [prec 60] .
   vars P Q R : BoolBR .
   eq P neq P = FALSE .
   eq P neq FALSE = P .
   eq P and FALSE = FALSE .
   eq P and TRUE = P .
   eq P and P = P .
   eq P and ( Q neq R ) = P neq Q neq P and R .
   eq P or Q = P neq Q neq P and Q .
   eq not P = P neq TRUE .
   eq P imp Q = P and Q neq P neq TRUE .
   eq P cos Q = P and Q equ Q neq TRUE .
endfm
```

A.3 $T_{\wedge/\equiv}^{\mathbf{DEC}}$

```
fmod BOOL-AE is
   sort BoolAE .
```

```
   ops TRUE FALSE : -> BoolAE [ctor] .
   op _equ_ : BoolAE BoolAE -> BoolAE [ctor assoc comm prec 80] .
   op _and_ : BoolAE BoolAE -> BoolAE [ctor assoc comm prec 50] .
   op not_ : BoolAE -> BoolAE [prec 10] .
   op _or_ : BoolAE BoolAE -> BoolAE [assoc comm prec 50] .
   op _imp_ : BoolAE BoolAE -> BoolAE [prec 60] .
   op _cos_ : BoolAE BoolAE -> BoolAE [prec 60] .
   vars P Q R : BoolAE .
   eq P equ P = TRUE .
   eq P equ TRUE = P .
   eq P and TRUE = P .
   eq P and FALSE = FALSE .
   eq P and P = P .
   eq P and ( Q equ R ) = P equ P and Q equ P and R .
   eq P or Q = P equ Q equ P and Q .
   eq not P = P equ FALSE .
   eq P imp Q = P and Q equ P .
   eq P cos Q = P and Q equ Q .
endfm
```

A.4 $T^{\mathbf{DEC}}_{\vee/\oplus}$

```
fmod BOOL-OX is
   sort BoolOX .
   ops TRUE FALSE : -> BoolOX [ctor] .
   op _neq_ : BoolOX BoolOX -> BoolOX [ctor assoc comm prec 80] .
   op _or_ : BoolOX BoolOX -> BoolOX [ctor assoc comm prec 50] .
   op not_ : BoolOX -> BoolOX [prec 10] .
   op _and_ : BoolOX BoolOX -> BoolOX [assoc comm prec 50] .
   op _imp_ : BoolOX BoolOX -> BoolOX [prec 60] .
   op _cos_ : BoolOX BoolOX -> BoolOX [prec 60] .
   vars P Q R : BoolOX .
   eq P neq P = FALSE .
   eq P neq FALSE = P .
   eq P or TRUE = TRUE .
   eq P or FALSE = P .
   eq P or P = P .
   eq P or ( Q neq R ) = P neq P neq Q neq P and R .
   eq P and Q = P neq Q neq P or Q .
   eq not P = P neq TRUE .
   eq P imp Q = P or Q neq Q neq TRUE .
   eq P cos Q = P or Q equ P neq TRUE .
endfm
```