Binary Decision Diagrams for First Order Predicate Logic

Jan Friso Groote

Department of Philosophy, Utrecht University

Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

email: JanFriso.Groote@phil.ruu.nl

Abstract

We present an extension of Binary Decision Diagrams (BDDs) such that they can be used for predicate logic. We present a sound and complete proof search method which we apply to a number of examples.

Key Words & Phrases: Automatic Reasoning, Binary Decision Diagrams, First Order Predicate Logic.

1 Introduction

In 1986 Randy Bryant proposed to represent propositional formulas by *Ordered Binary Decision Diagrams* (*BDD*s) [2]. A BDD is a node-labelled DAG (Directed Acyclic Graph) where in general each node has two outgoing vertices. Bryant provided straightforward algorithms to transform a formula into a BDD and moreover he proved that logically equivalent formulae have canonical BDD representations. In [2] these representations are called '*reduced*'. E.g. tautologies and contradictions have as associated reduced BDDs B_{t} and B_{f} (as depicted in Figure 1 on page 6). This yields a very simple procedure to find out whether a given formula ϕ belongs to one of these classes. Just calculate the reduced BDD of ϕ and see whether it matches B_{t} or B_{f}.

The calculation of BDDs is claimed to belong to the most effective techniques for proving propositional formulae tautologies. And indeed there are examples where BDDs outperform almost all existing techniques with several orders of magnitude, e.g., the Urquhart formulae [15]. There are also claims from various fields that the application of BDD techniques caused substantial breakthroughs (see [8] for VLSI design and [3] for process theory).

This coins the obvious question whether the BDD technique can be made suited for other purposes, with hopefully similar impacts. A primary area to look at is predicate logic.

In this paper we outline a way of extending BDDs to handle predicate logic. Basically it works as follows. Given a formula ϕ that we want to show a tautology. Deny ϕ and calculate the Skolem form of $\neg\phi$, which we call ψ, in order to dispose of quantifiers. We must now show ψ a contradiction. We construct the BDD of ψ in almost the same way as one would construct the BDD of a propositional formula. Now we enter a search procedure where we repeatedly and alternately do the following two operations on the obtained BDD. We calculate so-called *relevant* unifiers and apply these to the BDD. This is done using backtracking. If this does not lead to a proof after an a priori bounded number of steps, we make a copy of the BDD, rename its variables such that they become fresh, and put it in conjunction with the original BDD. Then we start applying unifiers again. If ψ is a contradiction the search will terminate after a finite number of steps.

We have attempted several other approaches, especially those where quantifiers were explicitly incorporated in the representation. But, none of them seemed to work, as they became too complicated. The current approach is very natural and relatively simple. This leads us to think that we have identified a rather natural way to represent and reason within the setting of predicate logic using BDDs.

We provide a number of elementary theorems about this representation. These theorems all work towards the particular proof search technique sketched above. It basically only uses the standard algorithms for finding most general unifiers for terms and the construction of BDDs. Given these algorithms, the presented search technique is rather straightforward.

An interesting question that we have only marginally addressed is how effective our method is. From a theoretical perspective it is hard to say very much about the comparative speeds of the method presented in this article. In absolute sense we know that predicate logic is undecidable and therefore no general terminating algorithm can be presented anyhow. Therefore, it is of course impossible to give a general treatment of efficiency. In relative sense it seems from our own investigations that BDDs for propositional logic are polynomially incomparable to standard techniques such as semantical tableaux and resolution (see [16] for the details of polynomial simulations to compare the different methods). Such results carry over to the setting with predicate logic. But we have not investigated this any further.

One might obtain some insight by implementing the proposed algorithm. As this is a far from trivial job, we leave this to others. We have only experimented by hand with proving numerous small problems for which easy proofs turned out to exist (see section 8). The method proposed in this article must be seen as an initial step towards a full fledged system. As we have seen with other major streams in automatic theorem proving basic methods must be extended substantially before they can get to work. E.g. for resolution there exist many variants of which hyper resolution and SLD-resolution (for PROLOG) are probably the most well known. Furthermore, features have been added to resolution for special purposes, such as modulation or paramodulation to handle equality.

This article is organised as follows. In section 2 predicate logic is introduced. In section 3 we define how binary decision diagrams for predicate logic look like. In sections 4 and 5 we provide a number of operations on BDDs of which we give the main completeness theorem in section 6. In section 7 we present the proof search algorithm and in section 8 we show how the method works on three examples taken from [11].

Acknowledgements. I thank Jaco van de Pol and Hans Zantema for their assistance in proving termination of the basic operators on BDDs. I also thank Jaco van de Pol for his detailed comments, that contributed considerably to the quality of this paper. Thanks also go to Jean Goubault, Joachim Posegga and Bas van Vlijmen for general discussion and comments.

2 First order predicate logic

In the sequel we assume a set $V = \{x_1, x_2, \ldots\}$ of variables, a set $F = \{f_1, f_2, \ldots\}$ of function and a set $Pr = \{P_1, P_2, \ldots\}$ of predicate symbols and we assume that we know the arity of each function symbol in F and of each predicate in Pr. The sets V, F and Pr are pairwise disjoint. If convenient we also use other letters than x, f and P to refer to variables, function- and predicate symbols.

Definition 2.1. *Terms* are inductively defined by:

- $x \in V$ is a term,
- if $f \in F$ is a function symbol of arity $r \geq 0$ and $t_1, \ldots, t_r$ are terms, then $f(t_1, \ldots, t_r)$ is a term.

The set of all terms over F and V is denoted by $\mathbb{T}(F, V)$ and the set of all predicates of the form $P(t_1, \ldots, t_r)$ with $t_1, \ldots, t_r$ terms and $P \in Pr$ is denoted by $\mathbb{P}(Pr, F, V)$. Terms not containing variables are called *closed*. For sequences of terms we use the vector notation, e.g., $\vec{t} = t_1, \ldots, t_n$. A substitution is a mapping $\zeta : V \to \mathbb{T}(F, V)$. The notation $\zeta[x_1{:=}t]$ represents a substitution ζ that maps each variable x to $\zeta(x)$, except that it maps x_1 to t. The substitution $\zeta[\vec{x}{:=}\vec{t}]$ behaves like ζ, except that it replaces variables in $\vec{x}$ by the corresponding term in $\vec{t}$. A substitution ζ is closed if $\zeta(x)$ contains no variables. We use '$\circ$' for composition of substitutions: $\zeta \circ \xi(t) = \zeta(\xi(t))$, and ι is

the identical substitution. We assume that ζ is extended to a mapping from terms to terms and from predicates to predicates in the standard way.

Formulas are inductively defined by:

- $\mathfrak{t}$ and $\mathfrak{f}$ are formulas,
- $P(t_1, \ldots, t_r) \in \mathbb{P}(Pr, F, V)$ is a formula,
- if ϕ is a formula, then $\neg\phi$ is a formula,
- if ϕ and ψ are formulas, then $\phi \wedge \psi$ is a formula,
- if ϕ is a formula, and $x \in V$ is a variable, then $\forall x.\phi$ and $\exists x.\phi$ are formulas.

The set of all formulas is denoted by $\mathbb{F}(Pr, F, V)$. The abbreviation $\phi \vee \psi$ stands for $\neg(\neg\phi \wedge \neg\psi)$, $\phi \to \psi$ stands for $\neg\phi \vee \psi$, and $\phi \leftrightarrow \psi$ represents $(\phi \to \psi) \wedge (\psi \to \phi)$. We assume that substitutions extend to formulas in the standard way.

Definition 2.2. A *structure* is a multi-tuple $\mathfrak{A} = \langle A; R_1, R_2, \ldots; F_1, F_2, \ldots\rangle$ where

- A is a non-empty set,
- $R_1, R_2, \ldots$ are relations on A. The arity of R_i is equal to the arity of the predicate symbol P_i.
- $F_1, F_2, \ldots$ are functions on A. The arity of F_j is equal to the arity of function symbol f_j.

Herbrand structures are particularly interesting, as they connect the semantical world of interpretations and the syntactical world of symbolic manipulation. Herbrand structures have the form $\mathfrak{A}_H = \langle \mathbb{T}(F, \emptyset), R_1, \ldots; f_1, \ldots, f_n\rangle$. I.e., the domain A consists exactly of all closed terms, and each function symbol is interpreted by itself. Relations can be chosen freely.

Definition 2.3. Let $\mathfrak{A} = \langle A; R_1, \ldots; F_1, \ldots\rangle$ be a structure and $\zeta : V \to A$ be a *valuation*. The *interpretation* $[\![t]\!]^\zeta_\mathfrak{A} : \mathbb{T}(F, V) \to A$ of a term t is inductively defined by:

- $[\![x]\!]^\zeta_\mathfrak{A} = \zeta(x)$ if $x \in V$,
- $[\![f_j(t_1, \ldots, t_r)]\!]^\zeta_\mathfrak{A} = F_j([\![t_1]\!]^\zeta_\mathfrak{A}, \ldots, [\![t_r]\!]^\zeta_\mathfrak{A})$.

The *interpretation* $[\![\phi]\!]^\zeta_\mathfrak{A} : \mathbb{P}(Pr, F, V) \to \{0, 1\}$ of a formula ϕ is inductively defined by:

- $[\![\mathfrak{f}]\!]^\zeta_\mathfrak{A} = 0$,
- $[\![\mathfrak{t}]\!]^\zeta_\mathfrak{A} = 1$,
- $[\![p_i(t_1, \ldots, t_r)]\!]^\zeta_\mathfrak{A} = \begin{cases} 1 & \text{if } \langle [\![t_1]\!]^\zeta_\mathfrak{A}, \ldots, [\![t_r]\!]^\zeta_\mathfrak{A}\rangle \in R_i, \\ 0 & \text{otherwise,} \end{cases}$
- $[\![\neg\phi]\!]^\zeta_\mathfrak{A} = 1 - [\![\phi]\!]^\zeta_\mathfrak{A}$,
- $[\![\phi \wedge \psi]\!]^\zeta_\mathfrak{A} = \min([\![\phi]\!]^\zeta_\mathfrak{A}, [\![\psi]\!]^\zeta_\mathfrak{A})$,
- $[\![\forall x.\phi]\!]^\zeta_\mathfrak{A} = \min_{a \in A}([\![\phi]\!]^{\zeta[x:=a]}_\mathfrak{A})$,
- $[\![\exists x.\phi]\!]^\zeta_\mathfrak{A} = \max_{a \in A}([\![\phi]\!]^{\zeta[x:=a]}_\mathfrak{A})$.

We write $\mathfrak{A}, \zeta \models \phi$ iff $[\![\phi]\!]^{\zeta}_{\mathfrak{A}} = 1$, and $\mathfrak{A}, \zeta \not\models \phi$ iff $[\![\phi]\!]^{\zeta}_{\mathfrak{A}} = 0$. We write $\mathfrak{A} \models \phi$ iff for all valuations ζ it holds that $\mathfrak{A}, \zeta \models \phi$. We say that ϕ is a *tautology*, notation $\models \phi$ if for all structures $\mathfrak{A}$ it holds that $\mathfrak{A} \models \phi$. If for each structure $\mathfrak{A}$ there is a valuation ζ such that $\mathfrak{A}, \zeta \not\models \phi$, we say that ϕ is *unsatisfiable*. Otherwise we say that ϕ is satisfiable.

We say that formulas ϕ and ψ are *strongly (logically) equivalent*, notation

$$\phi \cong \psi,$$

iff for all structures $\mathfrak{A}$ and all valuations ζ it holds that $\mathfrak{A}, \zeta \models \phi$ iff $\mathfrak{A}, \zeta \models \psi$. We say that formulas ϕ and ψ are *logically equivalent*, notation

$$\phi \approx \psi,$$

iff for all structures $\mathfrak{A}$ it holds that $\mathfrak{A} \models \phi$ iff $\mathfrak{A} \models \psi$. We say that ϕ and ψ are *weakly (logically) equivalent*, notation

$$\phi \sim \psi,$$

iff there is a structure $\mathfrak{A}$ and a valuation ζ such that $\mathfrak{A}, \zeta \models \phi$ iff there is a structure $\mathfrak{B}$ and a valuation η such that $\mathfrak{B}, \eta \models \psi$.

Note that logical equivalence is the ordinary notion of equivalence and that strong logical equivalence implies logical equivalence. For formulas in which no free variables occur strong logical equivalence and logical equivalence coincide, and logical equivalence implies weak logical equivalence. Furthermore, observe that if $\phi \sim \psi$ and ϕ is unsatisfiable, then ψ is also unsatisfiable.

There are numerous standard facts about first order predicate logic. We list three main results that are used in the sequel. The following theorem expresses that for each formula there is a corresponding formula which has only a set of leading universal quantifiers.

Theorem 2.4. *Let ϕ be a formula. Then there exist variables $x_1, \ldots, x_n$ and a quantifier free formula ψ such that*

$$\phi \ \sim \ \forall x_1 \ldots \forall x_n . \psi.$$

The formula $\forall x_1 \ldots \forall x_n . \psi$ is called the Skolem form or formula of ϕ.

Skolem formulas can efficiently be calculated.

We are essentially interested in proving a formula from predicate logic a tautology. This is equivalent to showing that the formula $\neg\phi$ is unsatisfiable. Using the previous theorem $\neg\phi$ can be transformed to a Skolem formula ψ maintaining unsatisfiability. The following theorem that restricts attention to a finite number of instances of ψ, is the basis of our proof procedure.

Theorem 2.5 *(Herbrands Theorem). Let ϕ be a quantifier free formula in which the variables $\vec{x} = x_1, \ldots, x_m$ occur. The formula ϕ is unsatisfiable iff there are closed terms $\vec{t_1}, \vec{t_2}, \ldots, \vec{t_n}$ such that $\bigwedge_{i=1}^{n} \phi[\vec{x} := \vec{t_i}] \cong \mathfrak{f}$.*

As we have restricted our attention to unsatisfiability of Skolemised formulae, we can also restrict our attention to Herbrand structures, using the following theorem. So, from now on, reference to structures means reference to a Herbrand structure.

Theorem 2.6. *Let ϕ be a formula in Skolem form. There is a structure $\mathfrak{A}$ and a valuation ζ such that $\mathfrak{A}, \zeta \models \phi$ iff there is a Herbrand structure $\mathfrak{A}_H$ and a valuation ξ such that $\mathfrak{A}_H, \xi \models \phi$.*

Note that the valuation ξ above is actually a closed substitution. So, from now on, closed substitutions and valuations can be identified.

3 Binary Decision Diagrams

In this section we define binary decision diagrams almost completely according to Bryant [2]. The only real difference is that we allow predicates as labels instead of proposition symbols.

Definition 3.1. A *Binary Decision Diagram* (*BDD*) $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ is an acyclic, node labelled graph where

- Q is a finite set of *nodes*,
- $l : Q \cup \{0,1\} \to \mathbb{P}(Pr, F, V) \cup \{0,1\}$ is a *node labelling*, satisfying that $l(0) = 0$, $l(1) = 1$ and $l(q) \neq 0, 1$ for all $q \in Q$.
- $\xrightarrow{\mathfrak{f}}: Q \to Q \cup \{0,1\}$ is the *false continuation* of a node,
- $\xrightarrow{\mathfrak{t}}: Q \to Q \cup \{0,1\}$ is the *truth continuation* of a node,
- $s \in Q \cup \{0,1\}$ is the start node; we assume that all nodes in Q are reachable from s using truth or false continuations,
- $0 \notin Q$ is a symbol representing false, and $1 \notin Q$ is a symbol representing truth.

The BDD B is acyclic in the sense that there is no infinite sequence of nodes $q_0 \xrightarrow{\diamond_0} q_1 \xrightarrow{\diamond_1} \ldots$ where for each $i \geq 0$ $\diamond_i = \mathfrak{f}$ or $\diamond_i = \mathfrak{t}$.

Note that as a consequence of the acyclicity of a BDD and the finiteness of the set of nodes each sequence $q_0 \xrightarrow{\diamond_1} q_1 \xrightarrow{\diamond_2} \ldots$ is bounded and, if it cannot be extended, must end in 0 or 1.

Notation 3.2. Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD. We use the following notations.

- $B{\uparrow}$ for the initial node s,
- Q_B for Q,
- $p{\downarrow_\mathfrak{t}}$ for the node q such that $p \xrightarrow{\mathfrak{t}} q$,
- $p{\downarrow_\mathfrak{f}}$ for the node q such that $p \xrightarrow{\mathfrak{f}} q$.

We assume a total ordering $>$ on $\mathbb{P}(Pr, F, V) \cup \{0,1\}$ such that $0 > P(t_1, \ldots, t_r)$ and $1 > P(t_1, \ldots, t_r)$ for all predicates $P(t_1, \ldots, t_r)$.

Definition 3.3. *(Interpretation of a BDD).* Let B be a BDD and let $\mathfrak{A}$ be a structure and ζ be a valuation. A $\mathfrak{A}$,ζ-*path* of a node $q_0 \in Q_B$ is the sequence

$$q_0 \xrightarrow{\diamond_0} q_1 \xrightarrow{\diamond_1} \ldots \xrightarrow{\diamond_{n-1}} q_n$$

where $q_n \in \{0,1\}$ and for each $0 \leq i < n$ $\diamond_i = \mathfrak{f}$ if $\mathfrak{A}, \zeta \not\models l(q_i)$ and $\diamond_i = \mathfrak{t}$ if $\mathfrak{A}, \zeta \models l(q_i)$. If the $\mathfrak{A}, \zeta$-path of q_0 ends in 1 we say that q_0 holds, notation $\mathfrak{A}, \zeta \models q_0$. Otherwise, i.e. when the $\mathfrak{A}, \zeta$-path of q_0 ends in 0, we say that q_0 does not hold, notation $\mathfrak{A}, \zeta \not\models q_0$. We write $\mathfrak{A}, \zeta \models B$ for $\mathfrak{A}, \zeta \models B{\uparrow}$ and $\mathfrak{A}, \zeta \not\models B$ for $\mathfrak{A}, \zeta \not\models B{\uparrow}$. Using this definition, the relations $\approxeq$ (strong equivalence), $\approx$ (logical equivalence) and $\sim$ (weak equivalence) and the notions tautology, satisfiable- and unsatisfiable formulas carry over to BDDs and nodes of BDDs.

So, a BDD yields, given a structure and a valuation, a truth value. As such they can be used to represent formulas. The following definition explains a way to do this. We sometimes use pictures, instead of rather laborious definitions of BDDs, as we think that these are as clear, and far easier to understand. We have adopted the convention to draw outgoing false continuations at the left and outgoing truth continuations at the right of a node. We tag the nodes only with their labels and we draw multiple occurrences of single node labelled with 0, and similarly for the unique node labelled with 1.

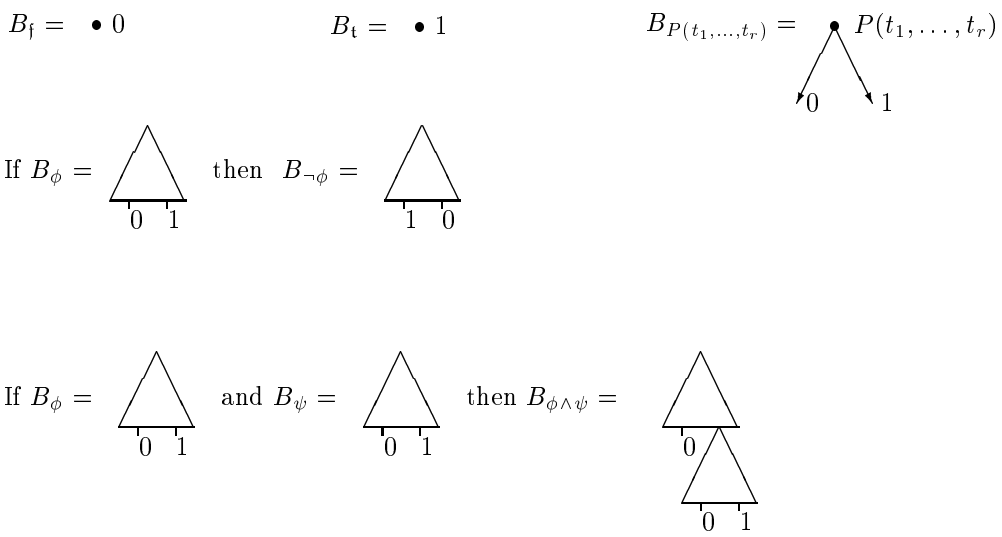

Figure 1: Definitions of $B_{\mathfrak{f}}, B_{\mathfrak{t}}, B_{P(t_1,\ldots,t_r)}$, $B_{\neg\phi}$ and $B_{\phi\wedge\psi}$

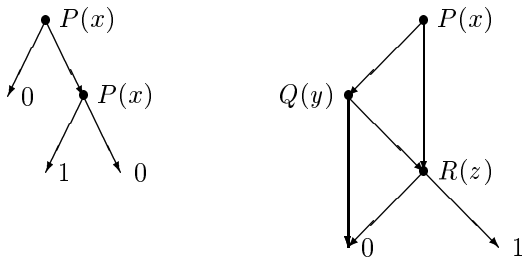

Figure 2: BDDs for $P(x) \wedge \neg P(x)$ and $(P(x) \vee Q(y)) \wedge R(z)$

Definition 3.4. Figure 1 shows the BDDs, $B_{\mathfrak{f}}$, $B_{\mathfrak{t}}$, $B_{P(t_1,\ldots,t_r)}$, $B_{\neg\phi}$ and $B_{\phi\wedge\psi}$ corresponding to the formulas $\mathfrak{f}$, $\mathfrak{t}$, $P(t_1,\ldots,t_r)$, $\neg\phi$ and $\phi \wedge \psi$. In $B_{\phi\wedge\psi}$ it does not matter which diagram is put on top and which one is put below.

Note that we divert here from [2] wrt. the definition of $B_{\phi\wedge\psi}$, where a strict ordering on the labels of the nodes is maintained. In [2] it is guaranteed that when traversing a BDD from the root to 0 or 1, the labels are run across in a strict ascending order. We introduce special rules to sort the labels, as we need these when applying unifiers. As these sorting rules are available anyhow, we have chosen for the simpler presentation of conjunction. As sorting a BDD is a very expensive operation, it seems wise to implement $\wedge$ on BDDs as is done in [1, 2].

Example 3.5. The BDDs belonging to the formulas $P(x) \wedge \neg P(x)$ and $(P(x) \vee Q(y)) \wedge R(z)$ are drawn in figure 2.

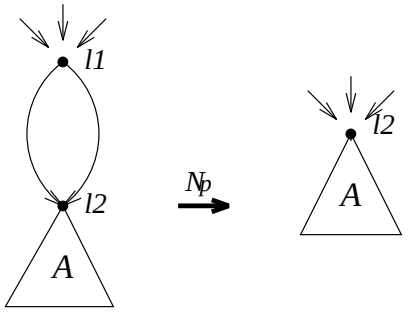

Figure 3: The neglect operator $N_p(B)$

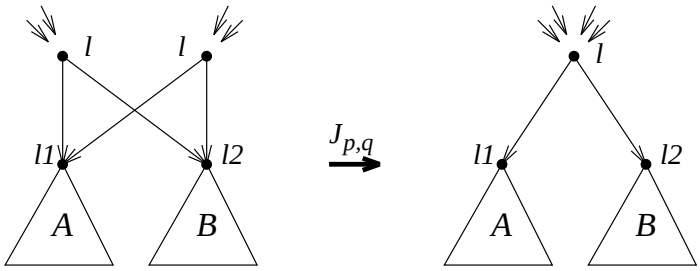

Figure 4: The p,q-join operator $J_{p,q}(B)$

Theorem 3.6. *Let ϕ be a (quantifier free) formula and B_ϕ its corresponding BDD. For each structure $\mathfrak{A}$ and each valuation ζ we find that*

$$\mathfrak{A},\zeta \models \phi \quad \textit{iff} \quad \mathfrak{A},\zeta \models B_\phi.$$

Proof. Straightforward on the structure of ϕ. □

4 Simple operations on BDDs

In this section we provide simple operations to transform BDDs into *reduced* or canonical form [2]. We show that the reduced BDDs of (strongly) equivalent formulas are isomorphic (Theorem 4.10) and that the application of simple operators must terminate (Theorem 4.11).

The operators N_p and $J_{p,q}$ are the same as those in [1], where it is pointed out how simultaneous application of these two operators can be carried out on a BDD in linear time. Because of the details of the operators are somewhat tricky, we give the definitions in full detail. For easy understanding each operator is depicted.

Definition 4.1. *(see Figure 3).* Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD. The *neglect operator* $N_p(B)$ is defined if for some $q \in Q$ $p \xrightarrow{\mathfrak{t}} q$ and $p \xrightarrow{\mathfrak{f}} q$ by:

$$N_p(B) = (Q', l, \xrightarrow{\mathfrak{f}}{}', \xrightarrow{\mathfrak{t}}{}', s', 0, 1)$$

where

$$\xrightarrow{\diamond}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\diamond} \mid r_1 \neq p, r_2 \neq p\} \cup \{\langle r_1, q\rangle \mid r_1 \xrightarrow{\diamond} p\} \quad \text{for } \diamond \in \{\mathfrak{t}, \mathfrak{f}\}$$

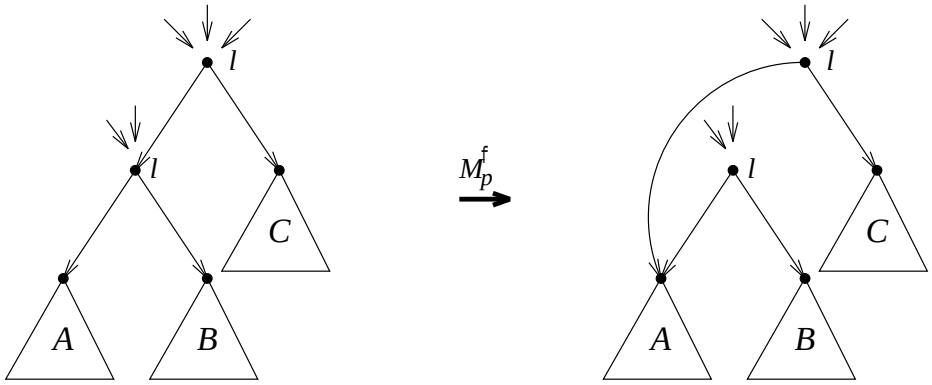

Figure 5: The f-merge operator $M_p^f(B)$

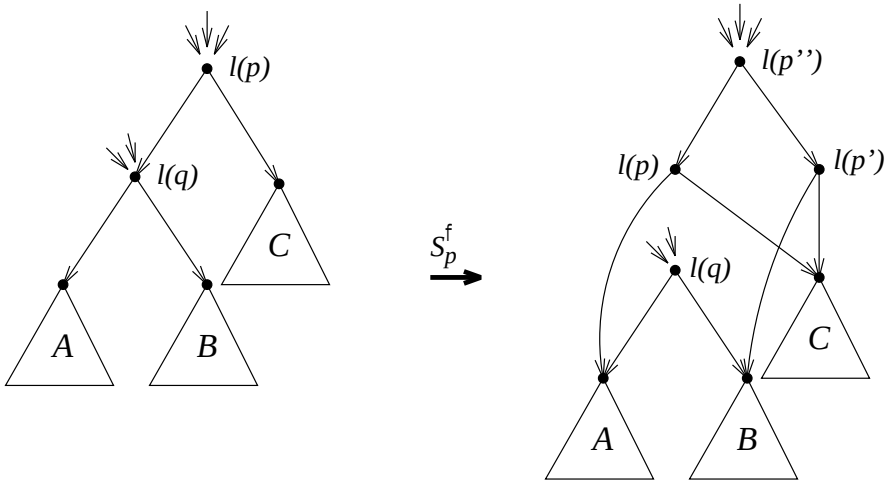

Figure 6: The f-sort operator $S_p^f(B)$, $l(p) > l(q)$, $l(p) = l(p')$ and $l(q) = l(p'')$

$$s' = \begin{cases} s & \text{if } s \neq p, \\ q & \text{if } s = p \end{cases}$$

$$Q' = Q \setminus \{p\}.$$

Definition 4.2. *(see Figure 4).* Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD. If $p, q \in Q$, $p \neq q$, $l(p) = l(q)$, $p \xrightarrow{\mathfrak{f}} r$, $p \xrightarrow{\mathfrak{t}} r'$, $q \xrightarrow{\mathfrak{f}} r$ and $q \xrightarrow{\mathfrak{t}} r'$ for some $r, r' \in Q$, then the *p,q-join operator* $J_{p,q}(B)$ is the BDD

$$(Q', l, \xrightarrow{\mathfrak{f}}{}', \xrightarrow{\mathfrak{t}}{}', s', 0, 1)$$

where

$$\xrightarrow{\mathfrak{f}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathfrak{f}} | r_2 \neq q\} \cup \{\langle r_1, p\rangle | r_1 \xrightarrow{\mathfrak{f}} q\}$$

$$\xrightarrow{\mathfrak{t}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathfrak{t}} | r_2 \neq q\} \cup \{\langle r_1, p\rangle | r_1 \xrightarrow{\mathfrak{t}} q\}$$

$$s' = \begin{cases} s & \text{if } s \neq q, \\ p & \text{if } s = q \end{cases}$$

$$Q' = Q \setminus \{q\}.$$

The operators $M_p^{\mathfrak{f}}$, $M_p^{\mathfrak{t}}$, $S_p^{\mathfrak{f}}$ and $S_p^{\mathfrak{t}}$ sort the BDD such that labels occur in a strict ascending order. It is impossible to implement simultaneous application of these operators on a BDD efficiently, as sorting a BDD is NP-hard[1]. It is possible to avoid sorting a BDD, except after application of a unifier.

Definition 4.3. *(see Figure 5).* Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD. If $p, q \in Q$, $p \xrightarrow{\mathfrak{f}} q$, and $l(p) = l(q)$, then the *$\mathfrak{f}$-merge operator* $M_p^{\mathfrak{f}}(B)$ is the BDD

$$(Q', l, \xrightarrow{\mathfrak{f}}{}', \xrightarrow{\mathfrak{t}}, s, 0, 1)$$

where

$$\xrightarrow{\mathfrak{f}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathfrak{f}} | r_1 \neq p \wedge r_2 \neq q\} \cup \{\langle p, r\rangle | q \xrightarrow{\mathfrak{f}} r\}$$

Q' is Q from which non reachable parts are removed.

If $p, q \in Q$, $p \xrightarrow{\mathfrak{t}} q$ and $l(p) = l(q)$, then the *$\mathfrak{t}$-merge operator* $M_p^{\mathfrak{t}}(B)$ is the BDD

$$(Q', l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}{}', s, 0, 1)$$

where

$$\xrightarrow{\mathfrak{t}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathfrak{t}} | r_1 \neq p \wedge r_2 \neq q\} \cup \{\langle p, r\rangle | q \xrightarrow{\mathfrak{t}} r\}$$

Q' is Q from which non reachable parts are removed.

Definition 4.4. *(see Figure 6).* Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD. If $p, q \in Q$, $p \xrightarrow{\mathfrak{f}} q$ and $l(p) > l(q)$ with respect to the ordering relation $>$ on predicates, then the *$\mathfrak{f}$-sort operation* is defined as follows:

$$S_p^{\mathfrak{f}} = (Q', l', \xrightarrow{\mathfrak{t}}{}', \xrightarrow{\mathfrak{f}}{}', s', 0, 1).$$

[1]When applying the operations on BDDs as described in this article to propositional logic, the only non polynomial operator is sorting.

Below p' and p'' are new nodes.

$$l'(r) = \begin{cases} l(p) & \text{if } r = p', \\ l(q) & \text{if } r = p'', \\ l(r) & \text{otherwise} \end{cases}$$

$$\xrightarrow{\mathsf{f}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathsf{f}} \mid r_1 \neq p \text{ or } r_2 \neq p\} \cup \{\langle r, p''\rangle \mid r \xrightarrow{\mathsf{f}} p\} \cup \{\langle p'', p\rangle\} \cup \{\langle p', r\rangle \mid q \xrightarrow{\mathsf{t}} r\} \cup \{\langle p, r\rangle \mid q \xrightarrow{\mathsf{f}} r\}$$

$$\xrightarrow{\mathsf{t}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathsf{t}} \mid r_2 \neq p\} \cup \{\langle r, p''\rangle \mid r \xrightarrow{\mathsf{t}} p\} \cup \{\langle p'', p'\rangle\} \cup \{\langle p', r\rangle \mid p \xrightarrow{\mathsf{t}} r\}$$

Q' is $Q \cup \{p', p''\}$ from which parts that become unreachable are removed.

If $p, q \in Q$, $p \xrightarrow{\mathsf{t}} q$ and $l(p) > l(q)$ with respect to the ordering relation $>$ on predicates, then the t-*sort operation* is defined to be

$$S^{\mathsf{t}}_p(B) = (Q', l', \xrightarrow{\mathsf{t}}{}', \xrightarrow{\mathsf{f}}{}', s', 0, 1).$$

Below, p' and p'' are new nodes.

$$l'(r) = \begin{cases} l(p) & \text{if } r = p', \\ l(q) & \text{if } r = p'', \\ l(r) & \text{otherwise} \end{cases}$$

$$\xrightarrow{\mathsf{f}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathsf{f}} \mid r_2 \neq p\} \cup \{\langle r, p''\rangle \mid r \xrightarrow{\mathsf{f}} p\} \cup \{\langle p'', p'\rangle\} \cup \{\langle p', r\rangle \mid p \xrightarrow{\mathsf{f}} r\}$$

$$\xrightarrow{\mathsf{t}}{}' = \{\langle r_1, r_2\rangle \in \xrightarrow{\mathsf{t}} \mid r_1 \neq p \text{ or } r_2 \neq p\} \cup \{\langle r, p''\rangle \mid r \xrightarrow{\mathsf{t}} p\} \cup \{\langle p'', p\rangle\} \cup \{\langle p', r\rangle \mid q \xrightarrow{\mathsf{f}} r\} \cup \{\langle p, r\rangle \mid q \xrightarrow{\mathsf{t}} r\}$$

Q' is $Q \cup \{p', p''\}$ from which parts that become unreachable are removed.

Lemma 4.5. *(Soundness). Let B be a BDD. We find for $p, q \in Q_B$ that in case O is applicable to B*

$$O(B) \cong B$$

where O is one of N_p, $J_{p,q}$, M^{t}_p, M^{f}_p, S^{t}_p and S^{f}_p.

Proof. It is trivial but tedious, to check that for all structures $\mathfrak{A}$ and valuations ζ it holds that $\mathfrak{A}, \zeta \models O(B)$ iff $\mathfrak{A}, \zeta \models B$. □

Definition 4.6. We say that a BDD B is *reduced* with respect to some total ordering $<$ on open predicates iff none of the operators N_p, $J_{p,q}$, M^{t}_p, M^{f}_p, S^{t}_p and S^{f}_p is applicable to B. In general the ordering $<$ is not mentioned, assuming it is clear from the context.

The next lemmas work towards Theorem 4.10 saying that strongly equivalent reduced BDDs are unique up to an isomorphism.

Lemma 4.7. *Let B, C be BDDs with nodes $p \in Q_B$ and $q \in Q_C$. Let $\mathfrak{A}$ be a structure and ζ a valuation such that $\mathfrak{A}, \zeta \models p$ and $\mathfrak{A}, \zeta \not\models q$. Let $P(t_1, \ldots, t_n)$ be a label not occurring in the subdags of B and C rooted with p and q. Then:*

1. *There exists a structure $\mathfrak{B}$ and a valuation ξ such that $\mathfrak{B}, \xi \models p$, $\mathfrak{B}, \xi \not\models q$ and $\mathfrak{B}, \xi \models P(t_1, \ldots, t_n)$.*
2. *There exists a structure $\mathfrak{B}$ and a valuation ξ such that $\mathfrak{B}, \xi \models p$, $\mathfrak{B}, \xi \not\models q$ and $\mathfrak{B}, \xi \not\models P(t_1, \ldots, t_n)$.*

Proof. Extend $\mathfrak{A}$ with new fresh constants, one for every variable in B, C or $P(t_1, \ldots, t_n)$. Define ξ such that it maps every variable to this newly created constant. Define $\mathfrak{B}$ to hold on every predicate $\xi(Q(u_1, \ldots, u_m))$ iff $\mathfrak{A}$ holds for $\zeta(Q(u_1, \ldots, u_m))$. Due to the structure of ξ, this is well defined. Moreover, it leaves open whether $\mathfrak{B}, \xi \models P(t_1, \ldots t_n)$ or not. □

Lemma 4.8. *Let B and C be reduced BDDs. Let $p \in Q_B$ and $q \in Q_C$ such that $p \cong q$. We find:*

1. $l(p) = l(q)$,
2. $p{\downarrow_{\mathfrak{f}}} \cong q{\downarrow_{\mathfrak{f}}}$,
3. $p{\downarrow_{\mathfrak{t}}} \cong q{\downarrow_{\mathfrak{t}}}$,
4. *if B and C are the same BDDs, then $p = q$.*

Proof. We prove this lemma by contradiction. Assume there are reduced BDDs B and C containing nodes $p \in Q_B$ and $q \in Q_C$ such that one of the conditions 1,2,3 or 4 do not hold. Consider such BDDs B and C with nodes p, q with minimal value $2^{|p|} + 2^{|q|}$ where $|p|$ is the length of the longest path leading from p to 0 or 1.

1. Suppose $l(p) \neq l(q)$. Consider the case where $l(p) < l(q)$. The case where $l(p) > l(q)$ is symmetric, and therefore omitted. As the sort and the merge operator are not applicable to B and C, there are no nodes below p in B and below q in C labelled with $l(p)$. Now we can show that $p{\downarrow_{\mathfrak{f}}} \cong q$. The proof of this fact is also by contradiction. Assume $p{\downarrow_{\mathfrak{f}}} \ncong q$. Then, there is a structure $\mathfrak{A}$ and a valuation ζ such that $\mathfrak{A}, \zeta \models p{\downarrow_{\mathfrak{f}}}$ and $\mathfrak{A}, \zeta \not\models q$, or vice versa, $\mathfrak{A}, \zeta \models p{\downarrow_{\mathfrak{f}}}$ and $\mathfrak{A}, \zeta \not\models q$. We only deal with the first case, as the other is almost symmetric. By Lemma 4.7, there is a structure $\mathfrak{B}$ and a valuation ξ such that $\mathfrak{B}, \xi \models p{\downarrow_{\mathfrak{f}}}$, $\mathfrak{B}, \xi \not\models q$ and $\mathfrak{B}, \xi \not\models l(p)$. Hence, $\mathfrak{B}, \xi \models p$. But this contradicts that $p \cong q$, as $\mathfrak{B}, \xi \not\models q$. So, $p{\downarrow_{\mathfrak{f}}} \cong q$. Similarly, we can show that $p{\downarrow_{\mathfrak{t}}} \cong q$. Hence, by transitivity of $\cong$, $p{\downarrow_{\mathfrak{t}}} \cong p{\downarrow_{\mathfrak{f}}}$. As $2^{|p{\downarrow_{\mathfrak{t}}}|} + 2^{|p{\downarrow_{\mathfrak{f}}}|} \leq 2^{|p|} < 2^{|p|} + 2^{|q|}$ it must be that $p{\downarrow_{\mathfrak{t}}} = p{\downarrow_{\mathfrak{f}}}$, as p and q were the nodes with smallest exponential distance to endnodes violating one of the properties 1 to 4 in this lemma. But in this case the neglect operator is applicable, contradicting that B is reduced. Hence, $l(p) = l(q)$.
2. Suppose $l(p) = l(q)$, but $p{\downarrow_{\mathfrak{f}}} \ncong q{\downarrow_{\mathfrak{f}}}$. Then there is a structure $\mathfrak{A}$ and some valuation ζ such that $\mathfrak{A}, \zeta \models p{\downarrow_{\mathfrak{f}}}$ and $\mathfrak{A}, \zeta \not\models q{\downarrow_{\mathfrak{f}}}$ or vice versa, $\mathfrak{A}, \zeta \not\models p{\downarrow_{\mathfrak{f}}}$ and $\mathfrak{A}, \zeta \models q{\downarrow_{\mathfrak{f}}}$. We only consider the first case for symmetry reasons. As $l(p)$ does not occur in the subdags in B and C rooted with $l(p)$, there is according to Lemma 4.7 a structure $\mathfrak{B}$ and a valuation ξ such that $\mathfrak{B}, \xi \models p{\downarrow_{\mathfrak{f}}}$, $\mathfrak{B}, \xi \not\models q{\downarrow_{\mathfrak{f}}}$ and $\mathfrak{B}, \xi \not\models l(p)$. Hence, $\mathfrak{B}, \xi \models p$, $\mathfrak{B}, \xi \not\models q$ contradicting that $p \cong q$.
3. Similar to case 2.
4. Let B and C be the same BDDs. Suppose that cases 1,2 and 3 hold for p and q. In order to generate a contradiction, assume $p \neq q$. Using 2 and 3, $p{\downarrow_{\mathfrak{f}}} \cong q{\downarrow_{\mathfrak{f}}}$ and $p{\downarrow_{\mathfrak{t}}} \cong q{\downarrow_{\mathfrak{t}}}$. As $2^{|p{\downarrow_{\mathfrak{f}}}|} + 2^{|q{\downarrow_{\mathfrak{f}}}|} < 2^{|p|} + 2^{|q|}$, it follows that $p{\downarrow_{\mathfrak{f}}} = q{\downarrow_{\mathfrak{f}}}$. In the same way it follows that $p{\downarrow_{\mathfrak{t}}} = q{\downarrow_{\mathfrak{t}}}$. Hence the join operator is applicable. But this contradicts that B is reduced.

□

Definition 4.9. Let $B = (Q_B, l_B, \xrightarrow{\mathfrak{f}}_B, \xrightarrow{\mathfrak{t}}_B, s_B, 0_B, 1_B)$ and $C = (Q_C, l_C, \xrightarrow{\mathfrak{f}}_C, \xrightarrow{\mathfrak{t}}_C, s_C, 0_C, 1_C)$ be BDDs. A function $f : Q_B \cup \{0_B, 1_B\} \to Q_C \cup \{0_C, 1_C\}$ is called a *homomorphism* iff $l_C(f(p)) = l_B(p)$, $f(p{\downarrow_{\mathfrak{f}}}) = f(p){\downarrow_{\mathfrak{f}}}$ and $f(p{\downarrow_{\mathfrak{t}}}) = f(p){\downarrow_{\mathfrak{t}}}$. In case f is bijective, f is called an *isomorphism*. If there exists an isomorphism $f : Q_B \cup \{0_B, 1_B\} \to Q_C \cup \{0_C, 1_C\}$, then B and C are called *isomorphic*, written as $B = C$.

Theorem 4.10. *Let B and C be reduced BDDs, such that $B \cong C$. Then B and C are isomorphic, i.e. $B = C$.*

Proof. We define functions $f : Q_B \to Q_C$ and $g : Q_C \to Q_B$ as follows.

$$f(p) = q \quad \text{for the } q \in Q_C \text{ such that } p \cong q,$$
$$g(q) = p \quad \text{for the } p \in Q_B \text{ such that } p \cong q.$$

Assuming that f and g are well defined functions, it is easy to see that f is a homomorphism using Lemma 4.8. Furthermore, g is clearly the inverse of f, proving f an isomorphism.

So, we must only show that f and g are proper functions. Due to symmetry we only do that for f. As $B \cong C$, it follows from Lemmas 4.8.2 and 4.8.3 and the fact that all nodes in B are reachable from the root that each node in Q_B is related via $\cong$ to at least one node in Q_C. Now, suppose that a node $p \in Q_B$ is related to nodes $q_1, q_2 \in Q_C$. By transitivity of $\cong$ it follows that $q_1 \cong q_2$. Using Lemma 4.8.4 it must be that $q_1 = q_2$. □

Theorem 4.11. *Let B be a BDD. The operators N_p, $J_{p,q}$, M_p^{t}, M_p^{f}, S_p^{t} and S_p^{f} can only be applied a finite number of times to B.*

Proof. The transformation operators can be formulated as rewrite rules in the following way (except the join operator), where l_1 and l_2 are predicates with $l_1 > l_2$.

$$\begin{array}{rcll} l_1(l_2(x,y),z) & \to & l_2(l_1(x,z),l_1(y,z)) & S_p^{\mathrm{f}} \\ l_1(x,l_2(y,z)) & \to & l_2(l_1(x,y),l_1(x,z)) & S_p^{\mathrm{t}} \\ l_1(x,x) & \to & x & N_p \\ l_1(l_1(x,y),z) & \to & l_1(x,z) & M_p^{\mathrm{f}} \\ l_1(x,l_1(y,z)) & \to & l_1(x,z) & M_p^{\mathrm{t}} \end{array}$$

To each DAG we can obtain its canonical tree by undoing the sharing of subdags. Using a recursive path ordering [7, 4], it is straightforward to see that application of these rules must terminate on these trees.

If the rules are applied on DAGs, observe that each rewrite of the DAG corresponds to one or more rewrites of the canonical tree. So, rewriting the DAG must also terminate.

Repeated application of the join operator must also terminate, as the number of nodes is strictly decreasing. The application of the Join operator does not change the canonical tree of a BDD. Therefore, it does not enable more rewrite steps to be applied. Hence, repeated application of all operators must terminate. □

Notation 4.12. Let B be a BDD and let C be a reduced BDD such that $B \cong C$. According to Theorem 4.10 C is unique up to an isomorphism. According to Theorem 4.11 C must exist, and can be effectively obtained. This allows us to write $R(B)$ for C.

Note that Theorem 4.10 implies that $R(B_\phi) = B_{\mathrm{t}}$ if ϕ is strongly equivalent to a tautology. Note also that $R(B_\phi) = B_{\mathrm{f}}$ if ϕ is strongly equivalent to a contradiction. This observation is the basis for using BDDs for propositional logic. Contrary to what is stated in [2], due to the different setting it is not the case that $R(B_\phi)$ is the smallest representation for ϕ. This is due to the particular construction of conjunction on BDDs and the sorting operator that can cause BDDs to grow.

5 Advanced operations on BDDs

In this section we present two operators on BDDs that are solely defined for predicate logic. The first one is a *copying operator* $C(B)$ that puts B in conjunction with a copy of itself, where variables are made fresh.

The second operator is the *unification operator* $U_\zeta(B)$ where ζ is a so-called *relevant* unifier. $U_\zeta(B)$ instantiates B according to ζ.

Definition 5.1. Let B be a BDD in which variables $\vec{x}$ occur. The *copy operator* $C(B)$ is defined as

$$C(B) = B \wedge B[\vec{x} := \vec{x_1}]$$

where $\vec{x_1}$ is a sequence of variables not occurring in B.

Definition 5.2. Let $P(t_1, \ldots, t_n), Q(u_1, \ldots, u_m) \in \mathbb{P}(Pr, F, V)$. A substitution $\zeta : V \to \mathbb{T}(F, V)$ is called a *unifier* of $P(t_1, \ldots, t_n)$ and $Q(u_1, \ldots, u_m)$ iff $\zeta(P(t_1, \ldots, t_n)) = \zeta(Q(u_1, \ldots, u_m))$. A unifier ζ of $P(t_1, \ldots, t_n)$ and $Q(u_1, \ldots, u_m)$ is called *most general* iff for each unifier ζ' of $P(t_1, \ldots, t_n)$ and $Q(u_1, \ldots, u_m)$ there is a substitution ξ such that $\xi \circ \zeta = \zeta'$.

If predicates $P(t_1, \ldots, t_n)$ and $Q(u_1, \ldots, u_m)$ are unifiable, then they have a most general unifier (MGU), which is unique modulo renaming of variables. There is also an MGU that is idempotent, i.e. $\zeta(\zeta(x)) = x$. Moreover, the MGU can be determined in linear time [9, 10].

Definition 5.3. Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD and let ζ be a substitution. The BDD $\zeta(B)$ is defined as:

$$\zeta(B) = (Q, \lambda x.\zeta(l(x)), \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1).$$

In the following definition we define *relevant unifiers*. For arbitrary BDDs this definition is rather unattractive. But, as Lemma 5.5 helps us to see, relevant unifiers are easy to find in reduced BDDs.

Definition 5.4. Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a BDD. A node $p \in Q$ is called *redundant* iff $p{\downarrow_{\mathfrak{t}}} \approxeq p{\downarrow_{\mathfrak{f}}}$.
A path

$$p_0 \xrightarrow{\diamond_0} p_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{n-1}} p_n \xrightarrow{\diamond_n} \ell$$

for $\ell \in \{0, 1\}$ is called *allowed* iff there are no $0 \leq i < j \leq n$ such that $l(p_i) = l(p_j)$ and $\diamond_i \neq \diamond_j$.
A node $p \in Q$ is called *truth-truth capable* iff there is an allowed path

$$p \xrightarrow{\mathfrak{t}} p_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{n-1}} p_n \xrightarrow{\diamond_n} 1$$

A valuation ζ is called a *relevant unifier for* B iff there is a path

$$p_0 \xrightarrow{\diamond_0} p_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{n-1}} p_n \xrightarrow{\diamond_n} 1$$

with $p_0 = s$ and for all $0 \leq i \leq n$ it holds that p_i is not redundant, if $\diamond_i = \mathfrak{f}$, p_i is not truth-truth capable, and for some $0 \leq i, j \leq n$ $\diamond_i = \mathfrak{f}$, $\diamond_j = \mathfrak{t}$ and ζ is an idempotent most general unifier of $l(p_i)$ and $l(p_j)$.

Lemma 5.5. *Let B be a reduced BDD.*

- *There are no redundant nodes $p \in Q_B$.*
- *Every path in B is allowed.*
- *If p_i is not truth truth capable, $p_i{\downarrow_{\mathfrak{t}}} = 0$.*
- *ζ is a relevant unifier for B iff for some $0 \leq i, j \leq n$ $\diamond_i = \mathfrak{f}$, $\diamond_j = \mathfrak{t}$ and ζ is the most general unifier of $l(p_i)$ and $l(p_j)$ on the rightmost path*

  $$p_0 \xrightarrow{\diamond_0} p_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{n-1}} p_n \xrightarrow{\diamond_n} 1$$

  of B.

Proof.

- Suppose B is reduced and there is a redundant node $p \in Q_B$. Hence, $p{\downarrow_{\mathfrak{t}}} \approxeq p{\downarrow_{\mathfrak{f}}}$. According to Lemma 4.8.4 $p{\downarrow_{\mathfrak{t}}} = p{\downarrow_{\mathfrak{f}}}$. Hence, the neglect operator is applicable, contradicting that B is reduced.
- If a path in B would not be allowed, the sort and/or merge operators are applicable, contradicting that B is reduced.
- If p_i is not truth truth capable, every path starting with $p{\downarrow_{\mathfrak{t}}}$ ends in 0. Clearly, if $p_i{\downarrow_{\mathfrak{t}}} \neq 0$ the neglect operator is applicable at the one but last node on a path starting in $p{\downarrow_{\mathfrak{t}}}$. This contradicts that B is reduced.
- We can only turn left on a non truth truth preserving node on the path where we search for relevant unifiers. According to the previous items in this way we walk along the rightmost path of B to 1.

□

It is obvious from this characterisation that relevant unifiers are easy to find as we only need to inspect the rightmost path of B. For instance in the BDD at the left of Figure 7 on page 17 the only relevant unifier on the rightmost path to 1 is $y{:=}a$. And in the BDD in Figure 10 on page 20 the only relevant unifier on the rightmost path is $y{:=}d$.

Definition 5.6. Let B be a BDD. The *unification operator* $U_\zeta(B)$ is defined by

$$U_\zeta(B) = \zeta(B) \text{ if } \zeta \text{ is a relevant unifier of } B.$$

Note that if ζ is a relevant unifier, then $U_\zeta(B)$ contains strictly less variables than B.

Lemma 5.7. *(Soundness). Let B be a BDD.*

- $B \approx C(B)$.
- $B \approx B \wedge U_\zeta(B)$.

Proof. Easy logical consequence. □

6 Completeness

In this section we show that if a formula ϕ is unsatisfiable, then there is a sequence of operators on B_ϕ that turns it into $B_{\mathfrak{f}}$. The first lemma attracts attention to rightmost paths in BDDs for calculating relevant unifiers. The next lemma shows that if B_ϕ is strongly equivalent to $B_{\mathfrak{f}}$ then we can find it by repeatedly applying relevant unifiers on B_ϕ. Theorem 6.3 says, using Herbrand's theorem that if B_ϕ is unsatisfiable we must apply relevant unifiers to a certain number of copies to B, all interleaved with reduction operators. The algorithm in the next section is nothing more than recursively searching for this sequence of operators.

Lemma 6.1. *Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a reduced BDD and ξ a closed substitution such that $\xi(B) \approxeq B_{\mathfrak{f}}$. If there is a rightmost (allowed) path*

$$q_0 \xrightarrow{\diamond_0} q_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{m-1}} q_m \xrightarrow{\diamond_m} 1$$

with $q_0 = B{\uparrow}$, then there are $0 \leq i, j \leq m$ with $\diamond_i = \mathfrak{f}$, $\diamond_j = \mathfrak{t}$ and $\xi(l(q_i)) = \xi(l(q_j))$.

Proof. Let

$$s = q_0 \xrightarrow{\diamond_0} q_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{m-1}} q_m \xrightarrow{\diamond_m} 1$$

be the rightmost (allowed) path in B. Suppose there are no $0 \leq i, j \leq m$ with $\diamond_i = \mathfrak{f}$, $\diamond_j = \mathfrak{t}$ and $\xi(l(q_i)) = \xi(l(q_j))$. Then, we can construct a Herbrand structure $\mathfrak{A}_H$ such that $\mathfrak{A}_H, \xi \models B$.

Define the relations in $\mathfrak{A}_H$ such that for each node q_i:

$$[\![l(q_i)]\!]_\xi^{\mathfrak{A}_H} = \begin{cases} 0 & \text{if } \diamond_i = \mathfrak{f} \\ 1 & \text{if } \diamond_i = \mathfrak{t} \end{cases}$$

and take $[\![l(q_i)]\!]_\xi^{\mathfrak{A}_H} = 0$ elsewhere.

As for $1 \leq i, j \leq m$ $\diamond_i = \mathfrak{f}$, $\diamond_j = \mathfrak{t}$, implies $\xi(l(q_i)) \neq \xi(l(q_j))$, the definition of $\mathfrak{A}_H$ is indeed correct. Now it is trivial to check that $\mathfrak{A}_H, \xi \models B$, contradicting that $\xi(B) \cong B_\mathfrak{f}$, as $\mathfrak{A}_H, \xi \not\models B_\mathfrak{f}$. □

Lemma 6.2. *Let $B = (Q, l, \xrightarrow{\mathfrak{f}}, \xrightarrow{\mathfrak{t}}, s, 0, 1)$ be a reduced BDD and ξ a substitution such that $\xi(B) \cong B_\mathfrak{f}$. Then there is a sequence of relevant unifiers $\zeta_1, \ldots, \zeta_n$ such that*

$$R(U_{\zeta_1}(R(U_{\zeta_2}(\ldots(R(U_{\zeta_n}(B)))))))) = B_\mathfrak{f}.$$

Moreover, n is smaller or equal than the number of variables in B.

Proof. We apply induction on the number of variables that occur in B. Note that if there are no variables in B, $\xi(B) = B$, and therefore, $B \cong B_\mathfrak{f}$. Hence, as B is reduced, $B = B_\mathfrak{f}$. So, we can take the sequence of relevant unifiers to be empty.

If there are $k > 0$ variables in B, we know there is an allowed path to 1 in B, as otherwise $B \cong B_\mathfrak{f}$, and using the same reasoning as above, we take the sequence of relevant unifiers to be empty. As there is an allowed path to 1 in B then according to Lemma 6.1 there is also a rightmost allowed path

$$q_0 \xrightarrow{\diamond_0} q_1 \xrightarrow{\diamond_1} \cdots \xrightarrow{\diamond_{m-1}} q_m \xrightarrow{\diamond_m} 1$$

to 1 in B with $q_0 = B\uparrow$, and some $0 \leq i, j \leq m$ with $\diamond_i = \mathfrak{f}$, $\diamond_j = \mathfrak{t}$ and $\xi(l(q_i)) = \xi(l(q_j))$. Hence, there is a relevant unifier ζ such that $\zeta(l(p_i)) = \zeta(l(p_j))$. Note that in $U_\zeta(B)$ there are strictly less variables than in B. As the operator R does not introduce new labels of nodes, $R(U_\zeta(B))$ also contains strictly less variables than B. Moreover, as, due to the fact that ζ is an mgu, there is some substitution ξ' such that $\xi' \circ \zeta = \xi$. So, $\xi'(R(U_\zeta(B))) \cong B_\mathfrak{f}$. Furthermore, $R(U_\zeta(B))$ is reduced. Now using the induction hypothesis, there must be a sequence $\zeta_1, \ldots, \zeta_n$ such that

$$R(U_{\zeta_1}(\ldots(R(U_{\zeta_n}(R(U_\zeta(B)))))))) = B_\mathfrak{f}.$$

So, $\zeta_1, \ldots, \zeta_n, \zeta$ is the required sequence. □

Theorem 6.3. *Let B be an unsatisfiable BDD. Then*

$$R(U_{\zeta_1}(\ldots R(U_{\zeta_n}(\overbrace{R(C(\ldots R(C}^{k \text{ times}}(B)))))))) = B_\mathfrak{f}$$

for certain $n, k \geq 0$ and relevant unifiers $\zeta_1, \ldots, \zeta_n$.

Proof. As B is unsatisfiable, it follows from Theorem 2.5 and Theorem 3.6 that there are closed substitutions $\xi_1, \ldots, \xi_m$ such that

$$\bigwedge_{i=1}^{m} \xi_i(B) \cong B_\mathfrak{f}.$$

Select some k such that $2^k \geq m$. The term

$$(\xi_1 \circ \iota[\vec{x_1}{:=}\vec{x}]) \ldots (\xi_m \circ \iota[\vec{x_m}{:=}\vec{x}])(\overbrace{C(C(\ldots(C(B))\ldots))}^{k \text{ times}}) \approxeq B_{\mathfrak{f}}. \quad (1)$$

Here, the notation $\iota[\vec{x_i}{:=}\vec{x}]$ is a renaming that takes care that the substitution ξ_i operates on the appropriate variables. Write $\xi = (\xi_1 \circ \iota[\vec{x_1}{:=}\vec{x}]) \ldots (\xi_m \circ \iota[\vec{x_m}{:=}\vec{x}])$. According to Lemma 4.5 $R(B) \approxeq B$. So, we may interleave the copying operators with simple reduction operators without changing strong equivalence. Write $B' = \overbrace{R(C(\ldots(R(C(B))\ldots)))}^{k \text{ times}}$. Rephrasing (1) yields

$$\xi(B') \approxeq B_{\mathfrak{f}}.$$

By Lemma 6.2 it follows that there are relevant unifiers $\zeta_1, \ldots, \zeta_m$ such that

$$R(U_{\zeta_1}(\ldots(R(U_{\zeta_m}(B'))))) = B_{\mathfrak{f}}.$$

which is exactly what we must show. □

7 Algorithm

The previous lemmata suggest the following algorithm to find out whether a formula ϕ is unsatisfiable.

```
Solve(φ) =
   B: = R(B_φ)
   Repeat
      TryToReduce(B)
      B: = R(C(B))
   Endrepeat

TryToReduce(B) =
   If B = B_f, report 'unsatisfiable' and stop
   For all relevant unifiers ζ of B TryToReduce(R(U_ζ(B)))
```

It says that first $R(B_\phi)$ should be constructed. As is shown in [1, 2] this can be done efficiently (although for certain formulas the width of the BDD representation may blow up. The depth is linearly bounded by the number of different predicates). Note that if the construction is carried out as described in [1, 2] the expensive sorting operator is not applied.

Then, recursively, relevant unifiers are applied to B in `TryToReduce`(B). Finding the relevant unifiers can be done efficiently. All pairs of p_i, p_j of predicates labelling the rightmost path σ of B, with $p_i{\downarrow_{\mathfrak{t}}} = 0$ and $p_j{\downarrow_{\mathfrak{t}}} \neq 0$ must be examined. Hence, if the length of σ is l, there are at most $\frac{1}{4}l^2$ potentially unifiable predicates. Using the algorithms proposed by [9, 10] unifiers can be found linearly in the size of the terms.

Application of the unifier, calculation of $R(U_\zeta(B))$ may be costly. It is linear to calculate $U_\zeta(B)$. But this may destruct the ordering of the labels. When reducing, it may be necessary that the costly sorting operators are applied. An attempt can be made to avoid extensive sorting by grouping predicates with the same predicate symbol together.

Also the recursive nesting of calls to `TryToReduce` could be a cause of inefficiency. However, at each call at least one variable is instantiated. Therefore, the depth of recursive calls to TryToReduce is limited by the number of free variables in the BDD.

When `TryToReduce(B)` does not yield $B_{\mathfrak{t}}$, then a copy must be made, i.e. the command $B{:=}R(C(B))$ is carried out. The copy operator is defined using the $\wedge$ and hence, using the techniques in [1, 2] $R(C(B))$ can be calculated efficiently.

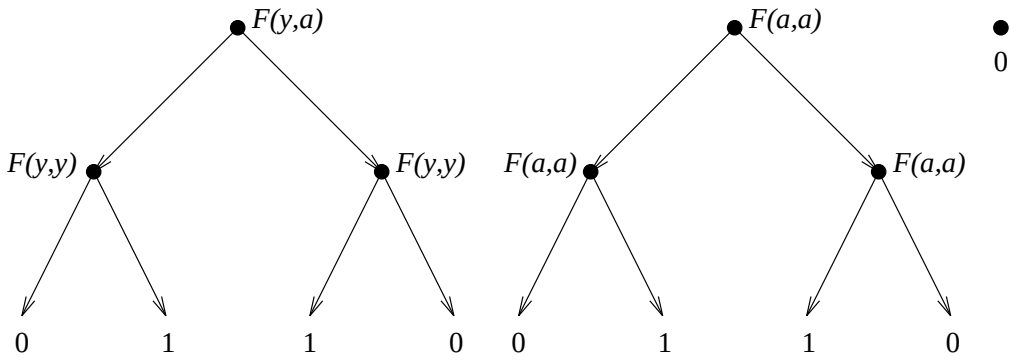

Figure 7: Russel's paradox

So, the programme is clearly browsing through larger and larger BDDs of the form

$$R(U_{\zeta_1}(R(U_{\zeta_2}(\ldots R(C(\ldots R(C(B_\phi)))))))) \tag{2}$$

where we stop if this BDD appears to be $B_{\mathfrak{f}}$. If ϕ is unsatisfiable, this algorithm must terminate according to Theorem 6.3. Moreover, if we find that (2) is equal to $B_{\mathfrak{f}}$ then we may conclude with Lemmas 4.5 and 5.7 that $B_\phi \approx B_\phi \wedge B_{\mathfrak{f}}$, which means that B_ϕ must be unsatisfiable.

Note that the algorithm presented here only sketches a basic approach on which a number of improvements are possible. First, it is sometimes possible to identify that a formula is satisfiable in an early phase of the protocol. This for instance seems to happen if finding unifiers fails for other reasons than the occur check. It is also the case that sometimes redundant unifiers are calculated in the approach above, for instance unifiers that undo a copying step. In [6] it has been described how to avoid some of the computational overhead.

8 Examples

In this section we apply the proposed method to three examples taken from [11]. The first one is chosen for its simplicity, while it still expresses an interesting fact. The second one is chosen because it needs copying and the third one is interesting because it is rated as reasonably difficult, while it is still small enough to carry out the construction of the BDD and the calculation of relevant unifiers by hand.

8.1 Russel's paradox

Problem 39 of [11] says that 'there is no Russell set', i.e. a set which contains exactly those sets which are not members of themselves. The predicate $F(x, y)$ must be read as 'x is a member of y'. The problem is originally stated as

$$\neg\exists x\forall y(F(y, x) \leftrightarrow \neg F(y, y)). \tag{3}$$

We negate and Skolemise the formula. After removal of the remaining universal quantifier we obtain

$$F(y, a) \leftrightarrow \neg F(y, y) \tag{4}$$

where a is a Skolem constant. The leftmost BDD in Figure 7 is obtained from this formula. There is one relevant unifier, which is obtained by making a rightmost walk through the BDD to an endnode 1. On this path the predicates $F(y, a)$ and $F(y, y)$ are unifiable, with unifier $\zeta(y) = a$. Applying the

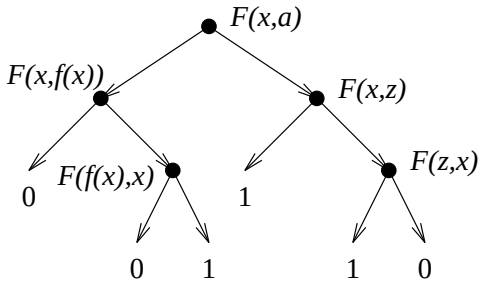

Figure 8: There are no circular sets

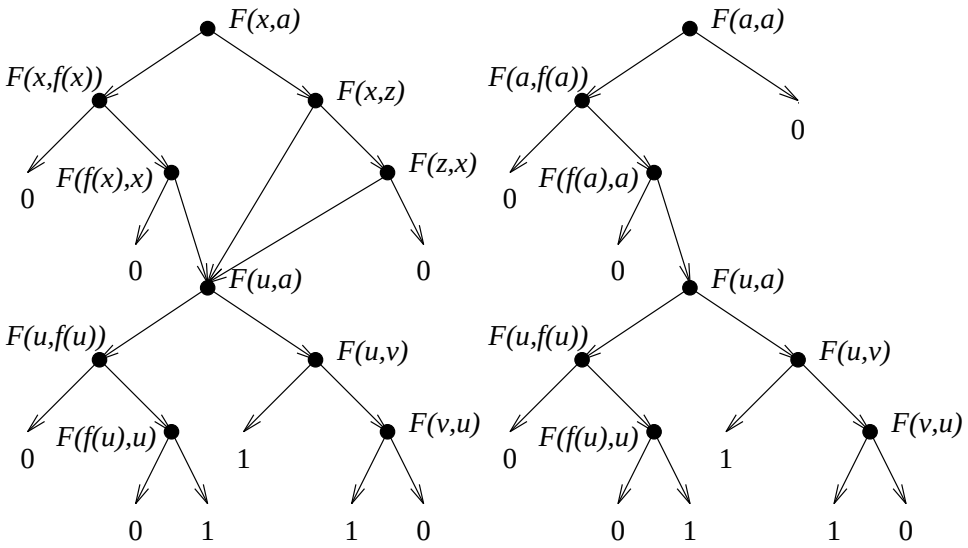

Figure 9: There are no circular sets (continued)

unifier to this BDD yields the BDD in the middle of Figure 7. Reduction leads to B_{f} (at the right in Figure 7) showing that (4) is unsatisfiable, and hence (3) a tautology.

8.2 There are no circular sets

This second example is problem 42 of [11]. It says that 'a set is circular if it is a member of another set, which in turn is a member of the original'. We show that there is no set containing all non circular sets. The original formulation of this theorem is

$$\neg\exists y\forall x(F(x,y) \leftrightarrow \neg\exists z(F(x,z) \wedge F(z,x))). \tag{5}$$

Negation, Skolemisation and removal of quantifiers yields

$$(F(x,a) \rightarrow \neg(F(x,z) \wedge F(z,x))) \wedge (\neg(F(x,f(x)) \wedge F(f(x),x)) \rightarrow F(x,a)). \tag{6}$$

Obviously a and f are Skolem functions. Note that the structure of the formula has changed somewhat due to Skolemisation, as in (5) there is a quantifier in the scope of a $\leftrightarrow$. In Figure 8 the BDD for (6) is depicted. There are two relevant unifiers, the first one mapping z and x to a, and the second one

mapping x to z. Application of the first unifier leads to a BDD B that neither is equal to $B_\mathfrak{f}$ nor has another relevant unifier. Application of the second unifier leads to a subsequent unifier, mapping z to a. In effect application of this unifier leads again to the BDD B. In this way the BDD $B_\mathfrak{f}$ cannot be obtained.

According to the algorithm we now must apply the copying operator. This leads to the BDD at the left of Figure 9. We have used fresh variables u and v for respectively x and z. Along a rightmost walk in this tree we obtain the following 6 unifiers.

$$\left\{ \begin{array}{l} x{:=}a \\ z{:=}a \end{array} \right. \qquad \left\{ \begin{array}{l} x{:=}v \\ u{:=}a \end{array} \right. \qquad \left\{ \; z{:=}x \right.$$

$$\left\{ \begin{array}{l} u{:=}a \\ v{:=}a \end{array} \right. \qquad \left\{ \begin{array}{l} u{:=}z \\ x{:=}a \end{array} \right. \qquad \left\{ \; v{:=}u \right.$$

If we apply the first one to the BDD we obtain the BDD at the righthandside of Figure 9. Now we find the following unifiers along the rightmost walk.

$$\left\{ \begin{array}{l} u{:=}f(a) \\ v{:=}a \end{array} \right. \qquad \left\{ \begin{array}{l} u{:=}f(a) \\ v{:=}a \end{array} \right. \qquad \left\{ \; u{:=}a \right.$$

$$\left\{ \begin{array}{l} u{:=}a \\ v{:=}a \end{array} \right. \qquad \left\{ \; u{:=}v \right.$$

Applying the first unifier yields a BDD that reduces to $B_\mathfrak{f}$. So, formula (6) is unsatisfiable and hence, formula (5) is a contradiction.

8.3 A problem for monadic logic

In this example we apply our BDD techniques to problem 28, which is rated among the hardest in [11]. There are a few problems that rated as more difficult (i.e. problems 34, 47, 69 and 70), which due to their length seem somewhat too large to do by hand. The formulation of the hardest problem among the 'more tedious monadic logic problems from Kalish and Montague' is formulated as follows. Note that there is a small mistake in its formulation in [11], which is mentioned in the Errata belonging to it.

$$\begin{array}{l} ((\forall x\,(P(x) \rightarrow \forall x\,Q(x)))\wedge \\ \;(\forall x\,(Q(x) \vee R(x)) \rightarrow \exists x\,(Q(x) \wedge S(x)))\wedge \\ \;(\exists x\,S(x) \rightarrow \forall x\,(F(x) \rightarrow G(x)))) \rightarrow \\ \forall x\,(P(x) \wedge F(x) \rightarrow G(x)) \end{array} \tag{7}$$

By denying and Skolemising the formula, we obtain (a, b, c and d are Skolem constants and w, x, y and z are variables):

$$\begin{array}{l} (P(w) \rightarrow Q(x))\wedge \\ ((Q(b) \vee R(b)) \rightarrow (Q(c) \wedge S(c)))\wedge \\ (S(z) \rightarrow (F(y) \rightarrow G(y)))\wedge \\ (P(d) \wedge F(d) \wedge \neg G(d)) \end{array} \tag{8}$$

The BDD of this formula is depicted in Figure 10. Its labels are sorted alphabetically. It has 35 nodes. The BDD has only one relevant unifier, $y{:=}d$. Application of this unifier reduces the size of the BDD with almost 50%. The newly obtained BDD is depicted in Figure 11 at the left. Again, it has only one relevant unifier, being $z{:=}c$. The BDD resulting after application of this unifier has been depicted at the righthandside of Figure 11. This last BDD has two relevant unifiers, $x{:=}b$ and $x{:=}c$. After applying either of those it we obtain the BDD in Figure 12. This BDD has the unique relevant unifier $w{:=}d$. Application of this last unifier yields the BDD $B_\mathfrak{f}$, showing (8) unsatisfiable, and hence (7) a tautology.

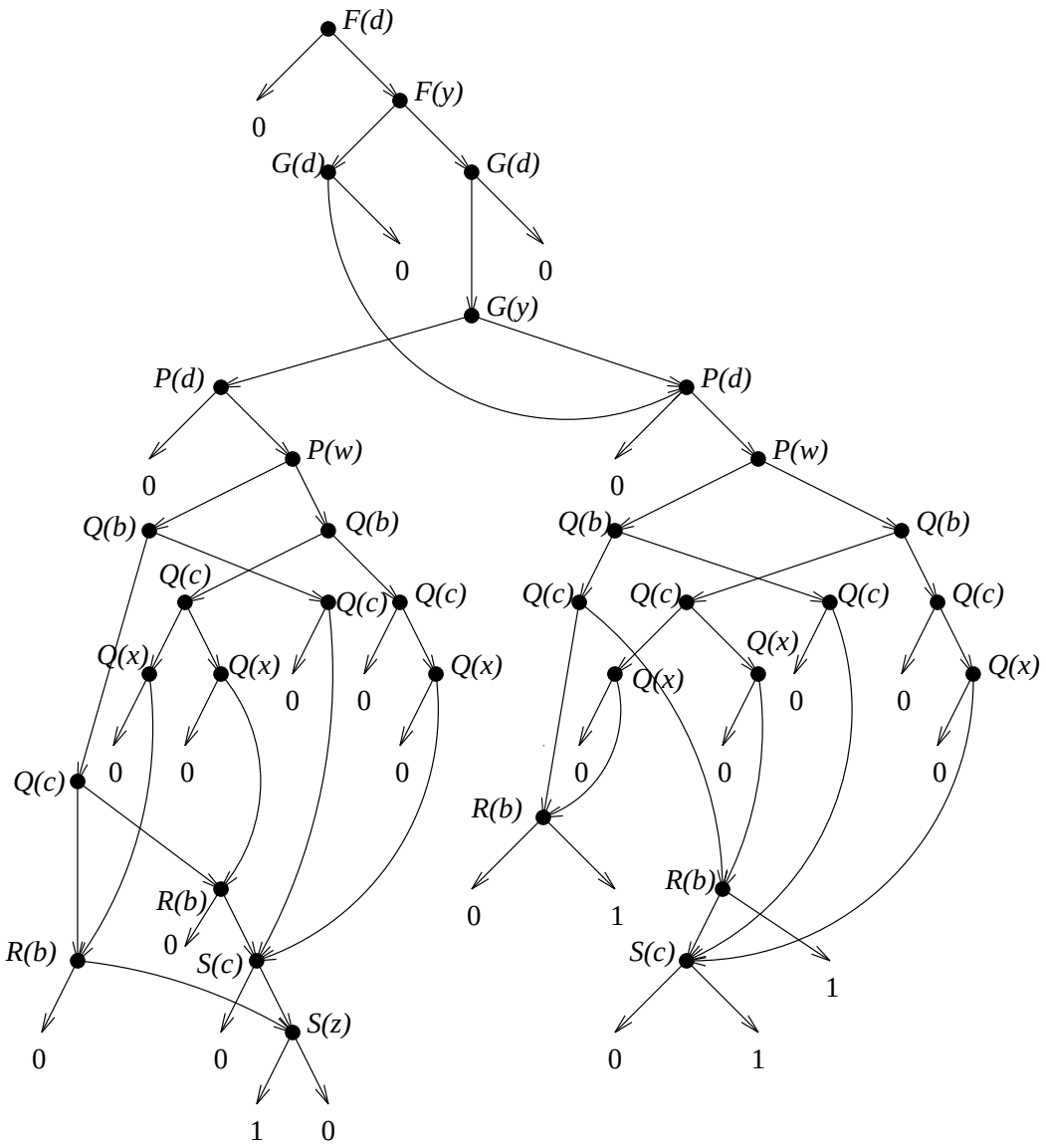

Figure 10: A tedious problem from monadic logic

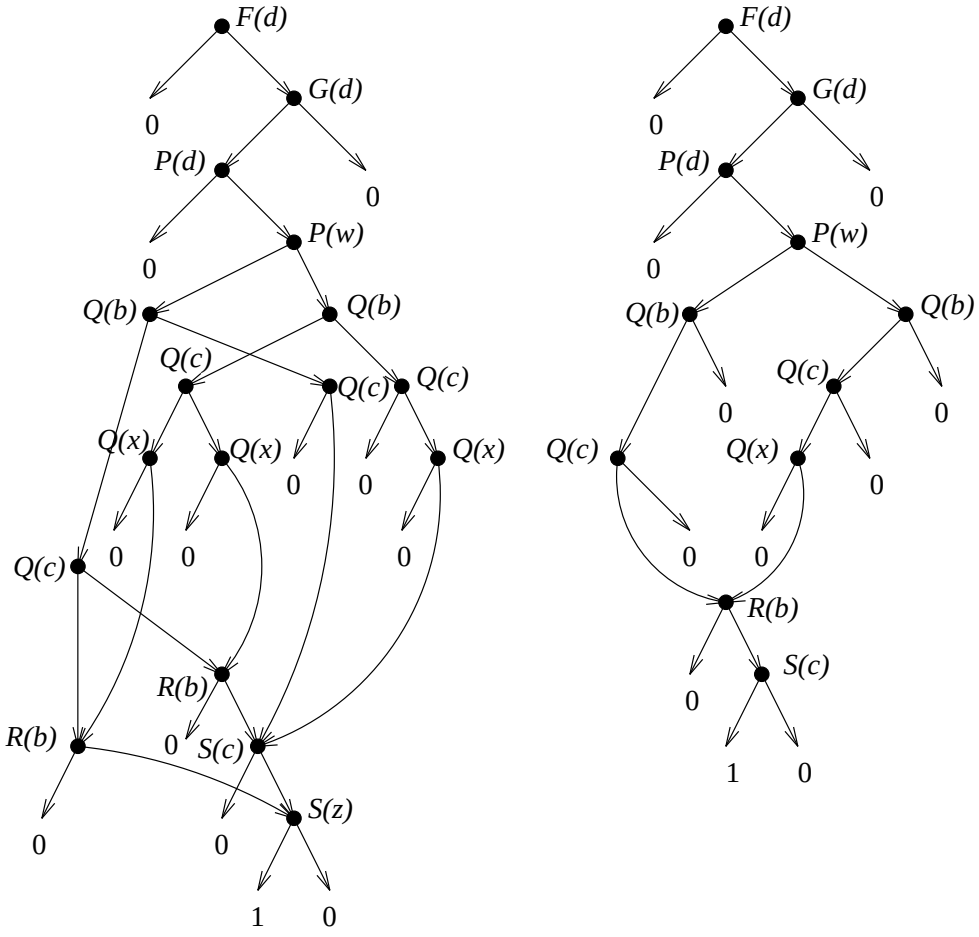

Figure 11: A tedious problem from monadic logic (continued I)

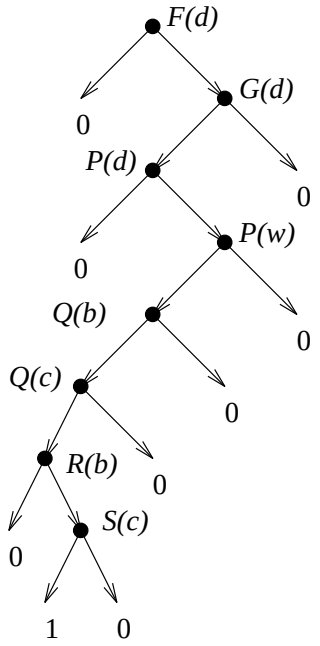

Figure 12: A tedious problem from monadic logic (continued II)

9 Conclusions and related work

We have designed a way to apply BDDs to predicate logic. We have shown that this yields a complete proof procedure and we have also shown that for the examples that we have considered the technique leads so quickly to results that they can be carried out by hand. We have not implemented the system and hence not tried to use it on larger examples. As experience shows (which is the best that is available due to lack of a theoretical foundation) the usefulness of techniques only show itself in application. It appears to be an art to get automatic theorem provers to prove sizable problems. We can only wait what application of these ideas will bring, and hope that the triumph stories that BDDs brought to us, will be repeated in the setting of predicate logic.

Independently of the work reported in here two other groups have been working on extending BDDs to predicate logic [5, 6, 12, 13] in a rather similar way, probably indicating the naturalness of the approach.

In [12, 13] Joachim Posegga reports about an approach where BDDs are constructed without sorting their labels. In order to reduce the overhead caused by copying BDDs he indicates subBDDs as logical entities. These subBDDs stand for universally quantified subformulas; when copies of them are used during the proof search, only a BDD for the scope of the formula is inserted in the surrounding BDD. These ideas have been implemented by transforming a BDD into a PROLOG programme. The PROLOG programme takes care of finding the unifiers, that in this case are found on the leftmost path (instead of on the rightmost path as has been done here). The implemented system is called SHARE and is available by contacting the author [14]. It is interesting to know that SHARE solved problem 1 to 46 of Pelletier [11] without mentionable problems (see [12]).

In [5, 6] a system is described operating in a very similar way. Here stress is put on determining optimal unifiers. In particular the copying operator C creates many pairs of unifiable labels, that need not be considered. Moreover, using a smart weight function certain unifiers get priority above others, which according to [6] very often selects the correct unifiers. A trial implementation exist that could

easily solve all problems of Pelletier except a few using equality. This is due to the fact that equality is encoded using the standard equality axioms, instead of via special features found elsewhere.

References

[1] K.S. Brace, R.L. Rudell and R.E. Bryant. Efficient implementation of a BDD package. 27th ACM IEEE Design Automation Conference 1990. Pages 40-45.

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

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[4] N. Dershowitz. Termination of rewriting. Journal of Symbolic Computation, 3(1):69–116, 1987.

[5] J. Goubault. Syntax independent connections. In D. Basin, Bertram Fronhöfer, Reiner Hähnle, J. Posegga and C. Schwind, editors, Proceedings of the 2nd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, 1993.

[6] J. Goubault. Proving with BDDs and control of information. Unpublished manuscript, 1995.

[7] J.W. Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science*, pages 1-116, volume 2. Oxford Science Publications, 1992.

[8] K.L. McMillan and J. Schwalbe. Formal verification of the Gigamax cache consistency protocol. In *International Symposium on Shared Memory Multiprocessing*, 1991.

[9] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, pages 258-282, Vol. 4, No. 2, 1982.

[10] M.S. Paterson and M.N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16:158–167, 1978.

[11] F.J. Pelletier. Seventy-Five Problems for Testing Automated Theorem Provers. *Journal of Automated Reasoning.* Volume 2, pp. 191-216, 1986. See also Errata. *Journal of Automated Reasoning.* Volume 4, pp. 235-236, 1988.

[12] J. Posegga and P.H. Schmitt. Automated Deduction with Shannon Graphs. *Journal of Logic and Computation.* In print. 1995.

[13] J. Posegga and K. Schneider. Deduction with First-order BDDs. In D. Basin, Bertram Fronhöfer, Reiner Hähnle, J. Posegga and C. Schwind, editors, Proceedings of the 2nd Workshop on Theorem Proving with Analytic Tableaux and Related Methods, 1993.

[14] J. Posegga. SHAnnon graph REfutation system. This is an implementation of a BDD based theorem prover. Information: posegga@ira.uka.de, 1995.

[15] A. Urquhart. Hard Examples for Resolution. *Journal of the Association for Computing Machinery.* 24(1):209-219, 1987.

[16] A. Urquhart. Complexity of proofs in classical propositional logic. In Y. Moschovakis, editor, *Logic from Computer Science*, Springer-Verlag, pp. 596-608, 1992.