A BDD Representation for Positive Equational Formulas

Wenxin Song, Eugene W. Stark[⋆]

Department of Computer Science
State University of New York
at Stony Brook
Stony Brook, NY 11794 USA

Abstract. A *positive equational formula* (PEF) is a negation-free open formula in the language of pure equality. In this paper, we present a normal form for positive equational formulas that lends itself readily to the representation of such formulas using ordered *binary decision diagrams* (BDDs) whose nodes are labeled by equations. In contrast to previous work of Groote and van de Pol that treated the larger class of equational formulas with negation, we show that our normal forms for PEFs are unique in the sense that two normal forms are logically equivalent if and only if they are identical. This result means that a BDD-based representation of our normal forms can implement equivalence checking for PEFs in constant time, as well as avoid space inefficiency that could otherwise result from storing multiple syntactically distinct variants of a formula. We present a recursive algorithm that traverses an arbitrary PEF given as input and builds an equivalent normal form in a bottom-up fashion. In addition, we apply ideas from the reduction algorithm to obtain bottom-up, normal-form-preserving algorithms for various logical operations on PEFs.

1 Introduction

A *positive equational formula* (PEF) is a negation-free open formula in the language of pure equality. In this paper, we present a normal form for positive equational formulas that lends itself readily to the representation of such formulas using ordered *binary decision diagrams* (BDDs) whose nodes are labeled by equations. Specifically, we represent such formulas as *decision forms*, where a decision form is either **T** or **F** or has the form $(x = y \wedge \phi_1) \vee \phi_2$, where $x = y$ is an equation and ϕ_1 and ϕ_2 are (recursively) decision forms. Decision forms that satisfy certain syntactic conditions that we identify are called *normal forms*, and we show that normal forms are canonical in the sense that two normal forms are logically equivalent if and only if they are identical. This result means that normal forms can be stored using a maximal structure-shared representation that permits equivalence checking to be performed in constant time. Such

[⋆] Authors' E-mail addresses: {`wenxin,stark`}`@cs.sunysb.edu` a representation amounts to a BDD in which equations, rather than propositional variables, are used to label the nodes. We present a recursive algorithm for reducing an arbitrary PEF to normal form. In addition, we apply ideas from the reduction algorithm to obtain normal-form-preserving algorithms for various logical operations on PEFs.

We note at the outset that previous work on BDD representations for equational formulas has typically been motivated by applications in hardware verification. In such applications, verification of a hardware design is accomplished by checking the satisfiability or validity of an equational formula (containing negation, in general) extracted from the design. Previous work has typically focused on representations for such formulas that lead to efficient satisfiability or validity checking algorithms. However, for the positive equational formulas we consider in this paper, satisfiability and validity checking is not difficult, because a positive equational formula is valid if and only if it is satisfied by the identity relation, and it is satisfiable if and only if it is satisfied by the universal relation. Thus, checking the satisfiability or validity of such a formula reduces to simply checking whether the formula is satisfied by the identity relation or the universal relation, respectively. This checking can be performed efficiently (in linear time).

In contrast, our own motivations for considering this problem arise from our study of an approach to a certain problem of nonlinear pattern matching on terms. In contrast to linear patterns, in which each variable is permitted to appear at most once, in nonlinear patterns there can be multiple occurrences of a single variable and in matching such a pattern against a target term all the occurrences of a particular variable in the pattern must be matched to identical subterms of the target term. Roughly speaking, in our approach we separate nonlinear patterns into a linear part, represented by an MTBDD, and associated equational constraints, placed at the leaf nodes of the MTBDD. The results of our paper provide the canonical form necessary for positive equational formulas to appear at the leaves of an MTBDD. In particular, efficient equivalence checking is needed, and as positive equational formulas represent finite (but possibly exponentially large) unions of equivalence relations, a naive algorithm for equivalence checking will not suffice. Our results also provide the operations on canonical forms necessary to perform common manipulations (such as union and join) on patterns in MTBDD form.

Since a non-linear pattern can also be thought of as representing a possibly infinite set of ground terms that satisfy a predicate, we also anticipate applications to symbolic execution of logic programs and theorem-proving.

1.1 Previous Work

Binary Decision Diagrams (BDDs) [Bry86,Bry92] are efficient data structures for representing boolean formulas. BDDs have been widely used in formal verification and there has been significant interest in extending their applicability from boolean formulas to more expressive logics that incorporate equations and/or function symbols. In recent years, there have been three approaches to applying BDDs to equational formulas. In the first approach, domain variables are eliminated in favor of a collection of boolean variables, and equations involving the domain variables are replaced by equivalent boolean formulas. The simplest version of this approach is based on the straightforward observation that a variable ranging over a domain of size n can be replaced by a collection of $\lceil \log n \rceil$ boolean variables. However, this naive idea tends to result in an excessively large number of boolean variables. Better results can be obtained by exploiting the *finite domain property* of equational formulas; namely, that an equational formula is satisfiable if and only if it is satisfiable in a suitably large finite domain (in fact, the domain need be no larger than the number of distinct variables occurring in the formula). Pnueli, *et al* [PRSS99] present an algorithm to obtain small domains for each of the domain variables to reduce the total number of boolean variables in the resulting BDDs. Bryant, *et al* [BGV99] also reduced the state space dramatically by exploiting structural "positivity" properties of the formulas.

The second approach to applying BDDs to equational formulas involves directly encoding equations $x_i = x_j$ between domain variables as boolean variables $v_{i,j}$. The first proposal along these lines was by Goel, *et al* [GSZ+98]. The difficulty in this approach is in handling transitivity constraints, which have the form $v_{i,j} \wedge v_{j,k} \rightarrow v_{i,k}$, and which are not captured in a BDD representation. Goel showed that the problem of deciding satisfiability in the context of transitivity constraints is NP-complete. Bryant and Velev [BV00] proposed an approach to this satisfiability problem. In their method, a BDD B is built for the original formula as proposed by Goel, and another BDD B' is built to represent transitivity constraints over the *true support* of B, where the true support of a function f is the set of variables on which f depends. Therefore, in BDD $B \wedge B'$, any satisfying assignment obeys transitivity constraints.

Groote and van de Pol [GvdP00] propose a third approach using ordered *Equational Binary Decision Diagrams* (EQ-BDDs) that automatically incorporate transitivity constraints. An EQ-BDD is a DAG with two terminal nodes *0* and *1*, where each non-terminal node r is labeled by if-then-else expression $ITE(g, r_1, r_2)$. Here g is a *guard*, r_1 is the "then" ("high," or 1) branch, and r_2 is the "else" ("low," or 0) branch. A guard is either a propositional variable p (from set P) or an equation of the form $x = y$, where x and y are domain variables (from set V). A total order $\succ$ is imposed on $P \cup V$, and extended lexicographically to guards. Terminal node **0** represents formula **F** and node **1** represents formula **T**. Non-terminal node $ITE(g, r_1, r_2)$ represents a formula $(g \wedge \phi_{r_1}) \vee (\neg g \wedge \phi_{r_2})$, where ϕ_{r_1} and ϕ_{r_2} are formulas represented by r_1 and r_2 respectively. Groote and van de Pol presented a term rewriting system that reduces each EQ-BDD to an equivalent EQ-BDD that is ordered with respect to $\succ$. The term rewriting system automatically incorporates transitivity by ensuring that in a reduced EQ-BDD $ITE(x = y, \phi_1, \phi_2)$ all occurrences of y in ϕ_1 have been replaced by x.

Unlike standard BDDs, ordered EQ-BDDs are not canonical; that is, logically equivalent ordered EQ-BDDs need not be identical. However, Groote and van de Pol do show that ordered EQ-BDDs are still adequate for satisfiability and validity checking in the sense that **0** is the only contradictory ordered EQ-BDD and **1** is the only tautological one. An additional issue with ordered EQ-BDDs is the fact that the classical APPLY algorithm used to perform logical operations ($\wedge$, $\vee$, *etc.*) on standard BDDs fails to preserve orderedness when applied to ordered EQ-BDDs. To compensate for this, Groote and van de Pol propose an algorithm that, if applied repeatedly to a "simplified" EQ-BDD, is guaranteed to produce an equivalent ordered EQ-BDD after a finite number of applications. A potential disadvantage of this algorithm is its "top-down" nature, which constructs a series of non-ordered EQ-BDDs on its way to the final ordered result. This contrasts with the normal bottom-up approach, in which operations are applied only to already reduced BDDs. However, Groote and van de Pol report that careful implementation seems to avoid serious inefficiencies.

In subsequent work along these lines, EQ-BDDs are extended by Badban and van de Pol [BvdP05], to allow equations with zero and successor, and by van de Pol and Tveretin [vdPT05] to allow equalities between ground terms. Both of these extensions share with the original EQ-BDD work the failure of the BDD representations to be canonical.

In the present paper, we follow an approach similar to that of [GvdP00], except that we restrict our attention at the outset to *positive* (*i.e.* negation-free) equational formulas. We characterize a class of normal forms for PEFs such that each normal form is either **T** or **F** or has the form $(x = y \wedge \phi_1) \vee \phi_2$, where $x = y$ is an equation and ϕ_1 and ϕ_2 are again normal forms (and certain ordering and other syntactic restrictions are satisfied). Such normal forms have a direct structure-shared representation as ordered EQ-BDDs. In contrast to the case of formulas with negation, normal forms of PEFs are canonical in the sense that two normal forms are logically equivalent if and only if they are identical. This fact permits a maximally structure-shared implementation using a unique table as is done for standard BDDs. Moreover, our algorithm for reducing an arbitrary decision form to a normal form is bottom-up in the sense that only normal forms are ever constructed (from previously existing normal forms) during execution. In addition, we apply ideas used in the reduction algorithm to obtain bottom-up, normal-form-preserving algorithms for various logical operations on PEFs.

1.2 Outline of the Paper

In Section 2, we give basic definitions and notations for positive equational formulas. In Section 3, we present the concept of decision forms, which can be viewed as decision trees and correspond directly to EQ-BDDs. However, decision forms are not canonical in the sense that logically equivalent decision forms need not be identical. In Section 4, we identify additional conditions on decision forms sufficient to ensure that logically equivalent decision forms are identical. Since the defining conditions for normal forms do not directly suggest an algorithm for converting a decision form into a logically equivalent normal form, in Section 5 we obtain alternative recursive conditions that are sufficient to guarantee that a decision form is a normal form. Based on these sufficient conditions, an algorithm REDUCE for reducing decision forms to normal form is presented in

Section 6 and its correctness is proved. In Section 7, we present other algorithms that implement various logical operations on PEFs based on ideas from the reduce algorithm. A brief discussion about the implementation of normal forms is presented in Section 8. Finally, some conclusions are given in Section 9.

2 Positive Equational Formulas

A *positive equational formula* (PEF) is a negation-free open formula in the language of pure equality. That is, a PEF is a formula built up from the *atomic propositions* **T** (truth) and **F** (falsity) and *equations* $x = y$, via the boolean connectives $\wedge$ and $\vee$. Here x and y are *variables* drawn from a countably infinite set $\mathcal{V}$, which we take as fixed. A traditional formulation of the semantics of PEFs would define when it is that a PEF ϕ whose variables are among $\{x_1, \ldots, x_n\}$ is *satisfied* by a sequence of elements $a_1, \ldots, a_n$ of a set A (notation $A \models \phi[a_1, \ldots, a_n]$. For our purposes, it is more convenient for us to define satisfaction as a relation $R \models \phi$ between a PEF ϕ and an equivalence relation R on $\mathcal{V}$, so that the meaning of a PEF ϕ can be identified with the set of equivalence relations on $\mathcal{V}$ that satisfy it. Formally, $R \models \phi$ (read R *satisfies* ϕ or R *entails* ϕ) is defined recursively as follows: $R \models x = y$ holds exactly when $(x, y) \in R$, $R \models \phi \wedge \psi$ holds exactly when *both* $R \models \phi$ and $R \models \psi$ hold, and $R \models \phi \vee \psi$ holds exactly when *one of* $R \models \phi$ and $R \models \psi$ holds. PEFs ϕ and ψ are *logically equivalent* (or simply, *equivalent*) if for all equivalence relations R on $\mathcal{V}$, $R \models \phi$ if and only if $R \models \psi$.

When speaking about equivalence relations R on $\mathcal{V}$, if no confusion can result we will often identify an element (x, y) of R with the corresponding equation $x = y$, and we will often say that R *entails* $x = y$ to mean that $(x, y) \in R$. If R is an equivalence relation on $\mathcal{V}$ and $x = y$ is an equation, then we use $R + \{x = y\}$ to denote the least equivalence relation containing R and (x, y); *i.e.* the transitive closure of $R \cup \{(x, y)\}$. Similarly, if R and R' are equivalence relations we write $R + R'$ to denote the least equivalence relation containing R and R'.

An obvious consequence of the semantics of PEFs is *monotonicity*: If $R \models \phi$ then $R' \models \phi$ for all R' such that $R \subseteq R'$. Another obvious property is the following: $R' \models \phi$ if and only if $R \models \phi$ for some $R \subseteq R'$ that is *finitary* in the sense that each equivalence class has at most finitely many elements and all but finitely many of the equivalence classes of R are singleton sets.

We say that an equivalence relation R *minimally satisfies* ϕ, and we write $R \models_{\min} \phi$, if $R \models \phi$ and whenever $R' \subseteq R$ is such that $R' \models \phi$, then $R' = R$. The following easily established results are important for us:

Lemma 1. *Suppose ϕ is a PEF and R is an equivalence relation such that $R \models \phi$. Then there exists an equivalence relation $R' \subseteq R$ such that $R' \models_{\min} \phi$.*

Proof. The set S of equivalence relations R' such that $R' \subseteq R$ and $R' \models \phi$ is nonempty (it contains R) and is partially ordered by containment. According to Hausdorff's Maximal Principle, if X is a partially ordered set then X has a maximal linearly ordered subset. Hence, the set S contains a maximal linearly ordered subset C, which is nonempty because it must contain R. Moreover, since R is finitary, the set S is finite, hence C is also finite. Let R' be the least element of C, then $R' \models_{\min} \phi$. □

Lemma 2. *PEFs ϕ and ψ are equivalent if and only if whenever $R \models_{\min} \phi$ then $R \models \psi$ and vice versa.*

Proof. Suppose ϕ and ψ are equivalent. It is then obvious that $R \models \phi$ if and only if $R \models \psi$. Since $R \models_{\min} \phi$ implies $R \models \phi$, it follows that $R \models_{\min} \phi$ implies $R \models \psi$. Similarly, $R \models_{\min} \psi$ implies $R \models \phi$.

Conversely, suppose $R \models_{\min} \phi$ implies $R \models \psi$ and $R \models_{\min} \psi$ implies $R \models \phi$. Suppose R is an equivalence relation such that $R \models \phi$. Then by Lemma 1, there exists $R' \subseteq R$ such that $R' \models_{\min} \phi$. By hypothesis, $R' \models_{\min} \phi$ implies $R' \models \psi$, hence $R' \models \psi$. Symmetric reasoning shows that if $R \models \psi$ implies $R \models \phi$, hence ϕ and ψ are equivalent. □

3 Decision Forms

The set of *decision forms* is the least set of PEFs that contains **T** and **F** and is closed under the following formation rule:

– If ϕ_1 and ϕ_2 are decision forms and $x = y$ is an equation, then $(x = y \wedge \phi_1) \vee \phi_2$ is also a decision form.

The following is easily established:

Proposition 1. *There exist algorithms that:*

1. *Given decision forms ϕ_1 and ϕ_2, output a decision form ϕ that is equivalent to $\phi_1 \vee \phi_2$.*
2. *Given decision forms ϕ_1 and ϕ_2, output a decision form ϕ that is equivalent to $\phi_1 \wedge \phi_2$.*
3. *Given an arbitrary PEF ϕ, output a decision form ϕ' that is equivalent to ϕ.*

Algorithm Convert, shown in Figure 1, converts an arbitrary PEF ϕ to a logically equivalent decision form. The algorithm scans a formula twice such that the first scan (function Simplify) eliminates occurrences of **T** from the formula and the second scan (function Convert$'$) converts a formula without any **T**'s to a decision form. Function Simplify runs in time linear in the size of its argument and function Convert$'$ runs in time linear in the size of its first argument, hence Convert(ϕ) runs in time linear in the size of ϕ.

Lemma 3 (Correctness of Convert). *Suppose ϕ is an arbitrary PEF. Then* Convert(ϕ) *terminates, and it returns a decision form ψ that is logically equivalent to ϕ.*

```
fun $\textsc{Convert}(\phi) =$
  let fun $\textsc{Simplify}(\mathbf{T}) = \mathbf{T}$
        $\mid \textsc{Simplify}(\mathbf{F}) = \mathbf{F}$
        $\mid \textsc{Simplify}(x = y) = \ x = y$
        $\mid \textsc{Simplify}(\phi_1 \vee \phi_2) =$
          let $\phi_1' = \textsc{Simplify}(\phi_1)$
              $\phi_2' = \textsc{Simplify}(\phi_2)$
          in
            if $\phi_1' = \mathbf{T}$ orelse $\phi_2' = \mathbf{T}$
            then $\mathbf{T}$
            else $\phi_1' \vee \phi_2'$
          end
        $\mid \textsc{Simplify}(\phi_1 \wedge \phi_2) =$
          let $\phi_1' = \textsc{Simplify}(\phi_1)$
              $\phi_2' = \textsc{Simplify}(\phi_2)$
          in
            if $\phi_1' = \mathbf{T}$ then $\phi_2'$
            else if $\phi_2' = \mathbf{T}$ then $\phi_1'$
            else $\phi_1' \wedge \phi_2'$
          end

      fun $\textsc{Convert}'(\mathbf{F})\ (t, f) = \mathbf{F}$
        $\mid \textsc{Convert}'(x = y)\ (t, f) = \ (x = y \wedge t) \vee f$
        $\mid \textsc{Convert}'(\phi_1 \vee \phi_2)\ (t, f) = \textsc{Convert}'\ \phi_1\ (t, \textsc{Convert}'\ \phi_2\ (t, f))$
        $\mid \textsc{Convert}'(\phi_1 \wedge \phi_2)\ (t, f) = \textsc{Convert}'\ \phi_1\ (\textsc{Convert}'\ \phi_2\ (t, F), f)$

      $\phi' = \textsc{Simplify}(\phi)$
  in
    if $\phi' = \mathbf{T}$ then $\mathbf{T}$
    else $\textsc{Convert}'\ \phi'\ (\mathbf{T}, \mathbf{F})$
  end
```

Fig. 1. Function CONVERT

Proof. Since the recursive calls in SIMPLIFY and CONVERT$'$ are always made on proper subformulas, the termination of CONVERT is clear.

We first prove by induction that SIMPLIFY(ϕ) returns a logically equivalent formula ϕ' that is either just **T** or else a formula that does not contain any **T**'s. It is obvious that this is true in the basis cases in which ϕ is **T** or **F** or $x = y$. Now we consider the following cases in the induction step:

1. ϕ is $\phi_1 \vee \phi_2$. Let $\phi_1' =$ SIMPLIFY and $\phi_2' =$ SIMPLIFY(ϕ_2). By induction, ϕ_1' is logically equivalent to ϕ_1 and ϕ_2' is logically equivalent to ϕ_2. If $\phi_1' = \mathbf{T}$ or $\phi_2' = \mathbf{T}$ then $\phi' = \mathbf{T}$. Otherwise, $\phi' = \phi_1' \vee \phi_2'$. It then follows that ϕ' is equivalent to ϕ, and ϕ' is either **T** or a formula that does not contain any **T**'s.

2. ϕ is $\phi_1 \wedge \phi_2$. Let $\phi'_1 = $ SIMPLIFY and $\phi'_2 = $ SIMPLIFY(ϕ_2). By induction, ϕ'_1 is logically equivalent to ϕ_1 and ϕ'_2 is logically equivalent to ϕ_2. If $\phi'_1 = \mathbf{T}$ then $\phi' = \phi'_2$. If $\phi'_2 = \mathbf{T}$ then $\phi' = \phi'_1$. Otherwise, $\phi' = \phi'_1 \wedge \phi'_2$. It then follows that ϕ' is equivalent to ϕ, and ϕ' is either $\mathbf{T}$ or a formula that does not contain any $\mathbf{T}$'s.

We next prove by induction that CONVERT$'(\phi)$ (t, f) returns a decision form ϕ' that is logically equivalent to $(\phi \wedge t) \vee f$ provided ϕ is a formula without $\mathbf{T}$'s and t and f are decision forms. It is obvious that this is true in the basis cases in which ϕ is either $\mathbf{F}$ or $x = y$. Now we consider the following cases in the induction step:

1. ϕ is $\phi_1 \vee \phi_2$. Let $f' = $ CONVERT$'$ ϕ_2 (t, f). By induction, f' is a decision form that is logically equivalent to $(\phi_2 \wedge t) \vee f$. Let $\phi' = $ CONVERT$'$ ϕ_1 (t, f'). By induction, ϕ' is a decision form that is logically equivalent to $(\phi_1 \wedge t) \vee f'$. It then follows that ϕ' is logically equivalent to $(\phi_1 \wedge t) \vee (\phi_2 \wedge t) \vee f$, and is logically equivalent to $((\phi_1 \vee \phi_2) \wedge t) \vee f$. Hence ϕ' is logically equivalent to ϕ.
2. ϕ is $\phi_1 \wedge \phi_2$. Let $t' = $ CONVERT$'$ ϕ_2 (t, F). By induction, t' is a decision form that is logically equivalent to $(\phi_2 \wedge t) \vee F$. Let $\phi' = $ CONVERT$'$ ϕ_1 (t', f). By induction, ϕ' is a decision form that is logically equivalent to $(\phi_1 \wedge t') \vee f$. It then follows that ϕ' is logically equivalent to $(\phi_1 \wedge ((\phi_2 \wedge t) \vee F)) \vee f$, and is logically equivalent to $((\phi_1 \wedge \phi_2) \wedge t) \vee f$. Hence ϕ' is logically equivalent to ϕ.

Finally, let $\phi' = $ SIMPLIFY(ϕ). If $\phi' = \mathbf{T}$ then $\psi = \mathbf{T}$. Otherwise $\psi = $ CONVERT$'$ ϕ' $(\mathbf{T}, \mathbf{F})$. In either case, ψ is a decision form and ψ is logically equivalent to ϕ. □

There is an evident algorithm, shown in Figure 2 for determining whether an equivalence relation R satisfies a decision form. Algorithm SAT simply performs a depth-first traversal of the formula to be checked. For each subformula of the form $(x = y \wedge \phi_1) \vee \phi_2$ it first checks recursively if R satisfies ϕ_2. If that fails, it checks if R satisfies $x = y \wedge \phi_2$. Note that, when applied to a general decision form, algorithm SAT may make the same "query" $(x, y) \in R$ about the equivalence relation R multiple times. The normal forms discussed in the next section have an ordering property that makes it possible to check satisfaction without repeating any queries.

Lemma 4 (Correctness of SAT**).** *Suppose R is an equivalence relation and ϕ is a decision form. Then* SAT(R, ϕ) *terminates, and it returns* **true** *if and only if $R \models \phi$.*

Proof. Since recursive calls in SAT are always made on proper subformulas, termination is clear.

If there are no recursive calls, then either $\phi = \mathbf{F}$ and SAT(R, ϕ) returns **false** or else $\phi = \mathbf{T}$ and SAT(R, ϕ) returns **true**. In each case, it is obvious that **true** is returned if and only if $R \models \phi$.

```
fun SAT($R, \phi$) =
 if $\phi = \mathbf{F}$ then false
 else if $\phi = \mathbf{T}$ then true
 else
   let
     $(x = y \wedge \phi_1) \vee \phi_2$ be $\phi$
   in
     if SAT($R, \phi_2$) then true
     else if $(x, y) \in R$ then SAT($R, \phi_1$)
     else false
   end
```

Fig. 2. Function SAT

Otherwise, ϕ is $(x = y \wedge \phi_1) \vee \phi_2$. Suppose R does not satisfy ϕ. Then R does not satisfy ϕ_2 and if $R \models x = y$ then R also does not satisfy ϕ_1. So, by induction SAT(R, ϕ_2) returns **false**. If $(x, y) \in R$, then by induction SAT(R, ϕ_1) returns **false**, as does SAT(R, ϕ). If $(x, y) \notin R$, then SAT(R, ϕ) returns **false** in this case as well. Thus, if R does not satisfy ϕ, then SAT(R, ϕ) returns **false**.

Suppose $R \models \phi$. Then either $R \models \phi_2$ or else $R \models x = y \wedge \phi_1$. If $R \models \phi_2$, then by induction SAT(R, ϕ_2) returns **true**, as does SAT(R, ϕ). Otherwise, $(x, y) \in R$ and by induction SAT(R, ϕ_1) returns *true*, as does SAT(R, ϕ). $\square$

Decision forms have the property that, if they are viewed as trees, then each path from the root to a leaf determines an equivalence relation; namely the least equivalence relation generated by the set of equations encountered along the path. We call such equivalence relations *implicants* of ϕ. More formally, we define the set of implicants of a decision form ϕ recursively as follows:

1. The decision form $\mathbf{F}$ has no implicants.
2. The decision form $\mathbf{T}$ has the identity relation I on $\mathcal{V}$ as its only implicant.
3. An equivalence relation R is an implicant of a decision form $(x = y \wedge \phi_1) \vee \phi_2$ if and only if either R is an implicant of ϕ_2 or else $R = R' + \{x = y\}$ for some implicant R' of ϕ_1.

The following result relates the (more or less) syntactic notion of implicant to the semantic notion of satisfaction.

Lemma 5. *Let ϕ be a decision form. Then $R \models \phi$ if and only if $R' \subseteq R$ for some implicant R' of ϕ.*

Proof. We show by structural induction on ϕ that $R \models \phi$ implies $R' \subseteq R$ for some implicant R' of ϕ. If $\phi = \mathbf{F}$, then ϕ is unsatisfiable, and the result holds vacuously. If $\phi = \mathbf{T}$, then for any equivalence relation R we have $R \models \phi$, and the identity relation I is such that $I \subseteq R$ and $I \models \phi$. Now suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$. If $R \models \phi$, then either $R \models \phi_1$ or $R \models \phi_2$. By induction, either $R_1 \subseteq R$ for some implicant R_1 of ϕ_1 or $R_2 \subseteq R$ for some implicant R_2 of ϕ_2. In the first case, $R_1 + \{x = y\}$ is an implicant of ϕ that is contained in R. In the second case, R_2 is an implicant of ϕ that is contained in R.

Conversely, we show by structural induction on ϕ that if R contains some implicant R' of ϕ, then $R \models \phi$. If $\phi = \mathbf{F}$, then ϕ has no implicants and the result holds vacuously. If $\phi = \mathbf{T}$, then the only implicant of ϕ is the identity relation I. Clearly $I \models \mathbf{T}$, and since $I \subseteq R$ it follows that $R \models \mathbf{T}$. Now, suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$ and R contains some implicant R' of ϕ. Then either R' is an implicant of ϕ_2 or else $R' = R_1 + \{x = y\}$, where R_1 is an implicant of ϕ_1. If R' is an implicant of ϕ_2, then by induction $R' \models \phi_2$, hence also $R' \models \phi$. If $R' = R_1 + \{x = y\}$, where R_1 is an implicant of ϕ_1, then by induction $R' \models \phi_1$, and since R' entails $x = y$ it follows that $R' \models \phi$. Thus, in either case we have $R' \models \phi$. Since $R' \subseteq R$ it then follows that $R \models \phi$. $\square$

The following lemma, which relates implicants to minimal satisfaction, is established by a straightforward structural induction using Lemma 1.

Lemma 6. *Let ϕ be a decision form. If $R \models_{\min} \phi$ then R is an implicant of ϕ.*

Proof. By structural induction on ϕ. If ϕ is $\mathbf{F}$ then ϕ is unsatisfiable, so the result holds vacuously. If ϕ is $\mathbf{T}$, then the only equivalence relation R such such that $R \models_{\min} \mathbf{T}$ is the identity relation I, which by definition is an implicant of $\mathbf{T}$.

Now, suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$ and R is an equivalence relation such that $R \models_{\min} \phi$. Then clearly either $R \models_{\min} x = y \wedge \phi_1$ or $R \models_{\min} \phi_2$. If $R \models_{\min} \phi_2$, then by induction R is an implicant of ϕ_2, hence by definition it is also an implicant of ϕ. If $R \models_{\min} x = y \wedge \phi_1$, then $R \models \phi_1$. According to Lemma 5, there exists an equivalence relation R_1 such that $R_1 \subseteq R$ and R_1 is an implicant of ϕ_1. Now, $R_1 + \{x = y\} \subseteq R$ and $R_1 + \{x = y\} \models x = y \wedge \phi_1$, so because $R \models_{\min} x = y \wedge \phi_1$, we have that $R_1 + \{x = y\} = R$. But since R_1 is an implicant of ϕ_1, by definition $R_1 + \{x = y\}$, hence R, is an implicant of ϕ. $\square$

Note that decision forms are not canonical: two equivalent decision forms need not be identical. One reason for this is that the converse of Lemma 6 does not hold. That is, for an arbitrary decision form ϕ, even though each implicant R of ϕ obviously satisfies ϕ, it may not minimally satisfy ϕ because there could exist a strictly smaller implicant $R' \subset R$ associated with a different path in ϕ. Other reasons why decision forms are not canonical have to do with the fact that equations may appear in differing orders along the paths through ϕ, and that some paths through ϕ may contain equations that are redundant in the sense that they are logical consequences of other equations appearing along the same path. The objective of the next section is to identify additional conditions on decision forms sufficient to ensure they are canonical.

4 Normal Forms

From now on, we assume that the set $\mathcal{V}$ of variables is equipped with a total ordering, which we denote by $\succ$. We say that an equation $x = y$ is *oriented* if

$x \succ y$, and we consider only oriented equations. In particular, we only permit oriented equations to appear in formulas; one consequence of this is that the formula **T** becomes essential, as it is no longer possible to write $x = x$. In addition, we regard an equivalence relation R as consisting only of oriented equations. There is no loss of generality in this because any equivalence relation is completely determined by its oriented elements. We extend the ordering $\succ$ to equations lexicographically: $x = y \succ x' = y'$ if and only if either $x \succ x'$ or else $x = x'$ and $y \succ y'$.

A decision form is a *normal form* if it is either **T** or **F**, or else it has the form $(x = y \wedge \phi_1) \vee \phi_2$ and the following five conditions all hold:

N1: ϕ_1 is a normal form that is not **F**.
N2: ϕ_2 is a normal form that is not **T**.
N3: y does not appear in ϕ_1.
N4: If $x' = y'$ is an equation contained in some implicant of ϕ_1 or ϕ_2, then $x = y \succ x' = y'$.
N5: If R is an implicant of ϕ that contains $x = y$, then $R \not\models \phi_2$.

With respect to conditions (N4) and (N5), note that it is possible for an implicant of a decision form to entail an equation without that equation explicitly occurring in the decision form. This is because implicants of a decision form are obtained by taking transitive closures of the sets of equations that actually appear along paths.

The following properties of normal forms are established by structural induction.

Lemma 7. *For all normal forms ϕ, if ϕ is equivalent to* **T** *(resp.* **F***) then ϕ equals* **T** *(resp.* **F***).*

Proof. By structural induction on normal form ϕ. If ϕ is **T** or **F**, then clearly it is equivalent to **T** (resp. **F**) if and only if it is equal to **T** (resp. **F**), so the result holds in this case. For the remainder of the proof, suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$.

If ϕ is equivalent to **T**, then $R \models \phi$ holds for every equivalence relation R. In particular this must be true for the identity relation I. But because $x = y$ is an oriented equation, hence x and y are distinct variables, it cannot be the case that $I \models x = y \wedge \phi_1$, hence it must be the case that $I \models \phi_2$. But since $I \subseteq R$ for any equivalence relation R, it follows that $R \models \phi_2$ for any equivalence relation R; that is, ϕ_2 is equivalent to **T**. Then by induction $\phi_2 = \mathbf{T}$. But ϕ is a normal form, so we have a contradiction with condition (N2). We conclude that it is impossible for a normal form $\phi = (x = y \wedge \phi_1) \vee \phi_2$ to be equivalent to **T**.

If ϕ is equivalent to **F**, then $R \models \phi$ does not hold for any equivalence relation R. This implies that both $x = y \wedge \phi_1$ and ϕ_2 are equivalent to **F**. Now, $x = y \wedge \phi_1$ is equivalent to **F** if and only $R \not\models \phi_1$ whenever R is an equivalence relation that entails $x = y$. But this implies that ϕ_1 must be equivalent to **F**, since if $R' \models \phi_1$ were to hold for some R', then $R' + \{x = y\}$ would be an equivalence relation R such that $R \models x = y \wedge \phi_1$. Since ϕ_1 is equivalent to **F**, by induction $\phi_1 = \mathbf{F}$. But ϕ is a normal form, so we have a contradiction with condition (N1). We conclude that it is impossible for a normal form $\phi = (x = y \wedge \phi_1) \vee \phi_2$ to be equivalent to **F**. $\square$

Lemma 8. *Suppose ϕ is a normal form $(x = y \wedge \phi_1) \vee \phi_2$. Then no implicant of ϕ can entail any equation $x' = y'$ such that $x' = y' \succ x = y$.*

Proof. By induction on normal form ϕ. Suppose $\phi = (x = y \wedge \phi_1) \vee \phi_2$ and we have already established the result for the subformulas ϕ_1 and ϕ_2. According to conditions (N1) and (N2), ϕ_1 is not **F** and ϕ_2 is not **T**. If ϕ has no implicants, then it is equivalent to **F**, which is impossible by Lemma 7. So, let R be an implicant of ϕ. There are now two cases:

1. R is an implicant of ϕ_2. Since we have already noted that ϕ_2 cannot be **T**, either it is **F** or it has the form $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$. It is impossible for ϕ_2 to be **F**, since this would contradict the assumption that it has R as an implicant. So, ϕ_2 must have the form $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$. Then by induction, R does not entail any equation $x' = y'$ with $x' = y' \succ x_2 = y_2$. By condition (N4), $x = y \succ x_2 = y_2$. It then follows that R does not entail any equation $x' = y'$ with $x' = y' \succ x = y$.
2. R has the form $R_1 + \{x = y\}$ for some implicant R_1 of ϕ_1. We have already noted that ϕ_1 cannot be **F**. If ϕ_1 is **T**, then R_1 is the identity relation, and R entails the single (oriented) equation $x = y$. Hence in this case R does not entail any equation $x' = y'$ with $x' = y' \succ x = y$.
   If ϕ_1 is not **T**, then it has the form $(x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$. By induction, R_1 does not entail any equation $x' = y'$ with $x' = y' \succ x_1 = y_1$. Suppose now, for the purpose of obtaining a contradiction, that R does entail some equation $x' = y'$ with $x' = y' \succ x = y$. It then follows that either $x' \succ x$ or else $x' = x$ and $y' \succ y$. If $x' \succ x$, then we would have $x' \succ x \succ y$ and $x' \succ x \succeq x'' \succ y''$ for all (x'', y'') in R_1 by condition (N4). In that case neither R_1 nor the equation $x = y$ can contain any occurrences of x', so $R = R_1 + \{x = y\}$ cannot entail any equation involving x'. This is a contradiction and we conclude that $x' \succ x$ is impossible. It must therefore be the case that x' and x are identical and $x \succ y' \succ y$. By induction, no implicant of ϕ_1 can entail any equation $x'' = y''$ with $x'' = y'' \succ x_1 = y_1$. Hence R_1 can entail neither $x = y$ nor $x' = y'$ (because $x' = y' \succ x = y$ by hypothesis and $x = y \succ x_1 = y_1$ by condition (N4)).
   If $x' = y'$ is entailed by $R = R_1 + \{x = y\}$, then there are just three possibilities:
   (a) $x' = y'$ is already entailed by R_1. We have already argued that this case is impossible.
   (b) R_1 entails $x' = y$; that is, x' is in the R_1-equivalence class of y. This is also impossible, because we have already argued that x' and x are the same variable and that R_1 cannot entail $x = y$.
   (c) R_1 entails $y' = y$; that is, y' is in the R_1-equivalence class of y. By condition (N3), y does not occur in ϕ_1, hence it is impossible for the implicant R_1 of ϕ_1 to entail any equation involving y. So this case is impossible as well.

Since all three possibilities lead to contradictions, we conclude that R does not entail any equation $x' = y'$ with $x' = y' \succ x = y$.

□

The following result is a direct consequence of (N5).

Lemma 9. *Suppose ϕ is a normal form $(x = y \wedge \phi_1) \vee \phi_2$. If $R \models_{\min} \phi$, then R satisfies precisely one of $x = y$ and ϕ_2.*

Proof. Suppose ϕ is a normal form $(x = y \wedge \phi_1) \vee \phi_2$ such that $R \models_{\min} \phi$. Suppose further, for the purpose of obtaining a contradiction, that R satisfies both $x = y$ and ϕ_2. By Lemma 5, there exists some implicant R' of ϕ_2 such that $R' \subseteq R$. Since then also $R' \models \phi$, from the assumption that $R \models_{\min} \phi$ it follows that $R' = R$. But this means that that R is an implicant of ϕ that entails $x = y$ and also satisfies ϕ_2, contradicting (N5). □

We denote the equivalence relation consisting of all and only those equations in R that do not involve variable y by $R \setminus y$. The following facts are related to notation $R \setminus y$.

Lemma 10. *If $R' \subseteq R$, then $R' \setminus y \subseteq R \setminus y$.*

Proof. Suppose equation $x' = y'$ is entailed by $R' \setminus y$. Then $x' = y'$ is entailed by R' and does not involve y. Since $R' \subseteq R$, equation $x' = y'$ is also entailed by R, hence is in $R \setminus y$. □

Lemma 11. *If R entails $x = y$ then $R \setminus y + \{x = y\} = R$.*

Proof. We show that for any equation $x' = y'$, $R \setminus y + \{x = y\}$ entails $x' = y'$ if and only if R entails $x' = y'$.

First, assume $R \setminus y + \{x = y\}$ entails $x' = y'$. Since R entails $x = y$ and $R \setminus y \subseteq R$, we have that $R \setminus y + \{x = y\} \subseteq R$. It then follows that R entails $x' = y'$.

Now, assume R entails $x' = y'$. There are two cases:

1. $x' = y'$ is entailed by $R \setminus y$. It is then obvious that $R \setminus y + \{x = y\}$ also entails $x' = y'$.
2. $x' = y'$ is not entailed by $R \setminus y$. It then follows that $x' = y'$ involves variable y. This leaves two possibilities:
   (a) $x' = y'$ is $x' = y$. Since R entails $x' = y$ and $x = y$, it follows that R entails $x' = x$ (or $x = x'$). Since $x' = x$ (or $x = x'$) does not involve y, $R \setminus y$ also entails $x' = x$ (or $x = x'$). Consequently, $R \setminus y + \{x = y\}$ entails $x' = y$, which by assumption is the same as $x' = y'$.
   (b) $x' = y'$ is $y = y'$. Since R entails $y = y'$ and $x = y$, it follows that R entails $x = y'$. Since $x = y'$ does not involve y, $R \setminus y$ also entails $x = y'$. Consequently, $R \setminus y + \{x = y\}$ entails $y = y'$, which by assumption is the same as $x' = y'$.

□

Lemma 12. *If R does not entail any equation involving y, then $(R + \{x = y\}) \setminus y = R$.*

Proof. If R does not entail any equation involving y, then the R-equivalence class of y is the singleton set $\{y\}$. The equivalence classes of $R + \{x = y\}$ are therefore identical to those of R, except for the equivalence class of x, which now contains one additional element y. Thus, every equation entailed by $R + \{x = y\}$ is either already an equation entailed by R or it is an equation that relates y to an element of the R-equivalence class of x. That is, $(R + \{x = y\}) \setminus y = R$. $\square$

Lemma 13. *Suppose ϕ is a positive equational formula of the form $x = y \wedge \phi_1$, where ϕ_1 contains no occurrences of y. Then $R \models \phi$ if and only if R entails $x = y$ and $R \setminus y \models \phi_1$.*

Proof. First, suppose $R \models \phi$. Then R entails $x = y$. In addition, since any equation that does not involve y and is entailed by R is also entailed by $R \setminus y$, we have that $R \setminus y \models \phi_1$.

Conversely, suppose R entails $x = y$ and $R \setminus y \models \phi_1$. Clearly, $R \setminus y \subseteq R$. Since by hypothesis, $R \setminus y \models \phi_1$, it follows that $R \models \phi_1$. Since R entails $x = y$ and $R \models \phi_1$, we have $R \models \phi$. $\square$

In contrast to decision forms in general, normal forms satisfy the converse of Lemma 6.

Lemma 14. *Suppose ϕ is a normal form. If R is an implicant of ϕ, then $R \models_{\min} \phi$.*

Proof. By structural induction. It is easy to verify that the lemma is true in case ϕ is $\mathbf{T}$ or $\mathbf{F}$. Now, suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$, and the lemma is true for subformulas ϕ_1 and ϕ_2. There are two cases:

1. R is an implicant of ϕ_2. By induction, $R \models_{\min} \phi_2$. Since ϕ is a normal form, by condition (N5) R does not entail $x = y$. It then follows that if R' is any equivalence relation such that $R' \subseteq R$ and $R' \models \phi$ then $R' \models \phi_2$. Since $R \models_{\min} \phi_2$, $R' = R$. Hence, $R \models_{\min} \phi$.
2. R is $R_1 + \{x = y\}$, where R_1 is an implicant of ϕ_1. By induction, $R_1 \models_{\min} \phi_1$. Suppose R' is any equivalence relation such that $R' \subseteq R$ and $R' \models \phi$. We first claim that R' does not satisfy ϕ_2. Suppose, to the contrary, that $R' \models \phi_2$. Then $R' + \{x = y\} \models \phi_2$, and since $R' + \{x = y\} \subseteq R$, it follows that $R \models \phi_2$. But this contradicts (N5), which states that no implicant of ϕ that entails $x = y$ can satisfy ϕ_2. Thus R' does not satisfy ϕ_2, so instead $R' \models x = y \wedge \phi_1$. Now, since $R' \models x = y \wedge \phi_1$ and ϕ_1 contains no occurrences of y, we have $R' \setminus y \models \phi_1$ by Lemma 13. Since R_1 is an implicant of ϕ_1, which contains no occurrences of y, R_1 entails no equations involving y, hence $R \setminus y = (R_1 + \{x = y\}) \setminus y = R_1$ by Lemma 12. Since $R \setminus y = R_1$ and $R' \subseteq R$, it follows that $R' \setminus y \subseteq R_1$ by Lemma 10. Since $R_1 \models_{\min} \phi_1$, we have $R' \setminus y = R_1$, hence also $(R' \setminus y) + \{x = y\} = R_1 + \{x = y\}$. By Lemma 11, $R' = (R' \setminus y) + \{x = y\}$ and $(R \setminus y) + \{x = y\} = R$. It follows that $R' = R$. We have shown that if $R' \subseteq R$ is such that $R' \models \phi$, then $R' = R$. Thus, $R \models_{\min} \phi$.

□

Corollary 1. *Suppose ϕ and ϕ' are equivalent normal forms. Then every implicant of ϕ is an implicant of ϕ' and vice versa.*

Proof. Suppose R is an implicant of ϕ. By Lemma 14, we have $R \models_{\min} \phi$. Since ϕ and ϕ' are equivalent, it follows that $R \models_{\min} \phi'$. By Lemma 6 it follows that R is also an implicant of ϕ'. Thus, every implicant of ϕ is also an implicant of ϕ'. Symmetric reasoning shows that every implicant of ϕ' is also an implicant of ϕ. □

A structural induction using the properties we have established now shows that normal forms are canonical.

Theorem 1. *Normal forms ϕ and ϕ' are equivalent if and only if they are identical.*

Proof. If ϕ and ϕ' are identical then it is obvious that they are equivalent. Conversely, we show by induction that if ϕ and ϕ' are equivalent then they are identical. First, consider the case in which at least one of ϕ or ϕ' is either $\mathbf{F}$ or $\mathbf{T}$. In this case, if ϕ and ϕ' are equivalent, then they are identical by Lemma 7.

Now, suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$, ϕ' is $(x' = y' \wedge \phi'_1) \vee \phi'_2$, and ϕ is equivalent to ϕ'. Suppose further, as the induction hypothesis, that if ϕ_1 is equivalent to ϕ'_1, then ϕ_1 and ϕ'_1 are identical, and if ϕ_2 is equivalent to ϕ'_2, then ϕ_2 and ϕ'_2 are identical. We will show that ϕ and ϕ' are identical.

We first show that $(x, y) = (x', y')$. By Lemma 7, every normal form that is not identical to $\mathbf{F}$ is not equivalent to $\mathbf{F}$, which means that it is satisfiable and has a nonempty set of implicants. In particular this is true for ϕ_1, so that there exists some implicant R_1 of ϕ_1. Then $R = R_1 + \{x = y\}$ is an implicant of ϕ that entails $x = y$. By Corollary 1, R is also an implicant of ϕ'. But by Lemma 8, no implicant of ϕ' can entail any equation $x = y$ such that $(x, y) \succ (x', y')$. Symmetric reasoning shows that $(x', y') \succ (x, y)$ is likewise impossible, hence $(x, y) = (x', y')$ as asserted.

We next show that if an equivalence relation R minimally satisfies ϕ_2 then it minimally satisfies ϕ'_2 and vice versa. Assume $R \models_{\min} \phi_2$. Then R is an implicant of ϕ_2 by Lemma 6, hence R is also an implicant of ϕ. Since ϕ is a normal form, by condition (N4) R does not entail $x = y$. Since ϕ and ϕ' have the same set of implicants by Corollary 1, it follows that R is also an implicant of ϕ'. Since R does not entail $x = y$, it also does not entail $x' = y'$, which is the same equation as $x = y$. Thus, R cannot satisfy $x' = y' \wedge \phi'_1$, hence it must be an implicant of ϕ'_2. Then $R \models_{\min} \phi'_2$ by Lemma 14. Symmetric reasoning show that if $R \models_{\min} \phi_2$ then $R \models_{\min} \phi'_2$.

Since ϕ_2 and ϕ'_2 are minimally satisfied by the same set of equivalence relations, they are equivalent. By induction hypothesis, ϕ_2 and ϕ'_2 are identical.

We next show that if an equivalence relation R minimally satisfies ϕ_1 then R satisfies ϕ'_1 and vice versa. Assume $R \models_{\min} \phi_1$. Then R is an implicant of ϕ_1 by Lemma 6, hence $R + \{x = y\}$ is an implicant of ϕ. Moreover, according to condition (N3), R does not entail any equation that involves y since y does not appear in ϕ_1. Now, by Corollary 1, $R + \{x = y\}$ is also an implicant of ϕ'. Also, $R + \{x = y\} = R + \{x' = y'\}$ since $x = y$ and $x' = y'$ are the same equation. By condition (N5), we cannot have $R + \{x' = y'\} \models \phi'_2$, hence it must be the case that $R + \{x' = y'\} \models x' = y' \wedge \phi'_1$. Since y' does not appear in ϕ'_1, we have $(R + \{x' = y'\}) \setminus y' \models \phi'_1$ by Lemma 13. Since R does not entail any equation involving y', $(R + \{x' = y'\}) \setminus y' = R$ by Lemma 12. Hence, $R \models \phi'_1$. Symmetric reasoning shows that if $R \models_{\min} \phi'_1$ then $R \models \phi_1$.

Since ϕ_1 and ϕ'_1 are minimally satisfied by the same set of equivalence relations, they are equivalent. By induction hypothesis, ϕ_1 and ϕ'_1 are identical.

We have shown that $x = y$ and $x' = y'$ are identical, that ϕ_1 and ϕ'_1 are identical, and that ϕ_2 and ϕ'_2 are identical. It then follows that ϕ and ϕ' are identical, completing the proof. □

The fact that normal forms are canonical representatives of their logical equivalence class gives us a way to represent equivalence relations as normal forms. This will be useful in the next section. Formally, given an equivalence relation R, it is clear that the conjunction of all the equations entailed by R is a positive equational formula ϕ with the property that $R' \models \phi$ if and only if $R \subseteq R'$. There is therefore a unique normal form with the same property. Thus, we define the *characteristic formula* of an equivalence relation R to be the unique normal form χ_R such that $R' \models \chi_R$ if and only if $R \subseteq R'$.

We close this section by considering a variant of algorithm SAT that exploits the ordering properties of a normal form. We say that $x = y$ is the *top-level equation* of normal form $(x = y \wedge \phi_1) \vee \phi_2$. By condition (N4), if $x_1 = y_1$ is the top-level equation of ϕ_1 then $x = y \succ x_1 = y_1$ and if $x_2 = y_2$ is the top-level equation of ϕ_2, then $x = y \succ x_2 = y_2$. This property of normal forms suggests a variant of algorithm SAT, called SAT$'$, which is shown in Figure 3. Function SAT$'$ takes as arguments an equivalence relation R and a set Ψ of formulas which is sorted by top-level equation, and it returns **true** if $R \models \bigvee \Psi$ or **false** if $R \not\models \bigvee \Psi$. At each iteration, the set Ψ is examined. If Ψ is empty then **false** is returned, since $\bigvee \emptyset$ is equivalent to $\mathbf{F}$. If Ψ contains $\mathbf{T}$ then *true* is returned since $\bigvee\{\mathbf{T}\} = \mathbf{T}$. If $\Psi \neq \emptyset$ and Ψ does not contain $\mathbf{T}$ then only nontrivial formulas are contained in Ψ and they are sorted by their top-level equations. Let $x = y$ be the top-level equation of the first formula in Ψ. For each formula $\psi = (x = y \wedge \psi_1) \vee \psi_2$ in Ψ, we remove ψ from Ψ, add ψ_2 to Ψ, and if $(x, y) \in R$ then also add ψ_1 to Ψ. Then we go on to the next iteration. It is obvious that each "query" $(x, y) \in R$ need be made at most once in SAT$'$. Moreover, if the formulas in Ψ are stored in a structure-sharing form in which multiple occurrences of the same subformula share a single representation in storage, then it is possible to avoid considering each multiply occurring subformula more than once during the course of the algorithm.

```
fun Sat′($R, \Psi$) =
  if $\Psi = \emptyset$ then false
  else if $\mathbf{T} \in \Psi$ then true
  else
    // $\Psi$ is kept sorted by top-level equation
    let $(x = y \wedge \phi_1) \vee \phi_2$  be the first formula in $\Psi$
        $b$ = if $(x, y) \in R$ then true else false

        fun Process($\Psi'$) =
          if $\Psi'$ contains some formula $\psi$ with top-level equation $x = y$ then
            let $(x = y \wedge \psi_1) \vee \psi_2$ be $\psi$
                $\Psi'' = (\Psi' \setminus \psi)$
                        $\cup$ (if $b$ then $\{\psi_1\}$ else $\emptyset$)
                        $\cup$ (if $\psi_2 \neq \mathbf{F}$ then $\{\psi_2\}$ else $\emptyset$)
            in
              Process($\Psi''$)
            end
          else
            $\Psi'$
    in
      Sat′($R$, Process($\Psi$))
    end
```

Fig. 3. Function Sat′

5 Recursive Conditions for Normality

The defining properties (N1)–(N5) of normal forms do not directly suggest an algorithm for converting a decision form into an equivalent normal form. In this section we obtain recursively defined conditions sufficient to guarantee that a decision form is a normal form. In the next section we use these conditions as the basis for constructing an algorithm for reducing a decision form to a normal form.

We begin by defining the notion of a "decision context", which is essentially a "decision form with a hole." Formally, the set of *decision contexts* (or just *contexts*) is defined inductively as follows:

– $[\,]$ is a context, called the *empty context*.
– If Γ is a context, $x = y$ is an equation, and ψ is a decision form, then $\Gamma[(x = y \wedge [\,]) \vee \psi]$ is a context.

We define the notion "substitution of formula ϕ in context Γ," denoted $\Gamma\{\phi\}$, recursively as follows:

– If Γ is the empty context $[\,]$, then $\Gamma\{\phi\} = \phi$.
– If Γ is $\Delta[(x = y \wedge [\,]) \vee \psi]$, then $\Gamma\{\phi\}$ is $\Delta\{(x = y \wedge \phi) \vee \psi\}$.

To each context Γ we associate an equivalence relation R_Γ, defined recursively as follows:

– $R_{[\,]}$ is the identity relation.
– If $\mathit{\Gamma}$ is $\mathit{\Delta}[(x = y \wedge [\,]) \vee \psi]$, then $R_{\mathit{\Gamma}} = R_{\mathit{\Delta}} + \{x = y\}$.

That is, $R_{\mathit{\Gamma}}$ is just the equivalence relation generated by the set of equations appearing along the "spine" of context $\mathit{\Gamma}$.

If $\mathit{\Gamma}$ is a context and ϕ and ϕ' are formulas, then we say that ϕ *implies* ϕ' *in context* $\mathit{\Gamma}$ if $\mathit{\Gamma}\{\phi\}$ implies $\mathit{\Gamma}\{\phi'\}$. If ϕ implies ϕ' in context $\mathit{\Gamma}$ and also ϕ' implies ϕ in context $\mathit{\Gamma}$, then we say that ϕ and ϕ' are *equivalent in context* $\mathit{\Gamma}$. We say that ϕ is *subsumed by* $\mathit{\Gamma}$ if ϕ is equivalent to $\mathbf{F}$ in context $\mathit{\Gamma}$. If $\mathbf{T}$ is subsumed by $\mathit{\Gamma}$, then the context $\mathit{\Gamma}$ is called *degenerate*, otherwise it is called *nondegenerate*. It will be convenient for us to extend the concept of subsumption to equivalence relations by saying that an equivalence relation R is subsumed by $\mathit{\Gamma}$ precisely when the characteristic formula χ_R is subsumed by $\mathit{\Gamma}$.

Lemma 15. *If ϕ implies ϕ', then $\mathit{\Gamma}\{\phi\}$ implies $\mathit{\Gamma}\{\phi'\}$.*

Proof. By induction on contexts. If $\mathit{\Gamma}$ is $[\ ]$, the result is immediate. Suppose $\mathit{\Gamma}$ is $\mathit{\Gamma}'[(x = y \wedge [\,]) \vee \psi]$. Then $\mathit{\Gamma}\{\phi\}$ is $\mathit{\Gamma}'\{(x = y \wedge \phi) \vee \psi\}$. Since $(x = y \wedge \phi) \vee \psi$ implies $(x = y \wedge \phi') \vee \psi$, by induction $\mathit{\Gamma}'\{(x = y \wedge \phi) \vee \psi\}$ implies $\mathit{\Gamma}'\{(x = y \wedge \phi') \vee \psi\}$. But $\mathit{\Gamma}'\{(x = y \wedge \phi') \vee \psi\}$ is just $\mathit{\Gamma}\{\phi'\}$. □

Lemma 16. *For all contexts $\mathit{\Gamma}$ and formulas ϕ_1 and ϕ_2, $\mathit{\Gamma}\{\phi_1 \vee \phi_2\}$ is equivalent to $\mathit{\Gamma}\{\phi_1\} \vee \mathit{\Gamma}\{\phi_2\}$, and $\mathit{\Gamma}\{\phi_1 \wedge \phi_2\}$ is equivalent to $\mathit{\Gamma}\{\phi_1\} \wedge \mathit{\Gamma}\{\phi_2\}$.*

Proof. By induction on contexts. If $\mathit{\Gamma}$ is $[\,]$, the result is immediate. Suppose $\mathit{\Gamma}$ is $\mathit{\Gamma}'[(x = y \wedge [\,]) \vee \psi]$. Then $\mathit{\Gamma}\{\phi_1 \vee \phi_2\}$ is $\mathit{\Gamma}'\{(x = y \wedge (\phi_1 \vee \phi_2)) \vee \psi\}$. Since $(x = y \wedge (\phi_1 \vee \phi_2)) \vee \psi$ is equivalent to $((x = y \wedge \phi_1) \vee \psi) \vee ((x = y \wedge \phi_2) \vee \psi)$, by induction, $\mathit{\Gamma}'\{(x = y \wedge (\phi_1 \vee \phi_2)) \vee \psi\}$ is equivalent to

$$\mathit{\Gamma}'\{(x = y \wedge \phi_1) \vee \psi\} \vee \mathit{\Gamma}'\{(x = y \wedge \phi_2) \vee \psi\};$$

that is, to $\mathit{\Gamma}\{\phi_1\} \vee \mathit{\Gamma}\{\phi_2\}$. Similarly, $\mathit{\Gamma}\{\phi_1 \wedge \phi_2\}$ is $\mathit{\Gamma}'\{(x = y \wedge (\phi_1 \wedge \phi_2)) \vee \psi\}$. Since $(x = y \wedge (\phi_1 \wedge \phi_2)) \vee \psi$ is equivalent to $((x = y \wedge \phi_1) \vee \psi) \wedge ((x = y \wedge \phi_2) \vee \psi)$, by induction, $\mathit{\Gamma}'\{(x = y \wedge (\phi_1 \wedge \phi_2)) \vee \psi\}$ is equivalent to

$$\mathit{\Gamma}'\{(x = y \wedge \phi_1) \vee \psi\} \wedge \mathit{\Gamma}'\{(x = y \wedge \phi_2) \vee \psi\};$$

that is, to $\mathit{\Gamma}\{\phi_1\} \wedge \mathit{\Gamma}\{\phi_2\}$. □

Corollary 2. *Suppose ϕ_1 implies ϕ_1' in context $\mathit{\Gamma}$ and ϕ_2 implies ϕ_2' in context $\mathit{\Gamma}$. Then $\phi_1 \vee \phi_2$ implies $\phi_1' \vee \phi_2'$ in context $\mathit{\Gamma}$, and $\phi_1 \wedge \phi_2$ implies $\phi_1' \wedge \phi_2'$ in context $\mathit{\Gamma}$.*

Proof. By Lemma 16, $\mathit{\Gamma}\{\phi_1 \vee \phi_2\}$ is equivalent to $\mathit{\Gamma}\{\phi_1\} \vee \mathit{\Gamma}\{\phi_2\}$. Since ϕ_1 implies ϕ_1' in context $\mathit{\Gamma}$ and ϕ_2 implies ϕ_2' in context $\mathit{\Gamma}$, $\mathit{\Gamma}\{\phi_1\}$ implies $\mathit{\Gamma}\{\phi_1'\}$ and $\mathit{\Gamma}\{\phi_2\}$ implies $\mathit{\Gamma}\{\phi_2'\}$, hence $\mathit{\Gamma}\{\phi_1\} \vee \mathit{\Gamma}\{\phi_2\}$ implies $\mathit{\Gamma}\{\phi_1'\} \vee \mathit{\Gamma}\{\phi_2'\}$. But $\mathit{\Gamma}\{\phi_1'\} \vee \mathit{\Gamma}\{\phi_2'\}$ is equivalent to $\mathit{\Gamma}\{\phi_1' \vee \phi_2'\}$ by Lemma 16. Similarly, By Lemma 16, $\mathit{\Gamma}\{\phi_1 \wedge \phi_2\}$ is equivalent to $\mathit{\Gamma}\{\phi_1\} \wedge \mathit{\Gamma}\{\phi_2\}$. Moreover, $\mathit{\Gamma}\{\phi_1\} \wedge \mathit{\Gamma}\{\phi_2\}$ implies $\mathit{\Gamma}\{\phi_1'\} \wedge \mathit{\Gamma}\{\phi_2'\}$. But $\mathit{\Gamma}\{\phi_1'\} \wedge \mathit{\Gamma}\{\phi_2'\}$ is equivalent to $\mathit{\Gamma}\{\phi_1' \wedge \phi_2'\}$ by Lemma 16. □

Lemma 17. *For all contexts Γ, $\Gamma\{\phi\}$ is equivalent to $\Gamma\{\mathbf{F}\} \vee (\chi_{R_\Gamma} \wedge \phi)$.*

Proof. By induction on contexts. Suppose Γ is $[\,]$. Then $\Gamma\{\phi\} = \phi$, $\Gamma\{\mathbf{F}\} = \mathbf{F}$, and $\chi_{R_\Gamma} = \mathbf{T}$. Clearly, ϕ is equivalent to $\mathbf{F} \vee (\mathbf{T} \wedge \phi)$.

Suppose Γ is $\Gamma'[(x = y \wedge [\,]) \vee \psi]$. Then $\Gamma\{\phi\} = \Gamma'\{(x = y \wedge \phi) \vee \psi\}$. By induction, $\Gamma'\{(x = y \wedge \phi) \vee \psi\}$ is equivalent to $\Gamma'\{\mathbf{F}\} \vee (\chi_{R_{\Gamma'}} \wedge ((x = y \wedge \phi) \vee \psi))$. But $\chi_{R_{\Gamma'}} \wedge ((x = y \wedge \phi) \vee \psi)$ is equivalent to $(\chi_{R_{\Gamma'}} \wedge \psi) \vee (\chi_{R_{\Gamma'}} \wedge x = y \wedge \phi)$. By induction $\Gamma'\{\mathbf{F}\} \vee (\chi_{R_{\Gamma'}} \wedge \psi)$ is equivalent to $\Gamma'\{\psi\}$, which is in turn equivalent to $\Gamma\{\mathbf{F}\}$. Also, $(\chi_{R_{\Gamma'}} \wedge x = y \wedge \phi)$ is equivalent to $(\chi_{R_\Gamma} \wedge \phi)$. Thus, $\Gamma\{\phi\}$ is equivalent to $\Gamma\{\mathbf{F}\} \vee (\chi_{R_\Gamma} \wedge \phi)$, as asserted. □

Lemma 18. *Suppose ϕ implies ϕ' in context $\Gamma[(x = y \wedge [\,]) \vee \psi]$. Then $(x = y \wedge \phi) \vee \psi$ implies $(x = y \wedge \phi') \vee \psi$ in context Γ.*

Proof. By definition, if ϕ implies ϕ' in context $\Gamma[(x = y \wedge [\,]) \vee \psi]$, then $\Gamma\{(x = y \wedge \phi) \vee \psi\}$ implies $\Gamma\{(x = y \wedge \phi') \vee \psi\}$. But this exactly says that $(x = y \wedge \phi) \vee \psi$ implies $(x = y \wedge \phi') \vee \psi$ in context Γ. □

Lemma 19. *Suppose ψ implies ψ' in context Γ. Then $(x = y \wedge \phi) \vee \psi$ implies $(x = y \wedge \phi) \vee \psi'$ in context Γ.*

Proof. Since ψ is equivalent to $\mathbf{F} \vee \psi$, by Lemma 16, $\Gamma\{(x = y \wedge \phi) \vee \psi\}$ is equivalent to

$$\Gamma\{(x = y \wedge \phi) \vee \mathbf{F}\} \vee \Gamma\{\psi\}.$$

Since ψ implies ψ' in context Γ, the displayed formula implies

$$\Gamma\{(x = y \wedge \phi) \vee \mathbf{F}\} \vee \Gamma\{\psi'\},$$

which is equivalent to $\Gamma\{(x = y \wedge \phi) \vee \psi'\}$. □

Lemma 20. *If R is an equivalence relation such that $R_\Gamma \subseteq R$ and $R \models \phi$ then $R \models \Gamma\{\phi\}$.*

Proof. By induction on contexts. Suppose Γ is $[\,]$. Then $\Gamma\{\phi\} = \phi$. By hypothesis, $R \models \phi$. Thus, $R \models \Gamma\{\phi\}$.

Suppose Γ is $\Gamma'[(x = y \wedge [\,]) \vee \psi]$. Then $\Gamma\{\phi\} = \Gamma'\{(x = y \wedge \phi) \vee \psi\}$. Since $R_\Gamma \subseteq R$ and $R_\Gamma = R_{\Gamma'} + \{x = y\}$, it follows that $R \models x = y$. By hypothesis, $R \models \phi$, it then follows that $R \models (x = y \wedge \phi) \vee \psi$. By induction, $R \models \Gamma'\{(x = y \wedge \phi) \vee \psi\}$. Thus, $R \models \Gamma\{\phi\}$, as asserted. □

Lemma 21. *Equivalence relation R is subsumed by Γ if and only if $R_\Gamma + R$ satisfies $\Gamma\{\mathbf{F}\}$.*

Proof. Suppose R is subsumed by Γ. Then $\Gamma\{\chi_R\}$ is equivalent to $\Gamma\{\mathbf{F}\}$. Since $R_\Gamma + R$ satisfies $\Gamma\{\chi_R\}$, it follows that $R_\Gamma + R$ satisfies $\Gamma\{\mathbf{F}\}$.

Conversely, suppose $R_\Gamma + R$ satisfies $\Gamma\{\mathbf{F}\}$. By Lemma 17, $\Gamma\{\chi_R\}$ is equivalent to $\Gamma\{\mathbf{F}\} \vee (\chi_{R_\Gamma} \wedge \chi_R)$, which in turn is equivalent to $\Gamma\{\mathbf{F}\} \vee (\chi_{R_\Gamma + R})$. So, if R' satisfies $\Gamma\{\chi_R\}$, then either R' satisfies $\Gamma\{\mathbf{F}\}$ or R' satisfies $\chi_{R_\Gamma + R}$. But if R' satisfies $\chi_{R_\Gamma + R}$, then it satisfies $\Gamma\{\mathbf{F}\}$ by hypothesis. Hence if R' satisfies $\Gamma\{\chi_R\}$ then it also satisfies $\Gamma\{\mathbf{F}\}$, thus showing that $\Gamma\{\chi_R\}$ implies $\Gamma\{\mathbf{F}\}$; that is, R is subsumed by Γ.

```
fun DEGENERATE($\Gamma$) =
  let fun SATANY($R, \Gamma$) =
        if $\Gamma = [\,]$ then false
        else
          let $\Gamma'[(x = y \wedge [\,]) \vee \psi]$ be $\Gamma$
          in
            SAT$'(R, \{\psi\})$ orelse SATANY$(R, \Gamma')$
          end
  in
    SATANY$(R_\Gamma, \Gamma)$
  end
```

Fig. 4. Function DEGENERATE

Corollary 3. *Context Γ is degenerate if and only if R_Γ satisfies $\Gamma\{\mathbf{F}\}$.*

Proof. Context Γ is degenerate if and only if $\Gamma\{\mathbf{T}\}$ is equivalent to $\Gamma\{\mathbf{F}\}$. Since $\mathbf{T}$ is the characteristic formula of the identity relation I, $\Gamma\{\mathbf{T}\}$ is equivalent to $\Gamma\{\mathbf{F}\}$ if and only if I is subsumed by Γ. By Lemma 21, I is subsumed by Γ if and only if $R_\Gamma + I$ satisfies $\Gamma\{\mathbf{F}\}$. But $R_\Gamma + I = R_\Gamma$, so I is subsumed by Γ if and only if R_Γ satisfies $\Gamma\{\mathbf{F}\}$. Thus, Γ is degenerate if and only if R_Γ satisfies $\Gamma\{\mathbf{F}\}$. $\square$

Lemma 22. *Suppose $\Gamma = \Gamma'[(x = y \wedge [\,]) \vee \psi]$ and R is an equivalence relation such that $R_{\Gamma'} \subseteq R$. Then $R \models \Gamma\{\mathbf{F}\}$ if and only if either $R \models \psi$ or else $R \models \Gamma'\{\mathbf{F}\}$.*

Proof. Note that $\Gamma\{\mathbf{F}\}$ is equivalent to $\Gamma'\{\psi\}$, and by Lemma 17, $\Gamma'\{\psi\}$ is equivalent to $\Gamma'\{\mathbf{F}\} \vee (\chi_{R_{\Gamma'}} \wedge \psi)$. Thus $R \models \Gamma\{\mathbf{F}\}$ if and only if either $R \models \Gamma'\{\mathbf{F}\}$ or $R \models \chi_{R_{\Gamma'}} \wedge \psi$. Since $R_{\Gamma'} \subseteq R$ by hypothesis, $R \models \chi_{R_{\Gamma'}} \wedge \psi$ holds if and only if $R \models \psi$. $\square$

Based on Corollary 3 and Lemma 22, algorithm DEGENERATE for testing whether a context is degenerate, is presented in Figure 4.

Lemma 23 (Correctness of DEGENERATE). *If Γ is a context, then a call* DEGENERATE(Γ) *terminates and returns a value b which is* **true** *if and only if context Γ is degenerate.*

Proof. Immediate from Corollary 3 and Lemma 22. $\square$

A decision form ϕ is called *reduced in context Γ* if either $\phi = \mathbf{F}$, or else Γ is nondegenerate and the following three conditions are satisfied if ϕ has the form $(x = y \wedge \phi_1) \vee \phi_2$:

R1: ϕ_1 is reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$ and is not $\mathbf{F}$.
R2: ϕ_2 is reduced in context Γ and is not $\mathbf{T}$;

R3: x and y are the greatest elements (with respect to the ordering $\succ$) of their respective equivalence classes of R_Γ.

A decision form ϕ is called *ordered below* $x = y$ (resp. *ordered strictly below* $x = y$) if ϕ is either $\mathbf{T}$ or $\mathbf{F}$ or else it has the form $(x' = y' \wedge \phi_1) \vee \phi_2$ and the following three conditions are satisfied:

O1: ϕ_1 is ordered strictly below $x' = y'$.
O2: ϕ_2 is ordered strictly below $x' = y'$.
O3: $x = y \succeq x' = y'$ (resp. $x = y \succ x' = y'$).

A decision form ϕ is called *ordered* if it is either $\mathbf{T}$ or $\mathbf{F}$ or else it has the form $(x' = y' \wedge \phi_1) \vee \phi_2$ and is ordered below its top-level equation $x' = y'$. Clearly, if ϕ is ordered strictly below $x = y$, then it is ordered below $x = y$, and if it is ordered below $x = y$, then it is ordered.

Theorem 2. *Suppose decision form ϕ is ordered and reduced in some context Γ. Then ϕ is a normal form, which has the following additional properties:*

1. *No implicant of ϕ is subsumed by Γ.*
2. *No variable on the right-hand side of any equation in R_Γ occurs in ϕ.*

Proof. By structural induction on decision forms.

We first consider the basis cases in which ϕ is either $\mathbf{F}$ or $\mathbf{T}$. In either case, it is obvious that ϕ is a normal form. Since no variables occur in ϕ, property (2) holds vacuously. If $\phi = \mathbf{F}$, then ϕ has no implicants, so property (1) holds vacuously in that case. If $\phi = \mathbf{T}$ and R is an implicant of ϕ, then R is the identity relation I. Since ϕ is $\mathbf{T}$ and is assumed reduced in context Γ, the context Γ must be nondegenerate, hence I cannot be subsumed by Γ. Thus, property (1) holds in this case as well.

Suppose now that $\phi = (x = y \wedge \phi_1) \vee \phi_2$. We assume as the induction hypothesis that the result has already been established for ϕ_1 and ϕ_2 and we show that conditions (N1)–(N5) are satisfied by ϕ:

(N1) By conditions (R1) and (O1), ϕ_1 is ordered and reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, and it is not $\mathbf{F}$. It follows by induction that ϕ_1 is a normal form and is not $\mathbf{F}$. Hence, condition (N1) is satisfied by ϕ.

(N2) By conditions (R2) and (O2), ϕ_2 is ordered and reduced in context Γ, and it is not $\mathbf{T}$. It follows by induction that ϕ_2 is a normal form and is not $\mathbf{T}$. Hence, condition (N2) is satisfied by ϕ.

(N3) By induction, ϕ_1 satisfies property (2) with R_Γ replaced by $R_{\Gamma'}$, where $\Gamma' = \Gamma[(x = y \wedge [\,]) \vee \phi_2]$. Since $R_{\Gamma'} = R_\Gamma + \{x = y\}$, this means that no variable on the right-hand side of any equation in $R_\Gamma + \{x = y\}$ occurs in ϕ_1. In particular, y does not occur in ϕ_1, thus establishing that (N3) is satisfied by ϕ.

(N4) Suppose that R is an implicant of ϕ_1 and that $x' = y'$ is an equation entailed by R. Then ϕ_1 is not $\mathbf{F}$ because $\mathbf{F}$ has no implicants, and it is not $\mathbf{T}$ because the only implicant of $\mathbf{T}$ is the identity relation, which entails no equations. So, ϕ_1 has a top-level equation, say $x_1 = y_1$. By Lemma 8 and the fact that (by induction) ϕ_1 is a normal form, $x_1 = y_1 \succeq x' = y'$. Since by condition (O1) ϕ_1 is ordered strictly below $x = y$, we have that $x = y \succ x_1 = y_1$, from which it follows that $x = y \succ x' = y'$, establishing (N4) in this case. Essentially the same reasoning applies to an implicant R of ϕ_2.

(N5) Suppose R is an implicant of ϕ that entails $x = y$. We have shown that condition (N4) is satisfied by ϕ, hence no implicant of ϕ_2 can entail $x = y$. From this, it follows that R must have the form $R_1 + \{x = y\}$, where R_1 is an implicant of ϕ_1. By induction, ϕ_1 satisfies property (1) with Γ replaced by $\Gamma' = \Gamma[(x = y \wedge [\,]) \vee \phi_2]$; that is to say, no implicant of ϕ_1 is subsumed by Γ'. In particular, the implicant R_1 of ϕ_1 is not subsumed by Γ', from which it follows by Lemma 21 that $R_{\Gamma'} + R_1$ does not satisfy $\Gamma'\{\mathbf{F}\}$. But $\Gamma'\{\mathbf{F}\} = \Gamma\{(x = y \wedge \mathbf{F}) \vee \phi_2\}$, which is equivalent to $\Gamma\{\phi_2\}$. Thus, $R_{\Gamma'} + R_1$ does not satisfy $\Gamma\{\phi_2\}$. Since $R_{\Gamma'} + R_1$ contains R_Γ, by Lemma 20, if it satisfies ϕ_2 then it also satisfies $\Gamma\{\phi_2\}$. Hence $R_{\Gamma'} + R_1$ does not satisfy ϕ_2. Since $R_{\Gamma'} + R_1 = (R_\Gamma + \{x = y\}) + R_1 = R_\Gamma + (R_1 + \{x = y\}) = R_\Gamma + R$, it follows that $R_\Gamma + R$ does not satisfy ϕ_2. Consequently, R also does not satisfy ϕ_2, thus establishing that condition (N5) is satisfied by ϕ.

Finally, we show that properties (1)–(2) are satisfied by ϕ.

1. Suppose R is an implicant of ϕ. Then either R is an implicant of ϕ_2 or else $R = R_1 + \{x = y\}$, where R_1 is an implicant of ϕ_1. Suppose R is an implicant of ϕ_2. By induction, ϕ_2 satisfies property (1), hence R is not subsumed by Γ. Now suppose $R = R_1 + \{x = y\}$, where R_1 is an implicant of ϕ_1. By induction, ϕ_1 satisfies property (1) with Γ replaced by $\Gamma' = \Gamma[(x = y \wedge [\,]) \vee \phi_2]$. Thus, implicant R_1 of ϕ_1 is not subsumed by Γ'. By Lemma 21, $R_{\Gamma'} + R_1$ does not satisfy $\Gamma'\{\mathbf{F}\}$. But $R_{\Gamma'} + R_1 = R_\Gamma + R$, so $R_\Gamma + R$ does not satisfy $\Gamma'\{\mathbf{F}\}$. Also, $\Gamma'\{\mathbf{F}\}$ is equivalent to $\Gamma\{\phi_2\}$, so $R_\Gamma + R$ does not satisfy $\Gamma\{\phi_2\}$. By Lemma 17, $\Gamma\{\phi_2\}$ is equivalent to $\Gamma\{\mathbf{F}\} \vee (\chi_{R_\Gamma} \wedge \phi_2)$. Thus $R_\Gamma + R$ does not satisfy $\Gamma\{\mathbf{F}\}$; that is, R is not subsumed by Γ.
2. By induction, no variable on the right-hand side of any equation in R occurs in ϕ_2, and no variable on the right-hand side of any equation in $R + \{x = y\}$ occurs in ϕ_1. Moreover, by condition (R3), x and y are the greatest elements in their respective R-equivalence classes. Hence, neither x nor y can appear on the right-hand side of any equation in R. Since every variable occurring in ϕ occurs either in ϕ_1, in ϕ_2, or is one of x or y, we conclude that no variable occurring in ϕ can be on the right-hand side of any equation in R. Hence property (2) is satisfied by ϕ.

□

We conclude this section with some additional results about implication in context that will be needed in the next section. These involve a notion of entailment between contexts. Formally, we say that Γ *entails* Γ' if $R_\Gamma \subseteq R_{\Gamma'}$ and $\Gamma\{\phi\}$ implies $\Gamma'\{\phi\}$ for all formulas ϕ. We say that that Γ and Γ' are *equivalent* if Γ entails Γ' and Γ' entails Γ.

Lemma 24. *If ψ implies ψ' in context Γ, then $\Gamma[(x = y \wedge [\,]) \vee \psi]$ entails $\Gamma[(x = y \wedge [\,]) \vee \psi']$.*

Proof. Let $\Delta = \Gamma[(x = y \wedge [\,]) \vee \psi]$ and let $\Delta' = \Gamma[(x = y \wedge [\,]) \vee \psi']$. Then $R_\Delta = R_\Gamma + \{x = y\} = R_{\Delta'}$. Since ψ implies ψ' in context Γ, $\Gamma\{\psi\}$ implies $\Gamma\{\psi'\}$. But $\Gamma\{\psi\}$ is equivalent to $\Delta\{\mathbf{F}\}$ and $\Gamma\{\psi'\}$ is equivalent to $\Delta'\{\mathbf{F}\}$, hence $\Delta\{\mathbf{F}\}$ implies $\Delta'\{\mathbf{F}\}$. Let ϕ be a formula, then by Lemma 17, $\Delta\{\phi\} = \Delta\{\mathbf{F}\} \vee (\chi_{R_\Delta} \wedge \phi)$ and $\Delta'\{\phi\} = \Delta'\{\mathbf{F}\} \vee (\chi_{R_{\Delta'}} \wedge \phi)$. Since $R_\Delta = R_{\Delta'}$ and $\Delta\{\mathbf{F}\}$ implies $\Delta'\{\mathbf{F}\}$, it follows that $\Delta\{\phi\}$ implies $\Delta'\{\phi\}$. Since this is true for an arbitrary formula ϕ, Δ entails Δ'. □

Lemma 25. *If Γ entails Γ', then $\Gamma[(x = y \wedge [\,]) \vee \psi]$ entails $\Gamma'[(x = y \wedge [\,]) \vee \psi]$.*

Proof. Let $\Delta = \Gamma[(x = y \wedge [\,]) \vee \psi]$ and let $\Delta' = \Gamma'[(x = y \wedge [\,]) \vee \psi]$. Since Γ entails Γ', it follows that $R_\Gamma \subseteq R_{\Gamma'}$. Then $R_\Delta = R_\Gamma + \{x = y\} \subseteq R_{\Gamma'} + \{x = y\} = R_{\Delta'}$. Also, since Γ entails Γ', we have $\Gamma\{\phi\}$ implies $\Gamma'\{\phi\}$ for all formulas ϕ. In particular, $\Gamma\{(x = y \wedge \phi) \vee \psi\}$ implies $\Gamma'\{(x = y \wedge \phi) \vee \psi\}$ for all formulas ϕ. But $\Gamma\{(x = y \wedge \phi) \vee \psi\} = \Delta\{\phi\}$ and $\Gamma'\{(x = y \wedge \phi) \vee \psi\} = \Delta'\{\phi\}$. Thus for all formulas ϕ we have that $\Delta\{\phi\}$ implies $\Delta'\{\phi\}$.

We have shown that $R_\Delta \subseteq R_{\Delta'}$ and $\Delta\{\phi\}$ implies $\Delta'\{\phi\}$ for all formulas ϕ. Thus, Δ entails Δ'. □

Inductive application of Lemmas 24 and 25 shows that if Γ and Γ' have the same structure, with the same equations at the same positions, except that the formulas ψ in Γ imply their counterparts ψ' in Γ', then Γ entails Γ'.

Lemma 26. *Suppose Γ entails Γ'. Then:*

1. *If Γ is degenerate then so is Γ'.*
2. *If ϕ is reduced in context Γ', then ϕ is also reduced in context Γ.*

Proof. 1. If Γ is degenerate, then by Corollary 3, R_Γ satisfies $\Gamma\{\mathbf{F}\}$. Since $R_\Gamma \subseteq R_{\Gamma'}$ by hypothesis, $R_{\Gamma'}$ satisfies $\Gamma\{\mathbf{F}\}$ by monotonicity. Since Γ entails Γ', $\Gamma\{\mathbf{F}\}$ implies $\Gamma'\{\mathbf{F}\}$. Hence $R_{\Gamma'}$ satisfies $\Gamma'\{\mathbf{F}\}$, thus showing Γ' is degenerate by Corollary 3.

2. By induction on formulas. If ϕ is $\mathbf{F}$, then ϕ is reduced in any context, so the result holds. Suppose ϕ is $\mathbf{T}$ and ϕ is reduced in context Γ'. Then Γ' is nondegenerate. Hence Γ is nondegenerate by (1). Thus ϕ is reduced in context Γ.
   Suppose ϕ is $(x = y \wedge \phi_1) \vee \phi_2$ and ϕ is reduced in context Γ'. Then Γ' is nondegenerate. Hence Γ is nondegenerate by (1). By (R1), ϕ_1 is reduced in context $\Gamma'[(x = y \wedge [\,]) \vee \phi_2]$. Since Γ entails Γ', it follows by Lemma 25 that $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$ entails $\Gamma'[(x = y \wedge [\,]) \vee \phi_2]$, hence ϕ_1 is reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$ by induction.
   By (R2), ϕ_2 is reduced in context Γ'. Then ϕ_2 is reduced in context Γ by induction.
   By (R3), x and y are the greatest elements of their respective $R_{\Gamma'}$-equivalence classes. Since $R_\Gamma \subseteq R_{\Gamma'}$, x and y are also the the greatest elements of their respective R_Γ-equivalence classes.

```
fun REDUCE($\phi, \Gamma$) =
  if $\phi = \mathbf{F}$ orelse DEGENERATE($\Gamma$) then $\mathbf{F}$
  else if $\phi = \mathbf{T}$ then $\mathbf{T}$
  else
    let $(x = y \wedge \phi_1) \vee \phi_2$ be $\phi$
        $\psi_2 = $ REDUCE($\phi_2, \Gamma$)
        $(x', y') = $ if $\max_{R_\Gamma}[x] \succ \max_{R_\Gamma}[y]$ then
                      $(\max_{R_\Gamma}[x], \max_{R_\Gamma}[y])$
                    else
                      $(\max_{R_\Gamma}[y], \max_{R_\Gamma}[x])$
    in
      if $x' = y'$ then OR(REDUCE($\phi_1, \Gamma$), $\psi_2, \Gamma$)
      else
        let $\psi_1 = $ REDUCE($\phi_1, \Gamma[(x' = y' \wedge [\,]) \vee \psi_2]$)
        in
          REORDER($x' = y', \psi_1, \psi_2, \Gamma$)
        end
    end
```

Fig. 5. Function REDUCE

We have shown that Γ is nondegenerate, ϕ_1 is reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, that ϕ_2 is reduced in context Γ, and that x and y are the greatest elements of their respective R_Γ-equivalence classes. Hence ϕ is reduced in context Γ.

□

Corollary 4. *If Γ and Γ' are equivalent, then ϕ is reduced in context Γ if and only if it is reduced in context Γ'.*

Proof. Immediate from Lemma 26. □

6 Reduction Algorithm

In this section we present an algorithm for reducing an arbitrary decision form to a normal form, and we sketch the proof of correctness of this algorithm.

Code for function REDUCE is presented in Figure 5 in an ML-like functional style. REDUCE takes a decision form ϕ and a context Γ, and it returns a decision form ϕ', which is ordered and reduced in context Γ and which is therefore a normal form. In general, it is *not* the case that ϕ' is equivalent to ϕ. However, what *is* true is that ϕ' is equivalent to ϕ in context Γ.

Function REDUCE makes use of three auxiliary functions: SAT$'$, OR, and REORDER. Function SAT$'$ (shown in Figure 3) takes an equivalence relation R and a normal form ϕ and returns *true* if and only if $R \models \phi$ holds. Function OR (shown in Figure 8) takes as arguments two decision forms ϕ_1 and ϕ_2 and a nondegenerate context Γ, where ϕ_1 and ϕ_2 are assumed to be ordered and reduced in context Γ. It returns a decision form ϕ, which is ordered and reduced in context Γ, and which is equivalent to $\phi_1 \vee \phi_2$ in context Γ. Function Reorder (shown in Figure 9) takes as arguments an equation $x = y$, a decision form ϕ_1, a decision form ϕ_2, and a nondegenerate context Γ, where ϕ_2 is assumed to be ordered and reduced in context Γ, and ϕ_1 is assumed to be ordered and reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$. It returns a decision form ϕ, which is ordered and reduced in context Γ, and which is equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ. That is, Reorder "fixes up" any problems that occur in the construction of $(x = y \wedge \phi_1) \vee \phi_2$ in case one or both of ϕ_1 and ϕ_2 fails to be ordered strictly below $x = y$. Function Or makes use of an additional auxiliary function Filter (shown in Figure 7). The purpose of a call to Filter(ϕ, Γ) is to "filter out" any branches of ϕ whose associated implicants are subsumed by Γ. Finally, both Or and Reorder make use of auxiliary function Build, which builds a decision form equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ such that (R1) and (R2) are satisfied.

The following results formally state the correctness conditions for function Reduce and the auxiliary functions.

Lemma 27 (Correctness of Build). *If $x = y$ is an equation, and ϕ_1 and ϕ_2 are decision forms, such that ϕ_1 and ϕ_2 are ordered strictly below $x = y$, ϕ_2 is reduced in some nondegenerate context Γ, ϕ_1 is reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, and x and y are the greatest elements in their respective equivalence classes of R_Γ, then* Build$(x = y, \phi_1, \phi_2)$ *terminates and returns a decision form ϕ that is ordered below $x = y$, reduced in context Γ, and equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ.*

Proof. Termination of Build is obvious. If $\phi_1 = \mathbf{F}$ or $\phi_2 = \mathbf{T}$, then $\phi = \phi_2$. In this case, since ϕ_2 is assumed to be ordered strictly below $x = y$ and reduced in context Γ, it is clear that ϕ is ordered below $x = y$ and reduced in context Γ. In addition, ϕ is (unconditionally) equivalent to $(x = y \wedge \mathbf{F}) \vee \phi_2$, hence equivalent to it in context Γ.

Otherwise, ϕ_1 is not $\mathbf{F}$, ϕ_2 is not $\mathbf{T}$, and ϕ is $(x = y \wedge \phi_1) \vee \phi_2$. Obviously, ϕ is then equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ. Since ϕ_2 is assumed to be ordered strictly below $x = y$, reduced in context Γ, and it is not $\mathbf{T}$, and ϕ_1 is assumed to be ordered strictly below $x = y$, reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, and is not $\mathbf{F}$, ϕ satisfies conditions (R1), (R2), (O1), and (O2). Since x and y are assumed to be the greatest elements of their respective equivalence classes of R_Γ, it follows that ϕ also satisfies condition (R3). Since Γ is assumed nondegenerate, ϕ is reduced in context Γ. Since ϕ clearly also satisfies (O3) it is ordered below $x = y$. □

Lemma 28 (Correctness of Filter). *If ϕ is a normal form and Γ is a context, then* Filter(ϕ, Γ) *terminates and returns a decision form ϕ' that is ordered, reduced in context Γ, and equivalent to ϕ in context Γ. Moreover, if ϕ is ordered below (resp. strictly below) some equation $x' = y'$, then ϕ' is as well.*

Proof. Since Build and Degenerate have already been shown to terminate, and since each recursive call to Filter is made on a proper subformula of ϕ, the termination of Filter is clear.

```
fun BUILD($x = y, \phi_1, \phi_2$) =
  if $\phi_1 = \mathbf{F}$ orelse $\phi_2 = \mathbf{T}$ then $\phi_2$
  else $(x = y \wedge \phi_1) \vee \phi_2$
```

Fig. 6. Function BUILD

```
fun FILTER($\phi, \Gamma$) =
  if $\phi = \mathbf{F}$ orelse DEGENERATE($\Gamma$) then $\mathbf{F}$
  else if $\phi = \mathbf{T}$ then $\mathbf{T}$
  else
    let
      $(x = y \wedge \phi_1) \vee \phi_2$ be $\phi$
      $\phi'_2 = $ FILTER($\phi_2, \Gamma$)
      $\phi'_1 = $ FILTER($\phi_1, \Gamma[(x = y \wedge [\,]) \vee \phi'_2]$)
    in
      BUILD($x = y, \phi'_1, \phi'_2$)
    end
```

Fig. 7. Function FILTER

The proof of partial correctness of FILTER is by induction on the depth of recursive calls. We first consider the cases in which there are no recursive calls. If $\phi = \mathbf{F}$ or DEGENERATE(Γ) holds, then $\phi' = \mathbf{F}$. Clearly $\mathbf{F}$ is ordered and reduced in context Γ. If $\phi = \mathbf{F}$ or DEGENERATE(Γ) holds, then clearly ϕ' is reduced in context Γ and equivalent to ϕ in context Γ. Obviously also in this case ϕ' is ordered strictly below an arbitrary equation $x' = y'$.

If $\phi = \mathbf{T}$ and DEGENERATE(Γ) does not hold, then $\phi' = \mathbf{T}$. Clearly then ϕ' is ordered and equivalent to ϕ in any context. Since Γ is nondegenerate, ϕ is reduced in context Γ. Obviously also in this case ϕ' is ordered strictly below an arbitrary equation $x' = y'$.

Otherwise, $\phi = (x = y \wedge \phi_1) \vee \phi_2$ and DEGENERATE(Γ) does not hold. Since ϕ is assumed to be a normal form, it follows that ϕ_1 and ϕ_2 are normal forms ordered strictly below $x = y$. By induction, $\phi'_2 = $ FILTER(ϕ_2, Γ) is ordered strictly below $x = y$, reduced in context Γ, and equivalent to ϕ_2 in context Γ. Also by induction, $\phi'_1 = $ FILTER($\phi_1, \Gamma[(x = y \wedge [\,]) \vee \phi'_2]$) is ordered strictly below $x = y$, reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi'_2]$, and equivalent to ϕ_1 in context $\Gamma[(x = y \wedge [\,]) \vee \phi'_2]$. Since Γ is nondegenerate, it follows that $\phi' = $ BUILD($x = y, \phi'_1, \phi'_2$) is ordered, reduced in context Γ, and equivalent to $(x = y \wedge \phi_1) \vee \phi_2$, *i.e.* to ϕ, in context Γ. Moreover, in this case if ϕ is ordered below (resp. strictly below) $x' = y'$, then $x' = y' \succeq x = y$ (resp. $x' = y' \succ x = y$) and ϕ' is ordered below (resp. strictly below) $x' = y'$ as well. □

Lemma 29 (Correctness of OR). *Suppose decision forms ϕ_1 and ϕ_2 are ordered and reduced in the nondegenerate context Γ. Then* OR(ϕ_1, ϕ_2, Γ) *terminates and returns a decision form ϕ that is ordered, reduced in context Γ, and* *equivalent to $\phi_1 \vee \phi_2$ in context Γ. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then ϕ is as well.*

Proof. Since Filter and Build have already been shown to terminate, and each recursive call within Or is made on a proper subformula of either ϕ_1 or ϕ_2, or both, the termination of Or is clear.

The proof of partial correctness of Or is by induction on the depth of recursive calls. We first consider the cases in which there are no recursive calls. In these cases, either one of ϕ_1 or ϕ_2 is $\mathbf{T}$, or ϕ_1 is $\mathbf{F}$, or ϕ_2 is $\mathbf{F}$. If one of ϕ_1 or ϕ_2 is $\mathbf{T}$, then the formula ϕ' returned by $\textsc{Or}(\phi_1, \phi_2, \Gamma)$ is $\mathbf{T}$, which is clearly ordered strictly below an arbitrary equation $x = y$ and is equivalent to $\phi_1 \vee \phi_2$ in context Γ. Moreover, since Γ is assumed nondegenerate, it follows that ϕ' is reduced in context Γ.

If ϕ_1 is $\mathbf{F}$, then ϕ' is ϕ_2, which is then clearly equivalent to $\phi_1 \vee \phi_2$ in context Γ. By hypothesis, ϕ' is ordered and reduced in context Γ. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then clearly ϕ' is as well. If ϕ_2 is $\mathbf{F}$, then symmetric reasoning applies.

Otherwise, we are in the main body of Or, $\phi_1 = (x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$, and $\phi_2 = (x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$. By hypothesis, Γ is nondegenerate. Moreover, from (R3), x_1, y_1, x_2, and y_2 are each the $\succ$-maximum elements of their respective R_Γ-equivalence classes. There are now three cases:

1. $(x_1, y_1) = (x_2, y_2)$. In this case, $\phi_1 \vee \phi_2$ is (unconditionally) equivalent to $(x_1 = y_1 \wedge (\phi_{11} \vee \phi_{21})) \vee (\phi_{12} \vee \phi_{22})$, hence these formulas are also equivalent in context Γ. Since ϕ_1 and ϕ_2 are assumed ordered and reduced in context Γ, it follows that ϕ_{12} is ordered strictly below $x_1 = y_1$, ϕ_{22} is ordered strictly below $x_2 = y_2$ (which is the same as $x_1 = y_1$), and both are reduced in context Γ. Thus, by induction, $\psi_2 = \textsc{Or}(\phi_{12}, \phi_{22}, \Gamma)$ is ordered strictly below $x_1 = y_1$, reduced in context Γ, and equivalent to $\phi_{12} \vee \phi_{22}$ in context Γ. Also, since ϕ_1 and ϕ_2 are assumed ordered and reduced in context Γ, it follows that ϕ_{11} and ϕ_{21} are normal forms ordered strictly below $x_1 = y_1$. Thus, $\phi'_{11} = \textsc{Filter}(\phi_{11}, \Gamma')$ and $\phi'_{21} = \textsc{Filter}(\phi_{21}, \Gamma')$ are ordered strictly below $x_1 = y_1$, reduced in context $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$, and equivalent to ϕ_{11} and ϕ_{21}, respectively, in context Γ'. It then follows by induction that $\psi_1 = \textsc{Or}(\phi'_{11}, \phi'_{21}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to $\phi'_{11} \vee \phi'_{21}$ in context Γ'. Since ϕ'_{11} is equivalent to ϕ_{11} in context Γ' and ϕ'_{21} is equivalent to ϕ_{21} in context Γ', by Corollary 2, ψ_1 is equivalent to $\phi_{11} \vee \phi_{21}$ in context Γ'. The preconditions for Build are therefore satisfied, and we have that $\phi' = \textsc{Build}(x_1 = y_1, \psi_1, \psi_2)$ is ordered below $x_1 = y_1$, reduced in context Γ, and equivalent to $(x_1 = y_1 \wedge \psi_1) \vee \psi_2$ in context Γ. Since ψ_1 is equivalent to $\phi_{11} \vee \phi_{21}$ in context Γ', and ψ_2 is equivalent to $\phi_{12} \vee \phi_{22}$ in context Γ, it follows by Lemmas 18 and 19 that ϕ is equivalent in context Γ to $(x_1 = y_1 \wedge (\phi_{11} \vee \phi_{21})) \vee (\phi_{12} \vee \phi_{22})$; hence also equivalent in context Γ to $(x_1 = y_1 \wedge \phi_1) \vee \phi_2$. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then $x = y \succeq x_1 = y_1$ (resp. $x = y \succ x_1 = y_1$), from which it follows that ϕ' is ordered below (resp. strictly below) $x = y$ as well.

2. $(x_1, y_1) \succ (x_2, y_2)$. In this case, $\phi_1 \vee \phi_2$ is (unconditionally) equivalent to $(x_1 = y_1 \wedge \phi_{11}) \vee (\phi_{12} \vee \phi_2)$. Now by assumption, ϕ_1 is ordered below $x_1 = y_1$ and reduced in context Γ, so ϕ_{12} is ordered strictly below $x_1 = y_1$ and is reduced in context Γ. Also by assumption, ϕ_2 is ordered below $x_2 = y_2$ and reduced in context Γ, so since $x_1 = y_1 \succ x_2 = y_2$, it is also ordered strictly below $x_1 = y_1$. Then by induction $\psi_2 = \textsc{Or}(\phi_{12}, \phi_2, \Gamma)$ is ordered strictly below $x_1 = y_1$, reduced in context Γ, and equivalent to $\phi_{12} \vee \phi_2$ in context Γ. Also, ϕ_{11} is a normal form ordered strictly below $x_1 = y_1$, hence $\psi_1 = \textsc{Filter}(\phi_{11}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$, and equivalent to ϕ_{11} in context Γ'. The preconditions for $\textsc{Build}$ are therefore satisfied, and we have that $\phi' = \textsc{Build}(x_1 = y_1, \psi_1, \psi_2)$ is ordered, reduced in context Γ, and equivalent to $(x_1 = y_1 \wedge \psi_1) \vee \psi_2$. Since ψ_1 is equivalent to ϕ_{11} in context Γ', and ψ_2 is equivalent to $\phi_{12} \vee \phi_2$ in context Γ, it follows by Lemmas 18 and 19 that ϕ is equivalent in context Γ to $(x_1 = y_1 \wedge \phi_{11}) \vee (\phi_{12} \vee \phi_2)$; hence also equivalent in context Γ to $\phi_1 \vee \phi_2$. Moreover, if both ϕ_1 and ϕ_2 are ordered below (resp. strictly below) some equation $x = y$, then $x = y \succeq x_1 = y_1$ (resp. $x = y \succ x_1 = y_1$), hence ϕ' is ordered below (resp. strictly below) $x = y$ as well.
3. $(x_2, y_2) \succ (x_1, y_1)$. This case is symmetric to case (2).

□

Lemma 30. *Suppose x and y are each the greatest elements of their respective R-equivalence classes, that x_1 and y_1 are each the greatest elements in their respective $(R + \{x = y\})$-equivalence classes, that $x \succ y$, $x_1 \succ y_1$, and that $x_1 = y_1 \succ x = y$. Let $R' = R + \{x_1 = y_1\}$. Then $\max_{R'}[y] = y$ and either $\max_{R'}[x] = x_1$ or $\max_{R'}[x] = x$.*

Proof. Since $x_1 = y_1 \succ x = y$, either $x_1 \succ x$ or else $x_1 = x$ and $y_1 \succ y$. First, consider the case in which $x_1 = x$ and $y_1 \succ y$. Since $x_1 \succ y_1$ and $x \succ y$ by hypothesis, it follows that y, x_1, and y_1 are in distinct R-equivalence classes. But then the R'-equivalence class of y is the same as its R-equivalence class, so that $\max_{R'}[y] = y$. Also, the R'-equivalence class of x is the union of its R-equivalence class (*i.e.* the R-equivalence class of x_1) and the R-equivalence class of y_1. Since by assumption x is the greatest element of its R-equivalence class, $x = x_1 \succ y_1$, and y_1 is the greatest element of its $(R + \{x = y\})$-equivalence class, hence also of its R-equivalence class, it follows that $x \succ y'$ for every element y' of the R-equivalence class of y_1. Thus, x is the greatest element of its R'-equivalence class and we have $\max_{R'}[x] = x$.

Now, suppose $x_1 \succ x$. We distinguish two sub-cases: $y_1 = y$ and $y_1 \neq y$. If $y_1 = y$, then since $x \succ y$ we would have a contradiction with the assumption that y_1 is the greatest element of its $(R+\{x = y\})$-equivalence class. This case is therefore impossible. Suppose $y_1 \neq y$. In this case, since $x_1 \succ y_1$, the R-equivalence classes of x_1, y_1, and y are all distinct, from which it follows that the R'-equivalence class of y is the same as its R-equivalence class, hence $\max_{R'}[y] = y$.

If x and y_1 are in the same R-equivalence class, then $\max_{R'}[x] = x_1$, otherwise the R'-equivalence class of x is the same as its R-equivalence class and $\max_{R'}[x] = x$. □

Lemma 31 (Correctness of Reorder**).** *If $x = y$ is an equation, ϕ_1 and ϕ_2 are decision forms, and Γ is a nondegenerate context, such that ϕ_1 and ϕ_2 are ordered, ϕ_2 is reduced in context Γ, ϕ_1 is reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, and x and y are the greatest elements of their respective R_Γ-equivalence classes, then*

- Reorder$(x = y, \phi_1, \phi_2, \Gamma)$ *terminates and returns a decision form ϕ that is ordered, reduced in context Γ, and equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ.*
- *If ϕ_2 is ordered strictly below $x = y$, then* Reorder1$(x = y, \phi_1, \phi_2, \Gamma)$ *terminates and returns a decision form ϕ that is ordered, reduced in context Γ and equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ.*

Moreover, in either of the above two cases, if $x' = y'$ is any equation such that $x' = y' \succeq x = y$ (resp. $x' = y' \succ x = y$) and such that both ϕ_1 and ϕ_2 are ordered strictly below $x' = y'$, then ϕ is ordered below (resp. strictly below) $x' = y'$.

Proof. Since Build and Or have already been shown to terminate, the only potential sources of nontermination are the recursive calls from Reorder to itself and the mutually recursive calls between Reorder and Reorder1. However, it is easy to verify that each in each call to Reorder either from itself or from Reorder1, either ϕ_2 is replaced by one of its proper subformulas (if we regard $\mathbf{F}$ as a proper subformula of any formula of the form $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$), or else leaves ϕ_2 alone and replaces ϕ_1 by one of its proper subformulas. Thus, the arguments to each successive call to Reorder decrease according to a lexicographic order on pairs (ϕ_1, ϕ_2) with ϕ_2 in the "most significant" position. Termination of Reorder and Reorder1 is thus guaranteed.

The proof of partial correctness of Reorder and Reorder1 is by induction on the depth of recursive calls. As the induction hypothesis we assume the correctness of recursive calls to Reorder and Reorder1, and prove the correctness of a main call to Reorder or Reorder1. In each case, the structure of the correctness proof is a case analysis that follows the case structure of the code.

First, consider Reorder. If $\phi_2 = \mathbf{T}$, then $\phi = \phi_2$. By hypothesis, ϕ is ordered and reduced in context Γ. Moreover, ϕ is (unconditionally) equivalent to $(x = y \wedge \phi_1) \vee \phi_2$, hence it is equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ. Obviously, in this case, ϕ is ordered strictly below $x' = y'$ for any equation $x' = y'$. Next, suppose $\phi_2 = \mathbf{F}$. Then ϕ_2 is ordered strictly below $x = y$, so the preconditions for the recursive call to Reorder1$(x = y, \phi_1, \phi_2, R)$ are satisfied. By induction, the formula returned by this call has the properties asserted of the formula ϕ returned by the main call.

Now suppose that ϕ_2 has the form $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$. We consider three cases:

– Suppose $(x, y) = (x_2, y_2)$. Let $\psi_1 = \textsc{Or}(\phi_1, \phi_{21}, \Gamma')$, where Γ' is the context $\Gamma[(x = y \wedge [\,]) \vee \phi_{22}])$. Since ϕ_{22} implies ϕ_2, it follows by Lemma 24 that Γ' entails $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$. Since ϕ_1 is assumed ordered and reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, by Lemma 26, ϕ_1 is also ordered and reduced in context Γ'. Additionally, ϕ_2 is assumed ordered and reduced in context Γ, hence by (O1) and (R1) ϕ_{21} is ordered and reduced in context $\Gamma[(x_2 = y_2 \wedge [\,]) \vee \phi_{22}]$, which is the same as Γ'. The preconditions for the call to $\textsc{Or}(\phi_1, \phi_{21}, \Gamma')$ are therefore satisfied, and it follows that the formula ψ_1 returned is ordered and reduced in context Γ', and it is equivalent to $\phi_1 \vee \phi_{21}$ in context Γ'. Moreover, for any equation $x' = y'$ such that ϕ_1 and ϕ_{21} are both ordered strictly below $x' = y'$, we have that ψ_1 is ordered strictly below $x' = y'$ as well.
  Since x and y are the greatest elements in their respective R_Γ-equivalence classes and ϕ_{22} is ordered and reduced in context Γ, the preconditions for the recursive call to $\textsc{Reorder}(x = y, \psi_1, \phi_{22}, \Gamma)$ are therefore satisfied. It then follows by induction that $\phi = \textsc{Reorder}(x = y, \psi_1, \phi_{22}, \Gamma)$ is ordered, reduced in context Γ, and equivalent to $(x = y \wedge \psi_1) \vee \phi_{22}$ in context Γ. Since ψ_1 is equivalent to $\phi_1 \vee \phi_{21}$ in context $\Gamma[(x = y \wedge [\,]) \vee \phi_{22}]$, by Lemma 18 it follows that ϕ is equivalent to $(x = y \wedge (\phi_1 \vee \phi_{21})) \vee \phi_{22}$ in context Γ. Since by assumption $x_2 = y_2$ is the same equation as $x = y$, $(x = y \wedge (\phi_1 \vee \phi_{21})) \vee \phi_{22}$ is equivalent to $(x = y \wedge \phi_1) \vee ((x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22})$, hence by Lemma 15, ϕ is equivalent to to $(x = y \wedge \phi_1) \vee ((x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22})$ in context Γ; that is, it is equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context Γ. Moreover, suppose $x' = y'$ is any equation such that $x' = y' \succeq x = y$ (resp. $x' = y' \succ x = y$) and such that ϕ_1 and ϕ_2 are ordered strictly below $x' = y'$. Then ϕ_{21} and ϕ_{22} are also ordered strictly below $x' = y'$. As we have noted above, it then follows that ψ_1 is ordered strictly below $x' = y'$. By induction, $\phi = \textsc{Reorder}(x = y, \psi_1, \phi_{22}, R)$ is ordered below (resp. strictly below) $x' = y'$.
– Suppose $(x_2, y_2) \succ (x, y)$. Since ϕ_2 is assumed ordered and reduced in context Γ, by (R3) it follows that x_2 and y_2 are the greatest elements in their respective R_Γ-equivalence classes, by (R1) and (O1) ϕ_{21} is ordered strictly below $x_2 = y_2$ and reduced in context $\Gamma[(x_2 = y_2 \wedge [\,]) \vee \phi_{22}]$. Because $\mathbf{F}$ implies ϕ_{22}, context $\Gamma[(x_2 = y_2 \wedge [\,]) \vee \mathbf{F}]$ entails context $\Gamma[(x_2 = y_2 \wedge [\,]) \vee \phi_{22}]$ by Lemma 24, from which it follows by Lemma 26 that ϕ_{21} is also reduced in context $\Gamma[(x_2 = y_2 \wedge [\,]) \vee \mathbf{F}]$. Obviously, $\mathbf{F}$ is ordered strictly below $x_2 = y_2$ and reduced in context Γ. The preconditions for $\textsc{Build}$ are therefore satisfied, and we have that $\psi_1 = \textsc{Build}(x_2 = y_2, \phi_{21}, \mathbf{F})$ is ordered below $x_2 = y_2$, reduced in context Γ, and equivalent to $(x_2 = y_2 \wedge \phi_{21}) \vee \mathbf{F}$ in context Γ.
  Also, ϕ_1 is assumed ordered and reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, so because ϕ_{22} implies ϕ_2, $\Gamma[(x = y \wedge [\,]) \vee \phi_{22}]$ entails $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$ by Lemma 24, hence by Lemma 26 ϕ_1 is also reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_{22}]$. Since ϕ_2 is assumed ordered and reduced in context Γ, ϕ_{22} is also reduced in context Γ. The preconditions for $\textsc{Reorder}$ are therefore satisfied, and we have by induction that $\psi_2 = \textsc{Reorder}(x = y, \phi_1, \phi_{22}, \Gamma)$ is ordered, reduced in context Γ, and equivalent to $(x = y \wedge \phi_1) \vee \phi_{22}$ in context Γ.

Now, suppose $x' = y'$ is any equation such that $x' = y' \succeq x = y$ (resp. $x' = y' \succ x = y$) and such that ϕ_1 and ϕ_2 are ordered strictly below $x' = y'$. Since ϕ_2 is ordered strictly below $x' = y'$, we know that $x' = y' \succ x_2 = y_2$ and that ϕ_{22} is ordered strictly below $x' = y'$, hence ψ_2 is also ordered below (resp. strictly below) $x' = y'$. We have already noted that ψ_1 is ordered below $x_2 = y_2$, so ψ_1 is ordered strictly below $x' = y'$. Thus, by Lemma 29, $\phi = \textsc{Or}(\psi_1, \psi_2, \mathit{\Gamma})$ is ordered below (resp. strictly below) $x' = y'$, is reduced in context $\mathit{\Gamma}$, and is equivalent to $\psi_1 \vee \psi_2$ in context $\mathit{\Gamma}$. Thus by Corollary 2, ϕ is equivalent in context $\mathit{\Gamma}$ to $((x_2 = y_2 \wedge \phi_{21}) \vee \mathbf{F}) \vee ((x = y \wedge \phi_1) \vee \phi_{22})$, hence by Lemma 15 to $(x = y \wedge \phi_1) \vee ((x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22})$; that is, to $(x = y \wedge \phi_1) \vee \phi_2$.

– Suppose $(x, y) \succ (x_2, y_2)$. In this case, ϕ_2 is ordered strictly below $x = y$. The preconditions for $\textsc{Reorder1}$ thus hold, and by induction the formula $\phi = \textsc{Reorder1}(x = y, \phi_1, \phi_2)$ is ordered, reduced in context $\mathit{\Gamma}$, and equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context $\mathit{\Gamma}$. Moreover, if $x' = y'$ is an equation such that $x' = y' \succeq x = y$ (resp. $x' = y' \succ x = y$) and both ϕ_1 and ϕ_2 are ordered strictly below $x' = y'$, then ϕ is ordered below (resp. strictly below) $x' = y'$.

Now, consider $\textsc{Reorder1}$. If $\phi_1 = \mathbf{F}$, then $\phi = \phi_2$. By hypothesis, ϕ is ordered and reduced in context $\mathit{\Gamma}$. Moreover, ϕ is (unconditionally) equivalent to $(x = y \wedge \phi_1) \vee \phi_2$, hence it is equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context $\mathit{\Gamma}$. Obviously, in this case, ϕ is ordered strictly below $x' = y'$ for any equation $x' = y'$ such that $x' = y' \succeq x = y$ and such that both ϕ_1 and ϕ_2 are ordered strictly below $x' = y'$.

Suppose now that $\phi_1 = \mathbf{T}$. In this case, ϕ is given by $\textsc{Build}(x = y, \phi_1, \phi_2)$, and it is clear that ϕ_1 is ordered strictly below $x = y$ and that the remaining preconditions for the call to $\textsc{Build}$ are implied by the preconditions for the main call. Thus, ϕ is ordered below $x = y$, reduced in context $\mathit{\Gamma}$, and equivalent to $(x = y \wedge \phi_1) \vee \phi_2$ in context $\mathit{\Gamma}$. Clearly then ϕ is ordered below (resp. strictly below) $x' = y'$ for any equation $x' = y'$ such that $x' = y' \succeq x = y$ (resp. $x' = y' \succ x = y$).

It remains to consider the case in which ϕ_1 has the form $(x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$. Note that it is impossible in this case to have $(x_1, y_1) = (x, y)$, because ϕ_1 is assumed reduced in context $\mathit{\Gamma}[(x = y \wedge [\,]) \vee \phi_2]$, hence it cannot contain any occurrences of y. If $(x, y) \succ (x_1, y_1)$, then both ϕ_1 and ϕ_2 are ordered strictly below $x = y$ and the result is established as in the case that $\phi_1 = \mathbf{T}$.

If $(x_1, y_1) \succ (x, y)$, then the argument is as follows:

– Since ϕ_1 is assumed ordered and reduced in context $\mathit{\Gamma}[(x = y \wedge [\,]) \vee \phi_2]$, it follows that ϕ_{12} is also ordered and reduced in context $\mathit{\Gamma}[(x = y \wedge [\,]) \vee \phi_2]$. By assumption, x and y are the greatest elements in their respective $R_{\mathit{\Gamma}}$-equivalence classes. The preconditions for the recursive call to $\textsc{Reorder}(x = y, \phi_{12}, \phi_2, \mathit{\Gamma})$ are therefore satisfied, and it follows by induction that ψ_2 is ordered, reduced in context $\mathit{\Gamma}$, and equivalent to $(x = y \wedge \phi_{12}) \vee \phi_2$ in context $\mathit{\Gamma}$. Moreover, since ϕ_1 is ordered, ϕ_{12} is ordered strictly below $x_1 = y_1$. By hypothesis, ϕ_2 is ordered strictly below $x = y$, thus since $(x_1, y_1) \succ (x, y)$, it follows that ϕ_2 is ordered strictly below $x_1 = y_1$. But then ψ_2 is also ordered strictly below $x_1 = y_1$.

- By hypothesis, x and y are the greatest elements of their respective R_Γ-equivalence classes. Let $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$ and let $R' = R_{\Gamma'}$; then $R' = R_\Gamma + \{x_1 = y_1\}$. Since ϕ_1 is assumed reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, it follows that x_1 and y_1 are the greatest elements of their respective $(R_\Gamma + \{x = y\})$-equivalence classes. By Lemma 30, $\max_{R'}[y] = y$ and either $\max_{R'}[x] = x_1$ or $\max_{R'}[x] = x$, so that $\max_{R'}[x] \succ y$ and $x_1 = y_1 \succ \max_{R'}[x] = y$.

  Since ϕ_1 is assumed reduced in context $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$, it follows by (R1) that ϕ_{11} is reduced in context $\Gamma[(x = y \wedge [(x_1 = y_1 \wedge [\,]) \vee \phi_{12}]) \vee \phi_2]$, hence by Corollary 4 also in the equivalent context

$$\Gamma[(x_1 = y_1 \wedge [(x = y \wedge [\,]) \vee \mathbf{F}]) \vee ((x = y \wedge \phi_{12}) \vee \phi_2)].$$

  Since ψ_2 is equivalent to $(x = y \wedge \phi_{12}) \vee \phi_2$ in context Γ, by Lemma 24 we have that ϕ_{11} is reduced in context

$$\Gamma[(x_1 = y_1 \wedge [(x = y \wedge [\,]) \vee \mathbf{F}]) \vee \psi_2];$$

  that is, in context

$$\Gamma'[(x = y \wedge [\,]) \vee \mathbf{F}].$$

  Since $R' + \{x = y\} = R' + \{\max_{R'}[x] = y\}$, context $\Gamma'[(x = y \wedge [\,]) \vee \mathbf{F}]$ is equivalent to $\Gamma'[(\max_{R'}[x] = y \wedge [\,]) \vee \mathbf{F}]$, hence ϕ_{11} is also reduced in context $\Gamma'[(\max_{R'}[x] = y \wedge [\,]) \vee \mathbf{F}]$ by Corollary 4. Clearly $\mathbf{F}$ is reduced in context Γ'. By definition $\max_{R'}[x]$ is the greatest element in its R'-equivalence class, and we have established above that $\max_{R'}[y] = y$. The preconditions for the recursive call to Reorder($\max_{R'}[x] = y, \phi_{11}, \mathbf{F}, \Gamma'$) are therefore satisfied, and it follows by induction that ψ_1 is ordered, reduced in context Γ', and equivalent to $(x = y \wedge \phi_{11}) \vee \mathbf{F}$ in context Γ'. Moreover, if $x' = y'$ is any equation for which $x' = y' \succ \max_{R'}[x] = y$ and such that ϕ_{11} and $\mathbf{F}$ are ordered strictly below $x' = y'$, then ψ_1 is ordered strictly below $x' = y'$. In particular, take $x' = y'$ to be $x_1 = y_1$. Then both ϕ_{11} and $\mathbf{F}$ are ordered strictly below $x_1 = y_1$, and we have already argued above that $x_1 = y_1 \succ \max_{R'}[x] = y$. Hence ψ_1 is ordered strictly below $x_1 = y_1$.
- We have established that ψ_1 is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to $(x = y \wedge \phi_{11}) \vee \mathbf{F}$ in context Γ'. Also ψ_2 is ordered strictly below $x_1 = y_1$, reduced in context Γ, and equivalent to $(x = y \wedge \phi_{12}) \vee \phi_2$ in context Γ. Since ϕ_1 is assumed reduced in context Γ, it follows that x_1 and y_1 are the greatest elements in their respective R_Γ-equivalence classes. The preconditions for the call to Build are therefore satisfied, and it follows that $\phi = $ Build($x_1 = y_1, \psi_1, \psi_2$) is ordered strictly below $x_1 = y_1$, reduced in context Γ, and equivalent to $(x_1 = y_1 \wedge \psi_1) \vee \psi_2$ in context Γ. Since ψ_1 is equivalent to $(x = y \wedge \phi_{11}) \vee \mathbf{F}$ in context Γ', and ψ_2 is equivalent to $(x = y \wedge \phi_{12}) \vee \phi_2$ in context Γ, it follows by Lemmas 18 and 19 that ϕ is equivalent in context Γ to $(x_1 = y_1 \wedge ((x = y \wedge \phi_{11}) \vee \mathbf{F})) \vee$

$((x = y \wedge \phi_{12}) \vee \phi_2)$; hence by Lemma 15 to $(x = y \wedge \phi_1) \vee \phi_2$. Moreover, suppose $x' = y'$ is any equation such that $x' = y' \succ x = y$ and such that ϕ_1 and ϕ_2 are ordered strictly below $x' = y'$. Then $x' = y' \succ x_1 = y_1$, and since ϕ is ordered strictly below $x_1 = y_1$, it is also ordered strictly below $x' = y'$.

This completes the induction step and the proof. □

Theorem 3 (Correctness of Reduce). *If ϕ is a decision form and Γ is a context,* Reduce(ϕ, Γ) *terminates and returns a decision form ϕ' that is ordered, reduced in context Γ and equivalent to ϕ in context Γ.*

Proof. Since Or, Degenerate, and Reorder have already been shown to terminate, and each recursive call within Reduce is made on a proper subformula of ϕ, the termination of Reduce is clear.

The proof of partial correctness of Reduce is by induction on the depth of recursive calls. We first consider the case in which there are no recursive calls. If Degenerate(Γ) holds, then $\phi' = \mathbf{F}$ and clearly ϕ' is equivalent in context Γ to ϕ. Since it is also clear that $\mathbf{F}$ is ordered and reduced in context Γ, the returned formula ϕ' has the required properties in this case. If Degenerate(Γ) does not hold, $\phi = \mathbf{T}$ or $\phi = \mathbf{F}$, and $\phi' = \phi$, then it is clear that ϕ' is ordered, reduced in context Γ, and equivalent to ϕ in context Γ. (Note that if Degenerate(Γ) *did* hold, then in case $\phi = \mathbf{T}$ it would not be the case that ϕ is reduced in context Γ; this is why the Degenerate check is required first.)

If there is a recursive call, then we are in the main body of Reduce and $\phi = (x = y \wedge \phi_1) \vee \phi_2$. In this case, by induction $\psi_2 =$ Reduce(ϕ_2, Γ) is ordered, reduced in context Γ and equivalent to ϕ_2 in context Γ. The variables x' and y' are, by definition, the maximum elements of their respective R_Γ-equivalence classes, and are such that $R_\Gamma + \{x' = y'\} = R_\Gamma + \{x = y\}$. We consider separately the two cases represented in the main body of Reduce.

1. If $x' = y'$, then x and y are in the same R_Γ-equivalence class, and ϕ is equivalent in context Γ to $(\mathbf{T} \wedge \phi_1) \vee \phi_2$; that is, to $\phi_1 \vee \phi_2$. Now, by induction Reduce(ϕ_1, Γ) returns ψ_1 which is ordered, reduced in context Γ, and equivalent to ϕ_1 in context Γ. Hence $\phi' =$ Or(ψ_1, ψ_2, Γ) is ordered, reduced in context Γ, and equivalent to $\psi_1 \vee \psi_2$ in context Γ. Since ψ_1 is equivalent to ϕ_1 in context Γ and ψ_2 is equivalent to ϕ_2 in context Γ, it follows by Corollary 2 that $\psi_1 \vee \psi_2$ is equivalent to $\phi_1 \vee \phi_2$ in context Γ, which is equivalent to ϕ in context Γ. Thus ϕ' is equivalent to ϕ in context Γ.
2. Otherwise, by induction $\psi_1 =$ Reduce$(\phi_1, \Gamma[(x' = y' \wedge [\,]) \vee \psi_2])$ is ordered, reduced in context $\Gamma' = \Gamma[(x' = y' \wedge [\,]) \vee \psi_2]$, and equivalent to ϕ_1 in context Γ'. Moreover, x' and y' are the greatest elements of their respective R_Γ-equivalence classes and Γ is nondegenerate. Then $\phi' =$ Reorder$(x' = y', \psi_1, \psi_2, \Gamma)$ is ordered, reduced in context Γ, and equivalent to $(x' = y' \wedge \psi_1) \vee \psi_2$ in context Γ. Since $R_\Gamma + \{x' = y'\} = R_\Gamma + \{x = y\}$, context $\Gamma[(x = y \wedge [\,]) \vee \psi_2]$ is equivalent to Γ'. Since ψ_1 is equivalent to ϕ_1 in

context Γ', it follows that $\Gamma'\{\psi_1\}$ is equivalent to $\Gamma'\{\phi_1\}$. Since $\Gamma[(x = y \wedge [\,]) \vee \psi_2]$ is equivalent to Γ', we have that $(x = y \wedge \psi_1) \vee \psi_2$ is equivalent to $(x = y \wedge \phi_1) \vee \psi_2$, hence ψ_1 is equivalent to ϕ_1 in context $\Gamma[(x = y \wedge [\,]) \vee \psi_2]$ Moreover, since ψ_2 is equivalent to ϕ_2 in context Γ, it follows by Lemmas 18 and 19 that ϕ' is equivalent in context Γ to $(x = y \wedge \phi_1) \vee \phi_2$.

□

7 Operations on Normal Forms

Note that once we have REDUCE, any algorithm that produces a decision form as a result can be modified to produce a normal form simply by applying REDUCE as the last step. However, it is possible to adapt the ideas underlying REDUCE to code algorithms for some common operations in such a way that they preserve normal forms without the reordering performed by a full application of REDUCE. In particular, in this section we present an algorithm for conjunction of normal forms (note that disjunction is directly implemented by function OR presented in the previous section), an algorithm for permuting variables in a normal form, algorithms for existentially or universally quantifying a variable in a normal form, and an algorithm to determine whether a formula implies another.

Function AND (shown in Figure 10) takes as arguments two decision forms ϕ_1 and ϕ_2 and a nondegenerate context Γ, where ϕ_1 and ϕ_2 are assumed to be ordered and reduced in the context Γ. It returns a decision form ϕ, which is ordered and reduced in the context Γ, and which is equivalent to $\phi_1 \wedge \phi_2$ in context Γ.

Lemma 32 (Correctness of AND). *Suppose decision forms ϕ_1 and ϕ_2 are ordered and reduced in the nondegenerate context Γ. Then* AND(ϕ_1, ϕ_2, Γ) *terminates and returns a decision form ϕ that is ordered, reduced in context Γ, and equivalent to $\phi_1 \wedge \phi_2$ in context Γ. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then ϕ is as well.*

Proof. Since BUILD, FILTER and OR have already been shown to terminate, the only potential sources of nontermination are the recursive calls from AND to itself. Define the *depth* of a decision form recursively as follows: the depth of $\mathbf{T}$ and $\mathbf{F}$ is zero, and the depth of $(x = y \wedge \phi_1) \vee \phi_2$ is the maximum of the depths of ϕ_1 and ϕ_2. It is easy to verify by inspection of the code for FILTER that the depth of the formula ϕ' returned by a call of FILTER(ϕ, Γ) is no greater than that of the argument formula ϕ. Therefore, the sum of the depths of the arguments to each recursive call to AND is strictly less than the sum of the depths of the arguments to the main call, and termination of AND is thus guaranteed.

The proof of partial correctness of AND is by induction on the depth of recursive calls. We first consider the case in which there are no recursive calls. In these cases, either one of ϕ_1 or ϕ_2 is $\mathbf{F}$, or ϕ_1 is $\mathbf{T}$, or ϕ_2 is $\mathbf{T}$. If one of ϕ_1 or ϕ_2 is $\mathbf{F}$, then the formula ϕ' returned by AND(ϕ_1, ϕ_2, Γ) is $\mathbf{F}$, which is clearly ordered strictly below an arbitrary equation $x = y$, reduced in any context, and is (unconditionally) equivalent to $\phi_1 \wedge \phi_2$, hence is also equivalent to $\phi_1 \wedge \phi_2$ in context Γ.

If ϕ_1 is $\mathbf{T}$, then ϕ' is ϕ_2, which is then clearly equivalent to $\phi_1 \wedge \phi_2$ in context Γ. By hypothesis, ϕ' is ordered and reduced in context Γ. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then clearly ϕ' is as well. If ϕ_2 is $\mathbf{T}$, then symmetric reasoning applies.

Otherwise, we are in the main body of And, $\phi_1 = (x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$ and $\phi_2 = (x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$. By hypothesis, Γ is nondegenerate. Moreover, from (R3), x_1, y_1, x_2, and y_2 are each the $\succ$-maximum elements of their respective R_Γ-equivalence classes. There are now three cases:

1. $(x_1, y_1) = (x_2, y_2)$. In this case, $\phi_1 \wedge \phi_2$ is (unconditionally) equivalent to $(x_1 = y_1 \wedge ((\phi_{11} \wedge \phi_{21}) \vee (\phi_{11} \wedge \phi_{22}) \vee (\phi_{12} \wedge \phi_{21}))) \vee (\phi_{12} \wedge \phi_{22})$, hence these formulas are also equivalent in context Γ. Since ϕ_1 and ϕ_2 are assumed ordered and reduced in context Γ, it follows that ϕ_{12} is ordered strictly below $x_1 = y_1$, ϕ_{22} is ordered strictly below $x_2 = y_2$ (which is the same as $x_1 = y_1$), and both are reduced in context Γ. Thus, by induction, $\psi_2 = \text{And}(\phi_{12}, \phi_{22}, \Gamma)$ is ordered strictly below $x_1 = y_1$, reduced in context Γ, and equivalent to $\phi_{12} \wedge \phi_{22}$ in context Γ. Let $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$. By Lemma 28, $\phi'_{12} = \text{Filter}(\phi_{12}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to ϕ_{12} in context Γ'. Similarly, $\phi'_{22} = \text{Filter}(\phi_{22}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to ϕ_{22} in context Γ'. Since ϕ_2 is assumed ordered and reduced in context Γ, it follows that ϕ_{21} is ordered strictly below $x_2 = y_2$ (which is the same as $x_1 = y_1$) and reduced in context $\Gamma[(x_2 = y_2 \wedge [\,]) \vee \phi_{22}]$ (which is the same as $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \phi_{22}]$). Additionally, ψ_2 is equivalent to $\phi_{12} \wedge \phi_{22}$ in context Γ; that is, ψ_2 implies ϕ_{22} in context Γ. Hence, by Lemma 24, Γ' entails $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \phi_{22}]$, and then by Lemma 26, ϕ_{21} is also reduced in context Γ'. Similarly, since ϕ_1 is assumed ordered and reduced in context Γ, it follows that ϕ_{11} is ordered strictly below $x_1 = y_1$ and reduced in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \phi_{12}]$. Additionally, ψ_2 is equivalent to $\phi_{12} \wedge \phi_{22}$ in context Γ; that is, ψ_2 implies ϕ_{12} in context Γ. Hence, by Lemma 24, Γ' entails $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \phi_{12}]$, and then by Lemma 26, ϕ_{11} is also reduced in context Γ'. It then follows by induction that:
   (a) $\psi_3 = \text{And}(\phi_{11}, \phi_{21}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to $\phi_{21} \wedge \phi_{21}$ in context Γ'.
   (b) $\psi_4 = \text{And}(\phi_{11}, \phi'_{22}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to $\phi_{11} \wedge \phi'_{22}$ in context Γ'. Thus by Corollary 2, ψ_4 is equivalent to $\phi_{11} \wedge \phi_{22}$ in context Γ' since ϕ'_{22} is equivalent to ϕ_{22} in context Γ'.
   (c) $\psi_5 = \text{And}(\phi'_{12}, \phi_{21}, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to $\phi'_{12} \wedge \phi_{21}$ in context Γ'. Thus by Corollary 2, ψ_5 is equivalent to $\phi_{12} \wedge \phi_{21}$ in context Γ' since ϕ'_{12} is equivalent to ϕ_{12} in context Γ'.

   Therefore, by Lemma 29, $\psi_1 = \text{Or}(\text{Or}(\psi_3, \psi_4, \Gamma'), \psi_5, \Gamma')$ is ordered strictly below $x_1 = y_1$, reduced in context Γ', and equivalent to $\psi_3 \vee \psi_4 \vee \psi_5$ in context

Γ'. Thus, by Corollary 2, ψ_1 is equivalent to $(\phi_{11} \wedge \phi_{21}) \vee (\phi_{11} \wedge \phi_{22}) \vee (\phi_{12} \wedge \phi_{21})$ in context Γ'. The preconditions for BUILD are therefore satisfied, and we have that $\phi' = \textsc{Build}(x_1 = y_1, \psi_1, \psi_2)$ is ordered below $x_1 = y_1$, reduced in context Γ, and equivalent to $\psi_1 \wedge \psi_2$ in context Γ. Since ψ_1 is equivalent to $(\phi_{11} \wedge \phi_{21}) \vee (\phi_{11} \wedge \phi_{22}) \vee (\phi_{12} \wedge \phi_{21})$ in context Γ', and ψ_2 is equivalent to $\phi_{12} \wedge \phi_{22}$ in context Γ, it follows by Lemmas 18 and 19 that ϕ is equivalent in context Γ to $(x_1 = y_1 \wedge ((\phi_{11} \wedge \phi_{21}) \vee (\phi_{11} \wedge \phi_{22}) \vee (\phi_{12} \wedge \phi_{21}))) \vee (\phi_{12} \wedge \phi_{22})$; hence is also equivalent to $\phi_1 \wedge \phi_2$ in context Γ. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then $x = y \succeq x_1 = y_1$ (resp. $x = y \succ x_1 = y_1$), from which it follows that ϕ' is ordered below (resp. strictly below) $x = y$ as well.

2. $(x_1, y_1) \succ (x_2, y_2)$. In this case, $\phi_1 \wedge \phi_2$ is (unconditionally) equivalent to $(x_1 = y_1 \wedge (\phi_{11} \wedge \phi_2)) \vee (\phi_{12} \wedge \phi_2)$, hence these formulas are also equivalent in context Γ. Since ϕ_1 and ϕ_2 are assumed ordered and reduced in context Γ and $(x_1, y_1) \succ (x_2, y_2)$, it follows that ϕ_{12} and ϕ_2 are both ordered strictly below $x_1 = y_1$, and both are reduced in context Γ. Thus, by induction, $\psi_2 = \textsc{And}(\phi_{12}, \phi_2, \Gamma)$ is ordered strictly below $x_1 = y_1$, reduced in context Γ, and equivalent to $\phi_{12} \wedge \phi_2$ in context Γ. Also, since ϕ_1 is assumed ordered and reduced in context Γ, it follows that ϕ_{11} is ordered strictly below $x_1 = y_1$ and reduced in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \phi_{12}]$. Since ψ_2 is equivalent to $\phi_{12} \wedge \phi_2$ in context Γ, it follows that ψ_2 implies ϕ_{12} in context Γ. Hence, by Lemma 24, $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$ entails $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \phi_{12}]$, and then by Lemma 26, ϕ_{11} is also reduced in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$. By Lemma 28, $\phi_2' = \textsc{Filter}(\phi_2, \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2])$ is ordered strictly below $x_1 = y_1$, reduced in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$, and equivalent to ϕ_2 in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$. It then follows by induction that $\psi_1 = \textsc{And}(\phi_{11}, \phi_2', \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2])$ is ordered strictly below $x_1 = y_1$, reduced in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$, and equivalent to $\phi_{11} \wedge \phi_2'$ in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$. Thus by Corollary 2, ψ_1 is equivalent to $\phi_{11} \wedge \phi_2$ in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$ since ϕ_2' is equivalent to ϕ_2 in context $\Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$. The preconditions for BUILD are therefore satisfied, and we have that $\phi' = \textsc{Build}(x_1 = y_1, \psi_1, \psi_2)$ is ordered below $x_1 = y_1$, reduced in context Γ, and equivalent to $\psi_1 \wedge \psi_2$ in context Γ. Since ψ_1 is equivalent to $\phi_{11} \wedge \phi_2$ in context Γ', and ψ_2 is equivalent to $\phi_{12} \wedge \phi_2$ in context Γ, it follows by Lemmas 18 and 19 that ϕ is equivalent in context Γ to $(x_1 = y_1 \wedge (\phi_{11} \wedge \phi_2)) \vee (\phi_{12} \wedge \phi_2)$; hence is also equivalent to $\phi_1 \wedge \phi_2$ in context Γ. Moreover, if ϕ_1 and ϕ_2 are both ordered below (resp. strictly below) some equation $x = y$, then $x = y \succeq x_1 = y_1$ (resp. $x = y \succ x_1 = y_1$), from which it follows that ϕ' is ordered below (resp. strictly below) $x = y$ as well.
3. $(x_2, y_2) \succ (x_1, y_1)$. This case is symmetric to case (2).

□

We now consider existential quantification of a variable in a PEF. Formally, if ϕ is a PEF and z is a variable, then a formula ψ is called an *existential* *quantification of ϕ with respect to z* if and only if the following condition is satisfied:

– For all equivalence relations R, we have $R \models \psi$ if and only if there exists an equivalence relation R' such that $R' \models \phi$ and R' agrees with R on all variables other than z.

This condition uniquely determines ψ up to logical equivalence, if in fact some such formula ψ exists.

Lemma 33 below shows that existential quantifications always exist. For its formal statement, we introduce a notation for substitution of variables: If ϕ is a PEF and u and v are distinct variables, then $\phi[u/v]$ denotes the result of replacing each occurrence of the variable v by the variable u, making those adjustments necessary to ensure that each equation in the resulting formula is oriented. Specifically, each equation $u = v$ or $v = u$ is replaced by $\mathbf{T}$ and other equations $v = u'$ or $u' = v$ become either $u = u'$ or $u' = u$, depending on whether $u \succ u'$ or $u' \succ u$.

Lemma 33. *Suppose ϕ is a formula, z is a variable, and $\{x_1, x_2, \ldots, x_n\}$ are all the variables other than z that appear in ϕ. If $n = 0$ then ϕ itself is an existential quantification of ϕ with respect to z. If $n > 0$ then the formula*

$$\phi[x_1/z] \vee \phi[x_2/z] \vee \ldots \vee \phi[x_n/z]$$

is an existential quantification of ϕ with respect to z.

Proof. First consider the case $n = 0$. Then ϕ contains no variables other than z. In this case, it is easily verified that either ϕ is equivalent to $\mathbf{T}$ or ϕ is equivalent to $\mathbf{F}$. In either case, given an equivalence relation R, if $R \models \phi$ then in fact $\phi = \mathbf{T}$, hence taking $R' = R$ we obtain an equivalence relation R' that agrees with R on all variables other than z and is such that $R' \models \phi$. Conversely, if $R' \models \phi$ holds for some equivalence relation R' that agrees with R on all variables other than z, then $\phi = \mathbf{T}$, hence $R \models \phi$ holds as well. Thus, ϕ itself is an existential quantification of ϕ with respect to z.

Now suppose $n > 0$. Let ψ be the formula

$$\phi[x_1/z] \vee \phi[x_2/z] \vee \ldots \vee \phi[x_n/z].$$

We show that ψ satisfies the conditions required of an existential quantification of ϕ with respect to z. Suppose $R \models \psi$. Then $R \models \phi[x_i/z]$ for some i. Since $\phi[x_i/z]$ has no occurrences of z, it is also the case that $R \setminus z \models \phi[x_i/z]$. Let $R' = (R \setminus z) + \{z = x_i\}$; then $R' \models \phi$ holds and R' agrees with R on all variables other than z.

Conversely, let R be an equivalence relation and suppose $R' \models \phi$ for some equivalence relation R' that agrees with R on all variables but z. Then either z is in the R'-equivalence class of some x_i, or it is not in the R'-equivalence class of any x_i. If z is in the R'-equivalence class of x_i, then since $R' \models \phi$ it follows that $R \models \phi[x_i/z]$, hence $R \models \psi$. Suppose z is not in the R'-equivalence class of any x_i. In this case, R' does not satisfy any oriented equation involving z, and since R' agrees with R on all variables other than z, it follows that $R \models \phi$, hence $R \models \phi[x_i/z]$ for all i. Since $n > 0$, in fact we have $R \models \phi[x_i/z]$ for some i. Thus, $R \models \psi$. $\square$

We will write $\exists z.\phi$ to denote the unique normal form ψ that is an existential quantification of ϕ with respect to z.

Existential quantification of normal forms can be implemented by function Exists (shown in Figure 11). This function takes as arguments a variable z and a normal form ϕ such that z is the least variable (with respect to the ordering $\succ$) in ϕ, and returns $\exists z.\phi$. Function Exists is basically computing the disjunction of $\phi[x_1/z] \vee \phi[x_2/z] \vee \ldots \vee \phi[x_n/z]$ for all the variables $\{x_1, x_2, ..., x_n\}$ other than z that occur in ϕ, while Reduce($\phi, [(x_i = z \wedge [\,]) \vee \mathbf{F}]$) computes $\phi[x_i/z]$ for each x_i in $\{x_1, x_2, ..., x_n\}$.

Later we will show that the assumption that z is the $\succ$-least variable in ϕ is not restrictive, since we can always arrange (via function Rename shown in Figure 14) for z to be replaced by a variable z' that is smaller than all the variables occurring in ϕ.

Lemma 34. *Suppose ϕ is a PEF and v and z are variables with $v \succ z$. If R is an equivalence relation, then $R \models \phi[v/z]$ if and only if $(R \setminus z) + \{v = z\} \models \phi$ (resp. $(R \setminus z) + \{z = v\} \models \phi$).*

Proof. By induction on ϕ. If $\phi = \mathbf{T}$ then clearly we have $R \models \phi[v/z]$ and $(R \setminus z) + \{v = z\} \models \phi$ for any equivalence relation R. If $\phi = \mathbf{F}$ then no equivalence relation satisfies $\phi[v/z]$ and the lemma holds vacuously.

Next, suppose ϕ is a single equation $x = y$. There are three cases:

1. $x = y$ does not involve z. Then $(x = y)[v/z]$ is $x = y$. If $R \models (x = y)[v/z]$, then $R \models x = y$. But $x = y$ does not involve z, hence $R \setminus z \models x = y$, from which it follows by monotonicity that $(R \setminus z) + \{v = z\} \models x = y$. Conversely, suppose $(R \setminus z) + \{v = z\} \models x = y$. Then $R \setminus z$ contains no equations involving z, so $(R \setminus z) + \{v = z\}$ agrees with $R \setminus z$, hence also with R, on equations that do not involve z. Since $(R \setminus z) + \{v = z\} \models x = y$ and equation $x = y$ does not involve z, it follows that $R \models x = y$ and also $R \models (x = y)[v/z]$.
2. x is z. Then $(x = y)[v/z]$ is either $v = y$ or $y = v$, depending on which of the two is oriented. If $R \models (x = y)[v/z]$, then R satisfies either $v = y$ or $y = v$. But neither $v = y$ nor $y = v$ involves z, hence $R \setminus z$ satisfies either $v = y$ or $y = v$. Thus, either $(R \setminus z) + \{v = z\} \models z = y$ or $(R \setminus z) + \{v = z\} \models y = z$. But since x is z, we in fact have $(R \setminus z) + \{v = z\} \models x = y$. Conversely, suppose $(R \setminus z) + \{v = z\} \models x = y$. Since x is z, it follows that $(R \setminus z) + \{v = z\}$ equals $(R \setminus z) + \{v = x\}$, which therefore satisfies either $v = y$ or $y = v$. But neither $v = y$ nor $y = v$ involves z, hence $((R \setminus z) + \{v = z\}) \setminus z$ (that is, $R \setminus z$) satisfies either $v = y$ or $y = v$. By monotonicity, R satisfies either $v = y$ or $y = v$; that is, $R \models (x = y)[v/z]$.
3. y is z. This case is symmetric to case (2).

Now, suppose ϕ is $\phi_1 \wedge \phi_2$. Then $R \models \phi[v/z]$ if and only if both $R \models \phi_1[v/z]$ and $R \models \phi_2[v/z]$. By induction, $R \models \phi_1[v/z]$ if and only if $(R \setminus z) + \{v = z\} \models \phi_1$, and $R \models \phi_2[v/z]$ if and only if $(R \setminus z) + \{v = z\} \models \phi_2$. Thus, $R \models \phi[v/z]$ if and only if both $(R \setminus z) + \{v = z\} \models \phi_1$ and $(R \setminus z) + \{v = z\} \models \phi_2$; that is, if and only if $(R \setminus z) + \{v = z\} \models \phi_1 \wedge \phi_2$.

Finally, suppose ϕ is $\phi_1 \vee \phi_2$. Then $R \models \phi[v/z]$ if and only if either $R \models \phi_1[v/z]$ or $R \models \phi_2[v/z]$. By induction, $R \models \phi_1[v/z]$ if and only if $(R \setminus z) + \{v = z\} \models \phi_1$, and $R \models \phi_2[v/z]$ if and only if $(R \setminus z) + \{v = z\} \models \phi_2$. Thus, $R \models \phi[v/z]$ if and only if either $(R \setminus z) + \{v = z\} \models \phi_1$ or $(R \setminus z) + \{v = z\} \models \phi_2$; that is, if and only if $(R \setminus z) + \{v = z\} \models \phi_1 \vee \phi_2$. □

Lemma 35. *Suppose ϕ is a PEF and suppose $V = \{v_1, v_2, \ldots, v_n\}$ and $Z = \{z_1, z_2, \ldots, z_n\}$ are sets of variables, with $v_i \succ z_j$ for $1 \leq i \leq n$ and $1 \leq j \leq n$. If R is an equivalence relation, then $R \models \phi[v_1/z_1, v_2/z_2, \ldots, v_n/z_n]$ if and only if*

$$(R \setminus \{z_1, z_2, \ldots, z_n\}) + \{v_1 = z_1, v_2 = z_2, \ldots, v_n = z_n\} \models \phi.$$

Proof. Since $v_i \succ z_j$ for $1 \leq i \leq n$ and $1 \leq j \leq n$, the sets V and Z are disjoint. The simultaneous substitution $\phi[v_1/z_1, v_2/z_2, \ldots, v_n/z_n]$ is therefore the same as the iterated substitution $\phi[v_1/z_1, v_2/z_2, \ldots, v_{n-1}/z_{n-1}][v_n/z_n]$ and the relation $(R \setminus \{z_1, z_2, \ldots, z_n\}) + \{v_1 = z_1, v_2 = z_2, \ldots, v_n = z_n\}$ is the same as $(((R \setminus \{z_1, z_2, \ldots, z_{n-1}\}) + \{v_1 = z_1, v_2 = z_2, \ldots, v_{n-1} = z_{n-1}\}) \setminus z_n) + \{v_n = z_n\}$. The result can now be established by an induction using Lemma 34 and these recursive relationships. □

Lemma 36 (Correctness of Exists). *Suppose z is the least variable (with respect to the ordering $\succ$) in a normal form ϕ. Then* Exists(z, ϕ) *terminates and returns $\exists z.\phi$.*

Proof. Since Reduce and Or have already been shown to terminate, and each recursive call to the auxiliary function Ex is on a smaller set of variables V than the main call, the termination of Exists is clear.

If $\phi = \mathbf{T}$ then $\varphi = \phi$. Clearly, by Lemma 33, φ is is an existential quantification of ϕ with respect to z, and since ϕ is ordered and reduced in context [ ], it is a normal form, hence is $\exists z.\phi$. If $\phi \neq \mathbf{T}$ then $\varphi =$ Ex$(\mathbf{F}, V)$, where V is the set of all variables other than z that occur in ϕ. Since z is the least variable in ϕ, each equation $v = z$ with $v \in V$ is oriented, hence each call of Reduce$(\phi, [(v = z \wedge [\,]) \vee \mathbf{F}])$ returns a formula ψ' that is ordered and reduced in context $[(v = z \wedge [\,]) \vee \mathbf{F}]$, and equivalent to ϕ in context $[(v = z \wedge [\,]) \vee \mathbf{F}]$. It follows that ψ' is a normal form in which the variable z does not occur, such that $v = z \wedge \psi'$ is equivalent to $v = z \wedge \phi$. Then given an arbitrary equivalence relation R, we have $R \models \psi'$ iff $R \setminus z \models \psi'$ iff $(R \setminus z) + \{v = z\} \models v = z \wedge \psi'$ iff $(R \setminus z) + \{v = z\} \models v = z \wedge \phi$ iff $(R \setminus z) + \{v = z\} \models \phi$. By Lemma 34 the last assertion holds if and only if $R \models \phi[v/z]$. Thus $R \models \psi'$ if and only $R \models \phi[v/z]$; that is, ψ' is logically equivalent to $\phi[v/z]$.

It then follows that Ex$(\mathbf{F}, V)$ returns a normal form φ that is equivalent to $\phi[x_1/z] \vee \phi_[x_2/z] \vee \ldots \vee \phi[x_n/z]$, where $\{x_1, x_2, ..., x_n\} = V$. Thus, by Lemma 33, φ is $\exists z.\phi$. □

We now consider universal quantification of a variable in a PEF. Formally, if ϕ is a PEF and z is a variable, then a formula ψ is called a *universal quantification of ϕ with respect to z* if and only if the following condition is satisfied:

– For all equivalence relations R, we have $R \models \psi$ if and only if $R' \models \phi$ holds for all equivalence relations R' that agree with R on all variables but z.

This condition uniquely determines ψ up to logical equivalence, and Lemma 37 shows that such a formula ψ always exists.

Lemma 37. *Suppose ϕ is a normal form and z is a variable. Let ψ be obtained from ϕ by replacing each equation involving z by $\mathbf{F}$. Then ψ is a universal quantification of ϕ with respect to z.*

Proof. Suppose $R \models \psi$, and let R' be an equivalence relation that agrees with R on all variables but z. Since ψ contains no occurrences of z, and R' agrees with R on all variables but z, it follows that $R' \models \psi$. Since ψ is obtained from ϕ by replacing each equation involving z by $\mathbf{F}$, it follows that ψ implies ϕ. Thus, $R' \models \phi$.

Conversely, suppose $R' \models \phi$ for all equivalence relations R' that agree with R on all variables but z. Then in particular $R \setminus z \models \phi$. But $R \setminus z$ does not satisfy any equation involving z (recall that equations are oriented, hence each equation relates two distinct variables) and it agrees with R on all other equations, hence it must also be the case that $R \setminus z \models \psi$. But then $R \models \psi$ holds by monotonicity. □

We will write $\forall z.\phi$ to denote the unique normal form ψ that is a universal quantification of ϕ with respect to z.

Lemma 38. *Suppose ϕ is a normal form and z is a variable. Then $\forall z.\phi$ contains no occurrences of z and no variables that are not in ϕ.*

Proof. By induction on ϕ. If ϕ is $\mathbf{T}$ or $\mathbf{F}$, then $\forall z.\phi$ is ϕ, and the result clearly holds. If ϕ is $(z = y \wedge \phi_1) \vee \phi_2$ or $(x = z \wedge \phi_1) \vee \phi_2$, then $\forall z.\phi$ is equivalent to the normal form $\forall z.\phi_2$, hence equal to it, and by induction the result holds in this case as well. If ϕ is $(x = y \wedge \phi_1) \vee \phi_2$, where neither x nor y is z, then $\forall z.\phi$ is equivalent to $(x = y \wedge (\forall z.\phi_1)) \vee (\forall z.\phi_2)$. By induction, $\forall z.\phi_1$ contains no occurrences of z and no variables that are not in ϕ_1, and $\forall z.\phi_2$ contains no occurrences of z and no variables that are not in ϕ_2. Since ϕ_1 and ϕ_2 are ordered strictly below $x = y$, so are $\forall z.\phi_1$ and $\forall z.\phi_2$. If $\forall z.\phi_1$ is $\mathbf{F}$, then $\forall z.\phi$ is equivalent to $\forall z.\phi_2$, hence equal to it. If $\forall z.\phi_2$ is $\mathbf{T}$, then $\forall z.\phi$ is equivalent to $\mathbf{T}$, hence equal to it. Otherwise $\forall z.\phi$ is equivalent to the normal form $(x = y \wedge (\forall z.\phi_1)) \vee (\forall z.\phi_2)$, hence equal to it. In all these cases, $\forall z.\phi$ contains no occurrences of z and no variables that are not in ϕ. □

Universal quantification of normal forms can be implemented by function Forall (shown in Figure 12). This function takes as arguments a variable z and a normal form ϕ, and returns $\forall z.\phi$.

Lemma 39 (Correctness of $\textsc{Forall}$**).** *Suppose* z *is a variable and* ϕ *is a normal form. Then* $\textsc{Forall}(z, \phi)$ *terminates and returns* $\forall z.\phi$*.*

Proof. Since $\textsc{Build}$ always terminates and each recursive call within $\textsc{Forall}$ is made on a proper subformula of ϕ, the termination of $\textsc{Forall}$ is clear.

The proof of partial correctness of $\textsc{Forall}$ is by induction on the depth of recursive calls. We first consider the case in which there are no recursive calls. In these cases, ϕ is either $\mathbf{T}$ or $\mathbf{F}$, and ϕ is the formula returned. Clearly then, by Lemma 37, ϕ is $\forall z.\phi$.

Otherwise, we are in the main body of $\textsc{Forall}$, and ϕ is $(x = y \wedge \phi_1) \vee \phi_2$. There are now two cases:

1. $x = y$ involves z. The formula returned in this case is $\phi' = \textsc{Forall}(z, \phi_2)$, which by induction is $\forall z.\phi_2$. But in this case, $\forall z.\phi$ is equivalent to $(\mathbf{F} \wedge \forall z.\phi_1) \vee \forall z.\phi_2$, hence to $\forall z.\phi_2$, which is a normal form. Thus, $\forall z.\phi = \forall z.\phi_2$.
2. $x = y$ does not involve z. By induction, $\phi'_1 = \textsc{Forall}(z, \phi_1)$ is $\forall z.\phi_1$ and $\phi'_2 = \textsc{Forall}(z, \phi_2)$ is $\forall z.\phi_2$. Since ϕ is a normal form, by conditions $(O1)$ and $(O2)$, ϕ_1 and ϕ_2 are both ordered strictly below $x = y$. By Lemma 38, $\forall z.\phi_1$ contains no occurrences of z and no variables that are not in ϕ_1. Thus, $\forall z.\phi_1$ is also ordered strictly below $x = y$. Similarly, $\forall z.\phi_2$ is ordered strictly below $x = y$. The preconditions of $\textsc{Build}$ are therefore satisfied, and if $\phi' = \textsc{Build}(x = y, \phi'_1, \phi'_2)$, then ϕ' is equivalent to $(x = y \wedge (\forall z.\phi_1)) \vee (\forall z.\phi_2)$, which is a normal form. By Lemma 37, ϕ' is also equivalent to $\forall z.\phi$. Hence $\phi' = \forall z.\phi$. But ϕ' is the formula returned by $\textsc{Forall}(z, \phi)$ in this case, completing the proof.

□

It is useful to determine whether a formula ϕ implies another formula ψ. Formally, formula ϕ *implies* another formula ψ if every equivalence relation R that satisfies ϕ also satisfies ψ. It is easy to check that this definition is equivalent to the condition that every implicant of ϕ satisfies ψ. Thus, determining whether ϕ implies ψ reduces to the problem of determining whether every implicant of ϕ satisfies ψ. But the statement that every implicant of ϕ satisfies ψ is equivalent to the statement that every implicant of ϕ is subsumed by a suitable context. This observation leads to function $\textsc{Implies}$ shown in Figure 13, which takes as arguments a normal form ϕ and a normal form ψ, and returns *true* if and only if ϕ implies ψ.

Lemma 40 (Correctness of $\textsc{Implies}$**).** *Suppose* ϕ *and* ψ *are normal forms. Then* $\textsc{Implies}(\phi, \psi)$ *terminates and returns* true *if and only if* ϕ *implies* ψ*.*

Proof. Since $\textsc{Filter}$ has already been shown to terminate, the termination of $\textsc{Implies}$ is clear.

Let Γ be $[(x = y \wedge [\,]) \vee (x = y \wedge \psi) \vee \mathbf{F}]$, where x and y are variables such that $x \succ y$ and $y \succ v$ for all variables v appearing in either ϕ or ψ. Let $\varphi = \textsc{Filter}(\phi, \Gamma)$; then φ is ordered, reduced in context Γ, and equivalent to ϕ in context Γ. Now, φ equivalent to ϕ in context Γ means that $(x = y \wedge \varphi) \vee$

$((x = y \wedge \psi) \vee \mathbf{F})$ is equivalent to $(x = y \wedge \phi) \vee ((x = y \wedge \psi) \vee \mathbf{F})$, which in turn is equivalent to $x = y \wedge (\phi \vee \psi)$.

If $\varphi = \mathbf{F}$, then $(x = y \wedge \varphi) \vee ((x = y \wedge \psi) \vee \mathbf{F})$ is equivalent to $x = y \wedge \psi$. Since neither x nor y appear in either ϕ or ψ, from $x = y \wedge \psi$ equivalent to $x = y \wedge (\phi \vee \psi)$ we may conclude that ψ is equivalent to $\phi \vee \psi$. But ψ is equivalent to $\phi \vee \psi$ if and only if ϕ implies ψ.

On the other hand, if φ is not $\mathbf{F}$, then φ has some implicant R. Let $R' = R + \{x = y\}$, then R' is an implicant of $(x = y \wedge \varphi) \vee ((x = y \wedge \psi) \vee \mathbf{F})$, and since $(x = y \wedge \varphi) \vee ((x = y \wedge \psi) \vee \mathbf{F})$ is equivalent to $x = y \wedge (\phi \vee \psi)$, it follows that $R' \models x = y \wedge (\phi \vee \psi)$, hence $R' \models \phi \vee \psi$. Since φ is reduced in context Γ, no implicant of φ can satisfy ψ, hence in particular $R \not\models \psi$, thus also $R' \not\models \psi$ because x and y do not occur in ψ. But if $R' \models \phi \vee \psi$ and $R' \not\models \psi$, it must be that $R' \models \phi$. We have thus exhibited an equivalence relation R' such that $R' \models \phi$ but $R' \not\models \psi$, which shows that ϕ does not imply ψ. $\square$

A *renaming of variables* is a function $f : \mathcal{V} \to \mathcal{V}$ such that $x \neq f(x)$ for at most finitely many $x \in \mathcal{V}$. Function RENAME, shown in Figure 14, performs renaming of variables in normal forms, as expressed by Lemma 41 below. Function RENAME assumes that the set $\mathcal{V}$ of variables is partitioned into subsets of *unbarred* and *barred* variables, which are in one-to-one correspondence. That is, to each unbarred variable x, there corresponds a unique barred variable $\bar{x}$. Further, it is assumed that if x and y are unbarred variables such that $x \succ y$, then $x \succ y \succ \bar{x} \succ \bar{y}$.

Function RENAME makes use of a particular context $\Gamma_{f,V}$, associated with renaming of variables f and set of unbarred variables V. The context $\Gamma_{f,V}$ is defined recursively as follows:

- If $V = \emptyset$, then $\Gamma_{f,V} = [\,]$.
- If $V \neq \emptyset$, and v is the $\succ$-minimal element of V, then

$$\Gamma_{f,V} = \Gamma_{f,V\setminus\{v\}}[(f(v) = \bar{v} \wedge [\,]) \vee \mathbf{F}].$$

That is, the context $\Gamma_{f,V}$ is the context in which each barred variable $\bar{v}$ is equated to the variable $f(v)$.

Lemma 41 (Correctness of RENAME). *Suppose ϕ is a normal form that does not contain any barred variables and f is a renaming of variables such that $f^{-1}(\bar{v}) = \{\bar{v}\}$ for all unbarred variables v. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the variables occurring in ϕ. Then RENAME(f, ϕ) terminates and returns a normal form ϕ' that is equivalent to $\phi[f(v_1)/v_1, f(v_2)/v_2, \ldots, f(v_n)/v_n]$.*

Proof. The auxiliary function BAR used in RENAME takes an argument formula ϕ that contains no barred variables and makes a copy $\bar{\phi}$ of ϕ in which each unbarred variable has been replaced by the corresponding barred variable. Function BAR clearly terminates, and since the relative ordering of barred variables is the same as their unbarred counterparts, if the argument to BAR is a normal form, then the returned formula will be a normal form as well.

The termination of Rename is immediate from the fact that Bar and Reduce have already been shown to terminate. The formula ϕ' returned by Rename(ϕ, f) is given by Reduce$(\bar{\phi}, \mathit{\Gamma}_{f,V})$. By Theorem 3, ϕ' is ordered, reduced in context $\mathit{\Gamma}_{f,V}$, and is equivalent to $\bar{\phi}$ in context $\mathit{\Gamma}_{f,V}$. Since ϕ is reduced in context $\mathit{\Gamma}_{f,V}$, and $R_{\mathit{\Gamma}_{f,V}}$ contains an equation $f(v) = \bar{v}$ for each variable $v \in V$, it follows that ϕ' contains no occurrences of $\bar{v}$ for any variable v in V. But since V is the set of all variables occurring in ϕ, every variable occurring in $\bar{\phi}$ is $\bar{v}$ for some $v \in V$. Since Reduce$(\bar{\phi}, \mathit{\Gamma}_{f,V})$ does not contain any variables that do not already occur in $\bar{\phi}$ or $\mathit{\Gamma}_{f,V}$, it follows that ϕ' contains no barred variables.

We have thus shown that ϕ' is a normal form that contains no barred variables and is equivalent to $\bar{\phi}$ in context $\mathit{\Gamma}_{f,V}$. That is to say,

$$f(v_1) = \bar{v}_1 \wedge f(v_2) = \bar{v}_2 \wedge \ldots \wedge f(v_n) = \bar{v}_n \wedge \phi'$$

is equivalent to

$$f(v_1) = \bar{v}_1 \wedge f(v_2) = \bar{v}_2 \wedge \ldots \wedge f(v_n) = \bar{v}_n \wedge \bar{\phi}.$$

We claim that ϕ' is in fact equivalent to $\bar{\phi}[f(v_1)/\bar{v}_1, f(v_2)/\bar{v}_2, \ldots, f(v_n)/\bar{v}_n]$. Suppose R is any equivalence relation. Then since ϕ' contains no barred variables, $R \models \phi'$ holds if and only if $R \setminus \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n\} \models \phi'$ hence if and only if

$$(R \setminus \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n\}) + \{f(v_1) = \bar{v}_1, f(v_2) = \bar{v}_2, \ldots, f(v_n) = \bar{v}_n\}$$

satisfies

$$f(v_1) = \bar{v}_1 \wedge f(v_2) = \bar{v}_2 \wedge \ldots \wedge f(v_n) = \bar{v}_n \wedge \phi',$$

hence if and only if

$$(R \setminus \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n\}) + \{f(v_1) = \bar{v}_1, f(v_2) = \bar{v}_2, \ldots, f(v_n) = \bar{v}_n\}$$

satisfies

$$f(v_1) = \bar{v}_1 \wedge f(v_2) = \bar{v}_2 \wedge \ldots \wedge f(v_n) = \bar{v}_n \wedge \bar{\phi}.$$

By Lemma 35, this holds if and only if

$$R \models \bar{\phi}[f(v_1)/\bar{v}_1, f(v_2)/\bar{v}_2, \ldots, f(v_n)/\bar{v}_n].$$

Since all this is true for arbitrary R, it follows that ϕ' is in fact equivalent to $\bar{\phi}[f(v_1)/\bar{v}_1, f(v_2)/\bar{v}_2, \ldots, f(v_n)/\bar{v}_n]$. But $\bar{\phi}[f(v_1)/\bar{v}_1, f(v_2)/\bar{v}_2, \ldots, f(v_n)/\bar{v}_n]$ is the same as $\phi[f(v_1)/v_1, f(v_2)/v_2, \ldots, f(v_n)/v_n]$. Thus, ϕ' is equivalent to $\phi[f(v_1)/v_1, f(v_2)/v_2, \ldots, f(v_n)/v_n]$, completing the proof. □

8 Implementation

Extending a BDD package that we had previously built, we have coded a prototype implementation of our normal form algorithms under the Standard ML of

New Jersey (SML/NJ) language system. Below we discuss briefly several issues about the implementation.

In our implementation, as in other BDD implementations, we achieve maximally structure-shared storage of normal forms using a hash table, called the *node table*, to ensure that each normal form is represented by a unique node. Before adding a node into the node table for a normal form ϕ, we first check the table. If a node for ϕ already exists then we do nothing, otherwise we add a new entry for it. Therefore, each node in the node table represents a unique normal form. Additional hash tables, called *operation caches*, are used to avoid repeating the calculation of an operation on particular arguments after it has been done once. In order to avoid swamping the system through overly aggressive memory allocation, our implementation applies a heuristic that automatically monitors garbage-collection activity to determine when to adjust the maximal sizes of the node table and operation caches.

Equivalence relations are implemented as hash tables as well. A hash table that represents an equivalence relation R maps an element x to a *ref*-variable that refers to $\max_R[x]$. For the purpose of efficiency, we do not make x as a key if $\{x\}$ is a singleton set in R. The frequently used operations for equivalence relations are: (1) add an equation $x = y$ to an equivalence relation R; (2) check whether an equation $x = y$ is in equivalence relation R; (3) return $\max_R[x]$. We denote the hash table for equivalence relation R by *table*(R), denote the value (*ref*-variable) of a key x by *value*(x), and denote the element that is referred by a *ref*-variable v by *deref*(v). To add an equation $x = y$ to equivalence relation R, we consider the following cases:

1. Neither x nor y is a key in *table*(R). Then we add entries for x and y, and map both x and y to a new *ref*-variable v that refers to x.
2. x is a key in *table*(R) and y is not. Then we add an entry for y and map y to *value*(x).
3. y is a key in *table*(R) and x is not. Then we refer *value*(y) to x so that *deref*(*value*(y)) $= x$, add an entry for x, and map x to *value*(y).
4. x and y both are keys in *table*(R). Then we refer *value*(y) to *deref*(*value*(x)) so that *deref*(*value*(y)) $=$ *deref*(*value*(x)).

To determine whether an equation $x = y$ is in R, we first check *table*(R). If x and y are not both keys in *table*(R) then we return *false*, since in that case either $\{x\}$ or $\{y\}$ is a singleton set in R. If x and y are both keys in *table*(R) then there are two cases: (1) *value*(x) and *value*(y) refer to the same element; (2) *value*(x) and *value*(y) refer to different elements. We return *true* for case (1) since $\max_R[x] = \max_R[y]$ and return *false* for case (2). Note that x and y are in the same R-equivalence class if and only if $\max_R[x] = \max_R[y]$. To compute $\max_R[x]$, we first check *table*(R). If x is a key in the table then we return *deref*(*value*(x)), otherwise we return x directly. It is obvious that the above operations are all performed in constant time. Another important operation is to make a copy T of *table*(R). Since the values for the keys in *table*(R) are ref-variables, simply copying all entries in *table*(R) to T would cause interference between *table*(R) and T. The correct way to handle this is to create a new ref-variable v' for each ref-variable v in *table*(R) and refer v' to *deref*(v). For each key x in *table*(R) such that *value*$(x) = v$, we add an entry for x in T and map x to v'.

9 Conclusion

We have presented a normal form for positive equational formulas, such that logically equivalent normal forms are identical. Based on a characterization of normal forms as decision forms that are ordered and "reduced in context," we devised an algorithm for reducing an arbitrary decision form to normal form. The algorithm builds normal forms in a bottom-up fashion that is well-suited to BDD-based implementation. We have constructed such an implementation that includes, besides the reduction algorithm, additional algorithms for performing various logical operations on normal forms.

In developing the ideas presented in this paper, we have primarily focused on pure equational formulas. However, it now seems to us that the ideas will extend readily to formulas that include propositional variables as well as equations. Previous techniques that represent equational formulas as decision diagrams have made use of the Shannon expansion, which involves negation and is therefore not naturally applicable to positive formulas. Our notion of "decision form," which does not use negation, provides a different way to represent positive equational formulas as decision diagrams. An interesting question we are currently considering is whether negation can be "retro-fitted" into this apparently novel representation, thereby providing a canonical normal form for equational formulas with negation.

References

[BGV99] Randal E. Bryant, Steven M. German, and Miroslav N. Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 470–482, London, UK, 1999. Springer-Verlag.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[BV00] Randal E. Bryant and Miroslav N. Velev. Boolean satisfiability with transitivity constraints. In *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*, pages 85–98, London, UK, 2000. Springer-Verlag.

[BvdP05] Bahareh Badban and Jaco van de Pol. Zero, successor and equality in BDDs. *Annals of Pure and Applied Logic*, 133(1-3):101–123, 2005.

[GSZ$^+$98] Anuj Goel, Khurram Sajid, Hai Zhou, Adnan Aziz, and Vigyan Singhal. BDD based procedures for a theory of equality with uninterpreted functions. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 244–255, London, UK, 1998. Springer-Verlag.

[GvdP00] Jan Friso Groote and Jaco van de Pol. Equational binary decision diagrams. In *Logic Programming and Automated Reasoning*, pages 161–178, 2000.

[PRSS99] Amir Pnueli, Yoav Rodeh, Ofer Shtrichman, and Michael Siegel. Deciding equality formulas by small domains instantiations. In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 455–469, London, UK, 1999. Springer-Verlag.

[vdPT05] Jaco van de Pol and Olga Tveretina. A BDD-representation for the logic of equality and uninterpreted functions. In *Mathematical Foundations of Computer Science 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 769–780. Springer, 2005.

```
fun Or($\phi_1, \phi_2, \Gamma$) =
  if $\phi_1 = \mathbf{T}$ orelse $\phi_2 = \mathbf{T}$ then $\mathbf{T}$
  else if $\phi_1 = \mathbf{F}$ then $\phi_2$
  else if $\phi_2 = \mathbf{F}$ then $\phi_1$
  else
    let $(x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$ be $\phi_1$
        $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$ be $\phi_2$
    in
      if $(x_1, y_1) = (x_2, y_2)$ then
        let $\psi_2 = $ Or($\phi_{12}, \phi_{22}, \Gamma$)
            $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$
            $\phi'_{11} = $ Filter($\phi_{11}, \Gamma'$)
            $\phi'_{21} = $ Filter($\phi_{21}, \Gamma'$)
            $\psi_1 = $ Or($\phi'_{11}, \phi'_{21}, \Gamma'$)
        in
          Build($x_1 = y_1, \psi_1, \psi_2$)
        end
      else if $(x_1, y_1) \succ (x_2, y_2)$ then
        let $\psi_2 = $ Or($\phi_{12}, \phi_2, \Gamma$)
            $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$
            $\psi_1 = $ Filter($\phi_{11}, \Gamma'$)
        in
          Build($x_1 = y_1, \psi_1, \psi_2$)
        end
      else // $(x_2, y_2) \succ (x_1, y_1)$
        let $\psi_2 = $ Or($\phi_1, \phi_{22}, \Gamma$)
            $\Gamma' = \Gamma[(x_2 = y_2 \wedge [\,]) \vee \psi_2]$
            $\psi_1 = $ Filter($\phi_{21}, \Gamma'$)
        in
          Build($x_2 = y_2, \psi_1, \psi_2$)
        end
      end
    end
```

Fig. 8. Function Or

```
fun Reorder($x = y, \phi_1, \phi_2, \Gamma$) =
  if $\phi_2 = \mathbf{T}$ then $\phi_2$
  else if $\phi_2 = \mathbf{F}$ then
    Reorder1($x = y, \phi_1, \phi_2, \Gamma$)
  else // $\phi_2$ is neither T nor F.
    let $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$ be $\phi_2$
    in
      if $(x, y) = (x_2, y_2)$ then
        Reorder($x = y$, Or($\phi_1, \phi_{21}, \Gamma[(x = y \wedge [\,]) \vee \phi_{22}]$), $\phi_{22}, \Gamma$)
      else if $(x_2, y_2) \succ (x, y)$ then
        Or(Build($x_2 = y_2, \phi_{21}, \mathbf{F}$), Reorder($x = y, \phi_1, \phi_{22}, \Gamma$), $\Gamma$)
      else // $(x, y) \succ (x_2, y_2)$
        Reorder1($x = y, \phi_1, \phi_2, \Gamma$)
    end

and Reorder1($x = y, \phi_1, \phi_2, \Gamma$) =
  if $\phi_1 = \mathbf{F}$ then $\phi_2$
  else if $\phi_1 = \mathbf{T}$ then Build($x = y, \phi_1, \phi_2$)
  else // $\phi_1$ is neither T nor F.
    let $(x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$ be $\phi_1$
    in
      if $(x, y) \succ (x_1, y_1)$ then Build($x = y, \phi_1, \phi_2$)
      else // $(x_1, y_1) \succ (x, y)$
        // Note: $(x_1, y_1) = (x, y)$ is impossible,
        // since $\phi_1$ is reduced in context  $\Gamma[(x = y \wedge [\,]) \vee \phi_2]$.
        let $\psi_2$ = Reorder($x = y, \phi_{12}, \phi_2, \Gamma$)
            $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$
            $\psi_1$ = Reorder($\max_{R_{\Gamma'}}[x] = y, \phi_{11}, \mathbf{F}, \Gamma'$)
        in
          Build($x_1 = y_1, \psi_1, \psi_2$)
        end
    end
```

Fig. 9. Mutually Recursive Functions Reorder and Reorder1

```
fun $\textsc{And}(\phi_1, \phi_2, \Gamma) =$
  if $\phi_1 = \mathbf{F}$ orelse $\phi_2 = \mathbf{F}$ then $\mathbf{F}$
  else if $\phi_1 = \mathbf{T}$ then $\phi_2$
  else if $\phi_2 = \mathbf{T}$ then $\phi_1$
  else // $\phi_1$ and $\phi_2$ are not $\mathbf{T}$ or $\mathbf{F}$
    let $(x_1 = y_1 \wedge \phi_{11}) \vee \phi_{12}$ be $\phi_1$
        $(x_2 = y_2 \wedge \phi_{21}) \vee \phi_{22}$ be $\phi_2$
    in
      if $(x_1, y_1) = (x_2, y_2)$ then
        let $\psi_2 = \textsc{And}(\phi_{12}, \phi_{22}, \Gamma)$
            $\Gamma' = \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2]$
            $\phi'_{12} = \textsc{Filter}(\phi_{12}, \Gamma')$
            $\phi'_{22} = \textsc{Filter}(\phi_{22}, \Gamma')$
            $\psi_3 = \textsc{And}(\phi_{11}, \phi_{21}, \Gamma')$
            $\psi_4 = \textsc{And}(\phi_{11}, \phi'_{22}, \Gamma')$
            $\psi_5 = \textsc{And}(\phi'_{12}, \phi_{21}, \Gamma')$
            $\psi_1 = \textsc{Or}(\textsc{Or}(\psi_3, \psi_4, \Gamma'), \psi_5, \Gamma')$
        in
          $\textsc{Build}(x_1 = y_1, \psi_1, \psi_2)$
        end
      else if $(x_1, y_1) \succ (x_2, y_2)$
        let $\psi_2 = \textsc{And}(\phi_{12}, \phi_2, \Gamma)$
            $\phi'_2 = \textsc{Filter}(\phi_2, \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2])$
            $\psi_1 = \textsc{And}(\phi_{11}, \phi'_2, \Gamma[(x_1 = y_1 \wedge [\,]) \vee \psi_2])$
        in
          $\textsc{Build}(x_1 = y_1, \psi_1, \psi_2)$
        end
      else // $(x_2, y_2) \succ (x_1, y_1)$
        let $\psi_2 = \textsc{And}(\phi_1, \phi_{22}, \Gamma)$
            $\phi'_1 = \textsc{Filter}(\phi_1, \Gamma[(x_2 = y_2 \wedge [\,]) \vee \psi_2])$
            $\psi_1 = \textsc{And}(\phi'_1, \phi_{21}, \Gamma[(x_2 = y_2 \wedge [\,]) \vee \psi_2])$
        in
          $\textsc{Build}(x_2 = y_2, \psi_1, \psi_2)$
        end
    end
```

Fig. 10. Function AND

```
fun $\textsc{Exists}(z, \phi) =$
 let fun $\textsc{Ex}(\psi, V) =$
         if $V$ is empty then $\psi$
         else
            let $v$ be an element of $V$
                $\psi' = \textsc{Reduce}(\phi, [(v = z \wedge [\,]) \vee \mathbf{F}])$
            in
               $\textsc{Ex}(\textsc{Or}(\psi', \psi, [\,]), V \setminus v)$
            end
      $V$ be the set of all variables other than $z$ that occur in $\phi$
 in
    if $\phi = \mathbf{T}$ then $\phi$ else $\textsc{Ex}(\mathbf{F}, V)$
 end
```

Fig. 11. Function Exists

```
fun $\textsc{Forall}(z, \phi) =$
 if $\phi = \mathbf{T}$ orelse $\phi = \mathbf{F}$ then $\phi$
 else // $\phi \neq \mathbf{T}$ and $\phi \neq \mathbf{F}$
    let $(x = y \wedge \phi_1) \vee \phi_2$ be $\phi$
    in
       if $x = y$ involves $z$ then $\textsc{Forall}(z, \phi_2)$
       else $\textsc{Build}(x = y, \textsc{Forall}(z, \phi_1), \textsc{Forall}(z, \phi_2))$
    end
```

Fig. 12. Function Forall

```
fun $\textsc{Implies}(\phi, \psi) =$
 let
    $x$ and $y$ be variables such that $x \succ y$ and $y \succ v$
          for all variables $v$ that occur in $\phi$ or $\psi$
    $\varphi = \textsc{Filter}(\phi, [(x = y \wedge [\,]) \vee ((x = y \wedge \psi) \vee \mathbf{F})])$
 in
    if $\varphi = \mathbf{F}$ then true else false
 end
```

Fig. 13. Function Implies

```
fun RENAME($f, \phi$) =
  let fun BAR($\phi$) =
        if $\phi = \mathbf{T}$ orelse $\phi = \mathbf{F}$ then $\phi$
        else
          let $(x = y \wedge \phi_1) \vee \phi_2$ be $\phi$
          in
            BUILD($\bar{x} = \bar{y}$, BAR($\phi_1$), BAR($\phi_2$))
          end
      $V$ = the set of variables occurring in $\phi$
  in
    REDUCE(BAR($\phi$), $\Gamma_{f,V}$)
  end
```

Fig. 14. Function RENAME