Relational Approach to Boolean Logic Problems

Rudolf Berghammer and Ulf Milanese

Institut für Informatik und Praktische Mathematik
Universität Kiel, Olshausenstraße 40, D-24098 Kiel

Abstract. We present a method for specifying and implementing algorithms for Boolean logic problems. It is formally grounded in relational algebra. Specifications are written in first-order set theory and then transformed systematically into relation-algebraic forms which can be executed directly in RelView, a computer system for the manipulation of relations and relational programming. Our method yields programs that are correct by construction. It is illustrated by some examples.

1 Introduction

For many years, relational algebra (see [15, 13]) has been used successfully for formal problem specification, prototyping, and program development. This is mainly due to the fact that many fundamental structures and datatypes of discrete mathematics and computer science can be modeled naturally by relations and, hence, many computations on them can be reduced to relation-algebraic computations. Boolean logic is also a fundamental system of discrete mathematics and computer science. Of course, here relations are used, for instance, the satisfiability relation $\models$ between assignments and formulae. But when having a look through relevant literature, *relational algebra* in the sense of [15, 13] does not seem to play a role. Especially in the case of algorithmic problems, binary decision diagrams (BDDs; see [3]) and so-called BDD-packages (like BuDDy [10] and CUDD [14]) are currently the standard tools.

We present a relational approach to Boolean logic problems and illustrate its usefulness by solving some algorithmic problems. The design of the algorithms starts from mathematical problem specifications, which are based on a relational model of Boolean formulae in conjunctive normal form (CNF). With the aid of rigorous calculations the specifications are transformed gradually into relation-algebraic descriptions which, essentially, are given by expressions of relational algebra. Finally, the latter are translated into the language of the computer system RelView (see [2]) and then can be executed directly. In this way programs are built up very quickly and their correctness is guaranteed by construction.

RelView also uses (reduced ordered) BDDs to implement relations in a very efficient way. Details are presented in [9, 11]. Hence, our programs can be seen as BDD-algorithms at a very high level. We found their development relatively easy. Based on the experience made during the development of the RelView system, however, we think that programming at the level of BDD-packages is too

low-level to be effective. It is error prone and the rather complex BDD implementation, in particular the use of shared pointer structures with dynamically allocated and freed cells, makes these errors difficult to track down. Errors are often observable only after a long system operation time or under high load and so refrain from being detected by standard test scenarios.

The remainder of the paper is organized as follows. Section 2 collects some relational preliminaries. In Section 3 we introduce a relational model of CNF-formulae and develop algorithms for computing all satisfying assignments, solving the MAX-SAT problem, and testing the independence of CNF-formulae from variables. In Section 4 we explain how these algorithms can be implemented in the RELVIEW tool, present a small example, and report on the results of our practical experiments. Section 5 contains some concluding remarks.

2 Relational Preliminaries

We write $R : X \leftrightarrow Y$ if R is a relation with domain X and range Y, i.e., a subset of $X \times Y$. If the sets of R's *type* $X \leftrightarrow Y$ are finite, we may consider R as a Boolean matrix. Since this interpretation is well suited for many purposes, in the following we often use matrix terminology and matrix notation. Especially, we write R_{xy} instead of $(x, y) \in R$. We assume the reader to be familiar with the basic operations R^T (*transposition*), $\overline{R}$ (*complement*), $R \cup S$ (*union*), $R \cap S$ (*intersection*), $R\,S$ (*composition*), $R \subseteq S$ (*inclusion*), and the special relations O (*empty relation*), L (*universal relation*), and I (*identity relation*).

There are some relation-algebraic possibilities to model sets. Our first modeling uses *vectors*, which are relations v with $v = v\mathsf{L}$. Since for a vector the range is irrelevant we consider mostly vectors $v : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\bot\}$ as range and omit in such cases the second subscript, i.e., write v_x instead of $v_{x\bot}$. Such a vector can be considered as a Boolean matrix with exactly one column, i.e., as a Boolean column vector, and *represents* the subset $\{x \in X \mid v_x\}$ of X. In [13] it is shown that if $R : X \leftrightarrow X$ is a quasi-order and $v : X \leftrightarrow \mathbf{1}$ represents a subset Y of X, then the set of *greatest elements* of Y with respect to R is represented by the vector $\mathit{GreEl}(R, v) := v \cap \overline{\overline{R}^\mathsf{T} v} : X \leftrightarrow \mathbf{1}$.

A vector is a *point* if it is non-empty and injective. This means that it represents a singleton subset of its domain or an element from it if we identify a singleton set with the only element it contains. In the matrix model a point $v : X \leftrightarrow \mathbf{1}$ is a Boolean column vector in which exactly one component is true.

We will also use injective mappings for modeling subsets. Given an injective mapping as relation $\imath : Y \leftrightarrow X$, we may consider Y as a subset of X by identifying it with its image under $\imath$. If Y is actually a subset of X and $\imath$ is the identity mapping from Y to X, then the vector $\imath^\mathsf{T}\mathsf{L} : X \leftrightarrow \mathbf{1}$ represents Y as subset of X in the sense above. Clearly, the transition in the other direction is also possible, i.e., the generation of an injective mapping $\mathit{inj}(v) : Y \leftrightarrow X$ fulfilling $\mathit{inj}(v)_{yx}$ iff $y = x$ from a given vector $v : X \leftrightarrow \mathbf{1}$ representing the subset Y of X.

As a third possibility to model subsets of a given set X we will use the set-theoretic *membership relation* $\varepsilon : X \leftrightarrow 2^X$, defined by ε_{xY} iff $x \in Y$. Such

relations lead to a column-wise representation of sets of subsets. If $v : 2^X \leftrightarrow \mathbf{1}$ represents a subset $\mathfrak{S}$ of 2^X in the sense above, then for all $x \in X$ and $Y \in \mathfrak{S}$ we get the equivalence of $(\varepsilon\, inj(v)^\mathsf{T})_{xY}$ and $x \in Y$. Hence, the elements of $\mathfrak{S}$ are represented precisely by the columns of $\varepsilon\, inj(v)^\mathsf{T} : X \leftrightarrow \mathfrak{S}$. Generally, we say that a subset $\mathfrak{S}$ of 2^X is *represented by the columns* of $R : X \leftrightarrow \mathfrak{S}$ if R_{xY} is equivalent to $x \in Y$ for all $x \in X$ and $Y \in \mathfrak{S}$.

The relation $\varepsilon\, inj(v)^\mathsf{T}$ is a special case of *range restriction*. The general case is given by $R : X \leftrightarrow Y$ and a vector v with domain Y describing a subset Z of Y. Then the range restriction $R\, inj(v)^\mathsf{T} : X \leftrightarrow Z$ of R with respect to v restricts the range Y of R to the subset Z. Using component-wise notation this means that for all $x \in X$ and $z \in Z$ we have R_{xz} iff $(R\, inj(v)^\mathsf{T})_{xz}$.

Reduced ordered BDDs allow a very compact implementation of membership relations. In [9] an implementation of $\varepsilon : X \leftrightarrow 2^X$ is developed with the number of BDD-nodes in $\mathcal{O}(|X|)$. The same holds for the so-called *size-comparison relation* $S : 2^X \leftrightarrow 2^X$, which relates two sets $A, B \in 2^X$ iff $|A| \leq |B|$. Here $\mathcal{O}(|X|^2)$ BDD-nodes suffice for an implementation; see [11] for details.

3 Satisfiability and Related Problems

In this section we show how to model Boolean formulae in conjunctive normal form using relations. Based on this approach, we solve the satisfiability problem SAT, the MAX-SAT problem, and the independence problem for variables.

3.1 A Relational Model for CNF-Formulae

Let $\mathcal{V}$ be a set of Boolean variables. A *literal* over $\mathcal{V}$ is either a variable $x \in \mathcal{V}$ or the negation $\neg x$ of a variable $x \in \mathcal{V}$, and a *clause* over $\mathcal{V}$ is a disjunction $\lambda_1 \vee \ldots \vee \lambda_m$ of $m \geq 1$ literals. In the following we only consider Boolean formulae φ over $\mathcal{V}$ in CNF. This means that φ is of the form $\gamma_1 \wedge \ldots \wedge \gamma_n$ for some $n \geq 1$, where each γ_i, $1 \leq i \leq n$, is a clause over $\mathcal{V}$.

Now, assume $\mathcal{K}^\varphi$ to be the clause set of the CNF-formula φ. Then φ can be modeled by a pair $P^\varphi : \mathcal{V} \leftrightarrow \mathcal{K}^\varphi$ and $N^\varphi : \mathcal{V} \leftrightarrow \mathcal{K}^\varphi$ of relations such that

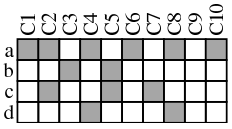
$$P^\varphi_{x\gamma} \;:\Longleftrightarrow\; x \text{ is a literal of } \gamma \qquad\qquad N^\varphi_{x\gamma} \;:\Longleftrightarrow\; \neg x \text{ is a literal of } \gamma\,.$$

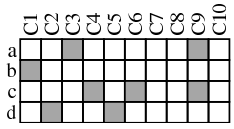
To enhance readability, in the following we omit the superscript φ and write $\mathcal{K}$, P, and N instead of $\mathcal{K}^\varphi$, P^φ, and N^φ.

For illustrative purposes we consider a simple example, taken from [7]. Suppose φ to be the following CNF-formula over the set $\mathcal{V} := \{a, b, c, d\}$.

$$\begin{aligned}&(a \vee \neg b) \wedge (a \vee c \vee \neg d) \wedge (b \vee \neg a) \wedge (a \vee d \vee \neg c)\\ &\wedge\, (b \vee c \vee \neg d) \wedge (a \vee \neg c) \wedge c \wedge (a \vee d) \wedge (\neg a \vee \neg c) \wedge a\end{aligned} \tag{1}$$

In RELVIEW relations can be depicted as Boolean matrices, where it is additionally possible to label rows and columns for explanatory purposes. Using this feature, RELVIEW displays the relations P and N of φ as subsequently shown.





To simplify presentation, in the sequel we always suppose that a clause neither contains a variable and its negation as literals nor contains duplicates of literals. These restrictions are harmless. Under this point of view our modeling consist of two steps. First, we go to a so-called *clause normal form* by neglecting associativity, commutativity, and idempotency of conjunction and disjunction. It represents a CNF-formula as set of clauses, where each clause is divided into the set of positive literals (variables) and negative literals (negated variables). After that, we describe this splitting of the clauses by the relations P and N.

3.2 Computing Satisfying Assignments

Usually, an assignment is defined as a function $a : \mathcal{V} \to \mathbb{B}$ that assigns a Boolean value *tt* of *ff* to every variable. Since there is a 1-1-correspondence between a and the set $\{x \in \mathcal{V} \mid a(x) = tt\}$, in the following we define assignments to be subsets of $\mathcal{V}$. Then an assignment $A \in 2^{\mathcal{V}}$ *satisfies* a variable $x \in \mathcal{V}$, denoted by $A \models x$, iff $x \in A$. The other cases of the relation $\models$ are defined as usual: $A \models \neg\varphi$ iff the relationship $A \models \varphi$ does not hold, $A \models \varphi \vee \psi$ iff $A \models \varphi$ or $A \models \psi$, and $A \models \varphi \wedge \psi$ iff $A \models \varphi$ and $A \models \psi$.

Testing satisfiability of CNF-formulae is one of the paradigmatic NP-complete problems and many algorithms have been developed for it (see e.g., [5, 12] for an overview). For clauses there exists a simple syntactic criterion, viz.:

Theorem 3.1. *Let $A \in 2^{\mathcal{V}}$ be an assignment and γ be a clause over $\mathcal{V}$. Then A satisfies γ iff a literal of γ is a variable from A or the negation of a variable from $\mathcal{V} \setminus A$.*

Proof. Assume the clause γ to be of the form $\lambda_1 \vee \ldots \vee \lambda_m$ for some $m \geq 1$.

For proving "$\Rightarrow$", suppose $A \models \gamma$. Then there exists i, $1 \leq i \leq m$, such that $A \models \lambda_i$. The literal λ_i is either a variable x or a negation $\neg x$ of a variable x. If λ_i equals x, then $x \in A$ follows from $A \models x$, and if λ_i equals $\neg x$, then $x \notin A$ follows from $A \models \neg x$, since the latter is equivalent to $A \not\models x$.

To prove "$\Leftarrow$", assume a variable x with $x \in A$. If there exists i, $1 \leq i \leq m$, such that λ_i equals x, then $x \in A$ shows $A \models \lambda_i$, which in turn implies $A \models \gamma$. Similarly one treats the case of λ_i being of the form $\neg x$ with $x \notin A$. □

Assume the CNF-formula φ over $\mathcal{V}$ to be modeled by the relations $P : \mathcal{V} \leftrightarrow \mathcal{K}$ and $N : \mathcal{V} \leftrightarrow \mathcal{K}$ as introduced in Section 3.1. We want to compute a vector of type $2^{\mathcal{V}} \leftrightarrow \mathbf{1}$ representing the satisfying assignments $\mathfrak{S} := \{A \in 2^{\mathcal{V}} \mid A \models \varphi\}$ as subset of $2^{\mathcal{V}}$ and, after that, a column-wise representation of $\mathfrak{S}$ as relation of type $\mathcal{V} \leftrightarrow \mathfrak{S}$. To obtain these goals, we take an assignment $A \in 2^{\mathcal{V}}$ and calculate as follows, where γ and x range over $\mathcal{K}$ and $\mathcal{V}$, respectively, the membership relation ε is of type $\mathcal{V} \leftrightarrow 2^{\mathcal{V}}$, and the universal vector L is of type $\mathcal{K} \leftrightarrow \mathbf{1}$. In the first step

the calculation uses the definition of satisfiability, in the second step Theorem 3.1 in combination with the definition of the two relations P and N, and in the third step the definition of the membership relation; the remaining steps apply predicate logic and some well-known correspondences between certain kinds of logical and relation-algebraic constructions.

$$\begin{aligned}
A \models \varphi &\iff \forall\, \gamma : A \models \gamma \\
&\iff \forall\, \gamma : (\exists\, x : x \in A \wedge P_{x\gamma}) \vee (\exists\, x : x \notin A \wedge N_{x\gamma}) \\
&\iff \forall\, \gamma : (\exists\, x : \varepsilon_{xA} \wedge P_{x\gamma}) \vee (\exists\, x : \overline{\varepsilon}_{\,xA} \wedge N_{x\gamma}) \\
&\iff \forall\, \gamma : (P^{\mathsf{T}}\varepsilon)_{\gamma A} \vee (N^{\mathsf{T}}\,\overline{\varepsilon}\,)_{\gamma A} \\
&\iff \forall\, \gamma : (P^{\mathsf{T}}\varepsilon \cup N^{\mathsf{T}}\,\overline{\varepsilon}\,)_{\gamma A} \\
&\iff \neg\exists\, \gamma : \neg(P^{\mathsf{T}}\varepsilon \cup N^{\mathsf{T}}\,\overline{\varepsilon}\,)_{\gamma A} \\
&\iff \neg\exists\, \gamma : \overline{P^{\mathsf{T}}\varepsilon \cup N^{\mathsf{T}}\,\overline{\varepsilon}}^{\,\mathsf{T}}_{A\gamma} \wedge \mathsf{L}_{\gamma} \\
&\iff \overline{\overline{P^{\mathsf{T}}\varepsilon \cup N^{\mathsf{T}}\,\overline{\varepsilon}}^{\,\mathsf{T}}\mathsf{L}}_{\,A}
\end{aligned}$$

If we remove the subscript A from the last expression of this calculation following the vector representation of sets introduced in Section 2 and apply after that some well-known relation-algebraic rules to transpose[1] only a "row vector" instead of a relation of type $\mathcal{K} \leftrightarrow 2^{\mathcal{V}}$, we get the relation-algebraic description

$$SatVect(P,N) := \overline{\mathsf{L}^{\mathsf{T}}\,\overline{P^{\mathsf{T}}\varepsilon \cup N^{\mathsf{T}}\,\overline{\varepsilon}}}^{\,\mathsf{T}} : 2^{\mathcal{V}} \leftrightarrow \mathbf{1} \tag{2}$$

of the vector representing the subset $\mathfrak{S}$ of $2^{\mathcal{V}}$. Hence, the CNF-formula φ is satisfiable iff $SatVect(P,N) \neq \mathsf{O}$. In this case the column-wise representation

$$SatSet(P,N) := \varepsilon\; inj(SatVect(P,N))^{\mathsf{T}} : \mathcal{V} \leftrightarrow \mathfrak{S} \tag{3}$$

of $\mathfrak{S}$ is an immediate consequence of the technique shown in Section 2.

3.3 Solving the Max-Sat Problem

Given again a CNF-formula φ over the variables $\mathcal{V}$, the objective of Max-Sat is to find an assignment that satisfies the maximal number of clauses of φ. A classical approach to solve this NP-hard problem is branch and bound; see e.g., [7]. In the following we show how to solve Max-Sat using relational algebra. Again we assume φ to be modeled by the relations $P : \mathcal{V} \leftrightarrow \mathcal{K}$ and $N : \mathcal{V} \leftrightarrow \mathcal{K}$.

Suppose $\varepsilon : \mathcal{V} \leftrightarrow 2^{\mathcal{V}}$ to be the membership relation with respect to variables. In Section 3.2 we have already shown that the relationships $(P^{\mathsf{T}}\varepsilon \cup N^{\mathsf{T}}\,\overline{\varepsilon}\,)_{\gamma A}$ and $A \models \gamma$ are equivalent. Now, let in addition $\epsilon : \mathcal{K} \leftrightarrow 2^{\mathcal{K}}$ be the membership relation with respect to the clauses of φ. Then we are able to calculate for each subset C of the set $\mathcal{K}$ as follows, where A ranges over $2^{\mathcal{V}}$, γ ranges over $\mathcal{K}$, and L is the universal vector of type $2^{\mathcal{V}} \leftrightarrow \mathbf{1}$. The second step of the calculation uses

[1] Using a BDD-implementation, transposition of a relation with domain or range $\mathbf{1}$ only means to exchange domain and range, the BDD remains unchanged; see [11].

the definition of ϵ and the equivalence just mentioned and the remaining steps again base on predicate logic and certain correspondences between logical and relation-algebraic constructions.

$$\begin{aligned}
C \text{ is satisfiable} &\iff \exists\, A : \forall\, \gamma : \gamma \in C \to A \models \gamma \\
&\iff \exists\, A : \forall\, \gamma : \epsilon_{\gamma C} \to (P^{\mathsf{T}} \varepsilon \cup N^{\mathsf{T}}\, \overline{\varepsilon})_{\gamma A} \\
&\iff \exists\, A : \neg \exists\, \gamma : \epsilon_{\gamma C} \wedge \neg (P^{\mathsf{T}} \varepsilon \cup N^{\mathsf{T}}\, \overline{\varepsilon})_{\gamma A} \\
&\iff \exists\, A : \neg \exists\, \gamma : \epsilon^{\mathsf{T}}_{C\gamma} \wedge \overline{P^{\mathsf{T}} \varepsilon \cup N^{\mathsf{T}}\, \overline{\varepsilon}}\,_{\gamma A} \\
&\iff \exists\, A : \overline{\epsilon^{\mathsf{T}}\, \overline{P^{\mathsf{T}} \varepsilon \cup N^{\mathsf{T}}\, \overline{\varepsilon}}}\,_{CA} \wedge \mathsf{L}_A \\
&\iff (\overline{\epsilon^{\mathsf{T}}\, \overline{P^{\mathsf{T}} \varepsilon \cup N^{\mathsf{T}}\, \overline{\varepsilon}}}\; \mathsf{L})_C
\end{aligned}$$

If we remove the subscript C from the last expression of this calculation and apply after that, as in Section 3.2, some simple transposition rules to improve efficiency in view of a BDD-implementation of relations, this leads to $(\mathsf{L}^{\mathsf{T}}\, \overline{\overline{\varepsilon^{\mathsf{T}} P \cup \overline{\varepsilon}^{\mathsf{T}} N}\, \epsilon})^{\mathsf{T}} : 2^{\mathcal{K}} \leftrightarrow \mathbf{1}$ as a relation-algebraic description of the vector representing the set of those clause sets which are satisfiable. Next, we use the size-comparison relation $S : 2^{\mathcal{K}} \leftrightarrow 2^{\mathcal{K}}$, which relates two clause sets C_1 and C_2 iff $|C_1| \leq |C_2|$. It is obvious how to combine S with the relation-algebraic description of the satisfiable clause sets and the relation-algebraic description of the greatest elements in order to compute the vector representation of the subset $\mathfrak{C}$ of $2^{\mathcal{K}}$ containing the largest satisfiable clause sets. Here is the result.

$$\mathit{MaxSatVectCl}(P, N) := \mathit{GreEl}(S, (\mathsf{L}^{\mathsf{T}}\, \overline{\overline{\varepsilon^{\mathsf{T}} P \cup \overline{\varepsilon}^{\mathsf{T}} N}\, \epsilon})^{\mathsf{T}}) : 2^{\mathcal{K}} \leftrightarrow \mathbf{1} \qquad (4)$$

Now, we are almost done. To find an assignment that satisfies the maximal number of clauses of the CNF-formula φ we select a point $p : 2^{\mathcal{K}} \leftrightarrow \mathbf{1}$ contained in the vector $\mathit{MaxSatVectCl}(P, N)$. It represents a set C from $\mathfrak{C}$ as an element of $2^{\mathcal{K}}$. The little calculation

$$(\epsilon p)_\gamma \iff \exists\, K : \epsilon_{\gamma K} \wedge p_K \iff \exists\, K : \gamma \in K \wedge K = C \iff \gamma \in C$$

for all clauses $\gamma \in \mathcal{K}$ shows that the same set C is represented as a subset of the clause set $\mathcal{K}$ by the vector $\epsilon p : \mathcal{K} \leftrightarrow \mathbf{1}$. Hence, the restrictions $P\, \mathit{inj}(\epsilon p)^{\mathsf{T}} : \mathcal{V} \leftrightarrow C$ and $N\, \mathit{inj}(\epsilon p)^{\mathsf{T}} : \mathcal{V} \leftrightarrow C$ of $P : \mathcal{V} \leftrightarrow \mathcal{K}$ and $N : \mathcal{V} \leftrightarrow \mathcal{K}$, respectively, to the range $C \subseteq \mathcal{K}$ model the sub-formula ψ of φ which consists of the conjunction of the clauses of C. Consequently, the following vector represents the assignments satisfying ψ, i.e., solutions of Max-Sat.

$$\mathit{MaxSatVect}(P, N) := \mathit{SatVect}(P\, \mathit{inj}(\epsilon p)^{\mathsf{T}}, N\, \mathit{inj}(\epsilon p)^{\mathsf{T}}) : 2^{\mathcal{V}} \leftrightarrow \mathbf{1} \qquad (5)$$

For the column-wise enumeration of the assignments satisfying ψ, in (5) only the relational function *SatVect* has to be replaced by *SatSet*.

3.4 Recognizing Independent Variables

Let φ be a Boolean formula over $\mathcal{V}$ and $x \in \mathcal{V}$ be a variable. Then φ is said to be *independent* of x if for all assignments $A \in 2^{\mathcal{V}}$ the relationships $A \cup \{x\} \models \varphi$ and

$A \setminus \{x\} \models \varphi$ are equivalent. Testing this property (or enumerating all variables with this property) is an important problem of Boolean logic and the contents of this section is its relation-algebraic solution. As in the previous sections, we assume φ to be given in CNF $\gamma_1 \wedge \ldots \wedge \gamma_n$.

We want to reduce the independence problem of the CNF-formula φ with respect to the variable x to satisfiability. For that reason we consider the formula $\varphi^x := \gamma_1^x \wedge \ldots \wedge \gamma_n^x$, which results from φ by removing from its clauses all literals x and $\neg x$. If a clause γ_i of φ is of the form x or $\neg x$, i.e., it disappears during this process, then the corresponding formula γ_i^x is defined as Boolean constant *false*. A formal way to obtain this is to represent *false* as $y \wedge \neg y$, where y is the next fresh variable. Then also φ^x is a formula over $\mathcal{V}$ in CNF, but the number of clauses of φ and φ^x may be different. The constant *false* is only introduced to obtain a 1-1 correspondence between the clauses of φ and the conjuncts of φ^x, which simplifies presentation. After these preparations, we are in a position to prove the following decisive fact.

Theorem 3.2. *The CNF-formula φ over $\mathcal{V}$ is independent of the variable $x \in \mathcal{V}$ iff the sets $\{A \in 2^{\mathcal{V}} \mid A \models \varphi^x\}$ and $\{A \in 2^{\mathcal{V}} \mid A \models \varphi\}$ are equal.*

Proof. In the first part of the proof of "$\Rightarrow$" we assume $A \in 2^{\mathcal{V}}$ to be an assignment with $A \models \varphi^x$. Then φ does not contain clauses of the form x or $\neg x$ and, as a consequence, each of its clauses γ_i, $1 \leq i \leq n$, is obtained from the corresponding clause γ_i^x of φ^x by adding x or $\neg x$ via a disjunction. This implies $A \models \varphi$ and, summing up, we have shown $\{A \in 2^{\mathcal{V}} \mid A \models \varphi^x\} \subseteq \{A \in 2^{\mathcal{V}} \mid A \models \varphi\}$.

In the second part, we assume that this is a proper inclusion and derive a contradiction. Therefore, let an assignment $A \in 2^{\mathcal{V}}$ be given such that $A \not\models \varphi^x$ and $A \models \varphi$. The first property implies $A \not\models \gamma_i^x$ for some i, $1 \leq i \leq n$, but $A \models \gamma_i$ holds due to the second property. Hence, γ_i^x and γ_i must be different.

Let γ_i be of the form $\lambda_1 \vee \ldots \vee \lambda_m$, where $m \geq 2$. Since clauses do not contain duplicates of literals and also the literals x and $\neg x$ cannot occur simultaneously in clauses, γ_i^x equals $\lambda_1 \vee \ldots \vee \lambda_{k-1} \vee \lambda_{k+1} \vee \ldots \vee \lambda_m$ for some k, $1 \leq k \leq m$. From $A \not\models \gamma_i^x$ we get $A \not\models \lambda_j$ for all $j \neq k$, which in turn implies $A \models \lambda_k$ due to $A \models \gamma_i$. Now, we distinguish two cases: If λ_k equals x, then $A \models \lambda_k$ is equivalent to $x \in A$ and the assumption $A \models \varphi$ becomes $A \cup \{x\} \models \varphi$. But for all $j \neq k$ the variable x does not occur in λ_j. Hence, for these literals $A \not\models \lambda_j$ is equivalent to $A \setminus \{x\} \not\models \lambda_j$ due to the coincidence lemma of logic. The property $A \setminus \{x\} \not\models \lambda_k$ is a consequence of $x \notin A \setminus \{x\}$. Altogether, we have $A \setminus \{x\} \not\models \lambda_j$ for all j, $1 \leq j \leq m$. As a consequence, $A \setminus \{x\} \not\models \gamma_i$ holds, which yields $A \setminus \{x\} \not\models \varphi$. This contradicts the assumption of φ being independent of x. If the literal λ_k equals $\neg x$, the proof is similar to the previous one.

It remains to consider the case of γ_i being a single literal. If γ_i is x, then $A \models \gamma_i$ yields $x \in A$ and the assumption $A \models \varphi$ becomes $A \cup \{x\} \models \varphi$. Now, the contradiction $A \setminus \{x\} \not\models \varphi$ follows from $x \notin A \setminus \{x\}$ and its consequence $A \setminus \{x\} \not\models \gamma_i$. In the same way one deals with the literal $\neg x$.

A proof of direction "$\Leftarrow$" is rather simple. Assume $A \in 2^{\mathcal{V}}$ to be an assignment. Then we have $A \cup \{x\} \models \varphi$ iff $A \setminus \{x\} \models \varphi$ due to $\{A \in 2^{\mathcal{V}} \mid A \models \varphi^x\} =$

$\{A \in 2^{\mathcal{V}} \mid A \models \varphi\}$ and the fact that x does not occur in φ^x (cf. coincidence lemma of logic). Hence, φ is independent of x. $\square$

Now, let $P : \mathcal{V} \leftrightarrow \mathcal{K}$ and $N : \mathcal{V} \leftrightarrow \mathcal{K}$ be the relational model of φ. Furthermore, suppose the point $p : \mathcal{V} \leftrightarrow \mathbf{1}$ to represent the variable $x \in \mathcal{V}$. Guided by the definition of φ^x and Theorem 3.2, we first compute the vector representation of the set of those clauses which are of the form x or $\neg x$. So, assume $\gamma \in \mathcal{K}$. Then we can calculate as follows, where y ranges over $\mathcal{V}$ and the second step exploits that the relational complement $\overline{p}$ represents the set complement $\mathcal{V} \setminus \{x\}$.

$$\begin{aligned} \gamma \text{ is of the form } x \text{ or } \neg x &\iff \neg\exists\, y : (P_{y\gamma} \vee N_{y\gamma}) \wedge y \neq x \\ &\iff \neg\exists\, y : (P^{\mathsf{T}}_{\gamma y} \vee N^{\mathsf{T}}_{\gamma y}) \wedge \overline{p}\,_y \\ &\iff \overline{(P \cup N)^{\mathsf{T}}\, \overline{p}}\,_{\gamma} \end{aligned}$$

Hence, $\overline{(P \cup N)^{\mathsf{T}}\, \overline{p}} : \mathcal{K} \leftrightarrow \mathbf{1}$ is the vector representation of the set of all clauses which are equal to x or to $\neg x$. If this vector is not empty, then a conjunct of φ^x is *false* and Theorem 3.2 implies the following fact.

$$\varphi \text{ independent of } x \iff \mathit{SatVect}(P, N) = \mathsf{O} \tag{6}$$

In the case $\overline{(P \cup N)^{\mathsf{T}}\, \overline{p}} = \mathsf{O}$ the relational model $P^x : \mathcal{V} \leftrightarrow \mathcal{K}$ and $N^x : \mathcal{V} \leftrightarrow \mathcal{K}$ of the formula φ^x is obtained by deleting from P and N all pairs (x, γ), where $\gamma \in \mathcal{K}$. Relation-algebraically this means that P^x equals $P \cap \overline{p\mathsf{L}}$ and N^x equals $N \cap \overline{p\mathsf{L}}$, where $\mathsf{L} : \mathbf{1} \leftrightarrow \mathcal{K}$, and the characterization

$$\varphi \text{ independent of } x \iff \mathit{SatVect}(P, N) = \mathit{SatVect}(P \cap \overline{p\mathsf{L}}, N \cap \overline{p\mathsf{L}}) \tag{7}$$

is again a consequence of Theorem 3.2. Note that in the case $\overline{(P \cup N)^{\mathsf{T}}\, \overline{p}} = \mathsf{O}$ neither $P \cap \overline{p\mathsf{L}}$ nor $N \cap \overline{p\mathsf{L}}$ contains empty columns, because of the general assumption that no clause contains a variable and its negation as literals.

It should be remarked that, if there are clauses in φ which contain a variable as well as its negation as literals, their removal is very easy to model with relation-algebraic means, too. We only have to restrict the range of P and N with respect to the vector $(P \cap N)^{\mathsf{T}}\mathsf{L} : \mathcal{K} \leftrightarrow \mathbf{1}$ since the equivalence of $((P \cap N)^{\mathsf{T}}\mathsf{L})_\gamma$ and $\exists\, x : P_{x\gamma} \wedge N_{x\gamma}$ for all $\gamma \in \mathcal{K}$ proves that this vector represents the set of clauses which contain a variable and its negation as literals.

4 Implementation and Experimental Results

In the case of finite carrier sets the operations of relational algebra can be implemented very efficiently. At Kiel University we have developed a visual computer system for the manipulation of relations and relational programming, called RelView. It is written in C, uses – as already mentioned – BDDs for representing relations, and makes full use of the X-windows graphical user interface.

The main purpose of RelView is the evaluation of relation-algebraic expressions which are constructed from the relations of its workspace using pre-defined

operations and tests, user-defined relational functions, and user-defined relational programs. Relational functions are of the form $F(X_1, \ldots, X_n) = t$, where F is the function name, the X_i, $1 \leq i \leq n$, are the formal parameters (standing for relations), and t is a relation-algebraic expression over the relations of the system's workspace that can additionally contain the formal parameters. A relational program is much like a function procedure in the programming languages Pascal or Modula 2, except that it only uses relations as data type.

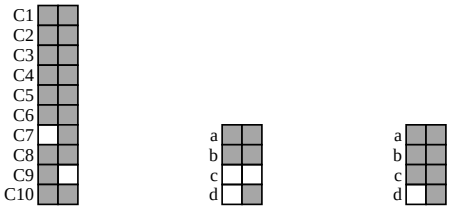
For example, the relation-algebraic description (2) of the vector representing all satisfying assignments immediately leads to the following RelView-program.

```
SatVect(P,N)
  DECL E
  BEG  E = epsi(0(P))
       RETURN -(L1n(P) * -(P^ * E | N^ * -E))^
  END.
```

A translation of the remaining relation-algebraic descriptions of Section 3 into RelView-code is also straightforward.

Now, we return to the example of Section 3.1 and assume the two relations P and N for modeling the formula (1) to be stored in RelView's workspace under the names `P` and `N`. Then the call `SatVect(P,N)` returns an empty Boolean vector of length 2^4. This means that the formula φ is not satisfiable.

To solve the Max-Sat problem for the formula φ of (1), we can use again RelView and compute the column-wise representation of the largest satisfiable clause sets and the assignments satisfying these clauses. The corresponding labeled Boolean matrices look as in the following pictures. From the Boolean matrix on the left we obtain that all clauses except c (the 7^{th} one) or all clauses except $\neg a \vee \neg c$ (the 9^{th} one) are satisfiable. The columns of the matrix in the middle represent the assignments $\{a, b\}$ and $\{a, b, d\}$ which satisfy all clauses except c, and the columns of the matrix on the right represent the assignments $\{a, b, c\}$ and $\{a, b, c, d\}$ which satisfy all clauses except $\neg a \vee \neg c$.



Besides this small example we have tested our approach with many further and larger examples. The tests have been carried out on a Sun Fire-880 workstation running Solaris 9 at 750 MHz and with 32 GByte main memory. From this memory, however, at most 4 GByte has actually been in use by RelView.

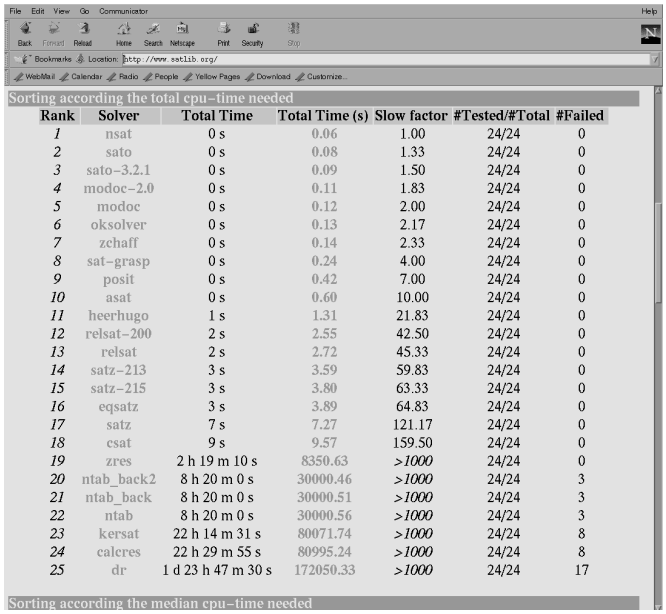
Using RelView's operations for random relations and random permutations (see [11]), we have generated a lot of uniformly distributed random instances of k-Sat. The following table shows some of the experimental results for 3-Sat

with 50 variables. As the first column indicates, we only looked at formulae where the number of clauses to variable ratio equals 3.0 to 5.4. It is known (see e.g., [4]) that for random instances of k-SAT within a region around the so-called *critical value* α_k (where $\alpha_3 \approx 4.2$) there is a sharp transition from satisfiabilty to unsatisfiability which makes problems particularly hard for SAT-solvers.

	satisfying assignments			time (sec.)		
ratio	min	mean	max	min	mean	max
3.0	453615	12392518	53870974	99	280	593
3.6	1556	51766	233759	169	509	1451
4.2	0	5032	98009	375	720	1998
4.8	0	0	14	369	843	1501
5.4	0	0	0	280	817	1854

To be precise, 49 of the 50 instances with ratio 4.8 we have investigated proved to be unsatisfiable. In the remaining case we obtained 14 satisfying assignments.

Of course, in general RELVIEW cannot compete in efficiency with special purpose tools for the problems we dealt with (although the complexities are usually the same). Nevertheless, our tests showed that in many cases our approach does not need too much time compared with other tools. Sometimes it is even superior. E.g., in the case of the 24 DIMACS aim-50 instances of www.satlib.org [6] (generated with the method of [1], 50 variables, between 80 and 300 clauses) the computation times of RELVIEW are between 0.03 to 2.63 seconds, with 0.5 seconds as arithmetic mean. Comparing this with the corresponding times of the 25 SAT-solvers of the rank list of www.satlib.org for aim-50, RELVIEW is between the numbers 9 (posit, 0.42 seconds) and 10 (asat, 0.60 seconds).



The aim-50 rank list has been established by means of a Pentium II PC running Linux at 400 MHz and with 512 MByte main memory. In practice, the computation times of RelView on a Pentium II PC and a Sun Fire-800 workstation are rather equal. But even if we assume that RelView runs on the PC half as fast than on the workstation, the tool is still faster than number 11 of the rank list (heerhugo, 1.31 seconds).

We also have applied the program `SatVect` to many other benchmarks presented at www.satlib.org. Here we used RelView in combination with a small C program for converting the DIMACS cnf-format into the ASCII file-format of RelView, but also Kure, a C library which has been developed in the course of the Ph.D. thesis [11] and consists of the functional core of RelView. For all 1000 uniform random 3-Sat instances with 20 variables and 91 clauses (critical value instances uf20-91) both systems computed the vectors describing the satisfying assignments in less than 0.06 seconds (arithmetic mean 0.037 seconds). The computation times of the 1000 uniform random 3-Sat instances with 50 variables and 218 clauses (critical value instances uf50-218 and uuf50-218) ranged from 77 to 3509 seconds; as arithmetic mean of all times we obtained 766 seconds.

We believe that the reason for the good behaviour of RelView when solving Sat is that during a run of `SatVect(P,N)` both input relations `P` and `N` are represented by relatively small BDDs, which have at most $|\mathcal{K}| * |\mathcal{V}|$ nodes. The membership relation is represented by another small BDD, complement and union of relations are not time consuming, and the same holds for the transposition of small relations. Therefore, we can only get large BDDs, which represent interim results, during the computation of the composition with the relation `L` in the `RETURN`-expression of `SatVect`. We found in experiments that our method is able to compete with Sat-solvers especially if the given formula is satisfiable. In the other case, Sat-solvers often stop the computation early with the result "unsatisfiable", whereas our approach always has to get to the end.

5 Conclusion

Our relation-algebraic approach can be applied to many further Boolean logic problems, especially those which are closely related to Sat. An example is Unique-Sat. It tests a CNF-formula φ to have a unique satisfying assignment. Relation-algebraically this means that the vector $SatVect(P, N)$ is a point, where φ is modeled by P and N. Recognizing *frozen variables* with respect to φ, i.e., variables which are either contained in all φ-satisfying assignments or in none of them, is another example. Calculation shows that the set of frozen variables is described by the vector $\overline{\overline{\varepsilon}\, v} \cap \overline{\varepsilon v} : \mathcal{V} \leftrightarrow \mathbf{1}$, where $v = SatVect(P, N)$ and $\varepsilon : \mathcal{V} \leftrightarrow 2^{\mathcal{V}}$ is the membership relation with respect to variables.

A clause of the form $\lambda_1 \vee \cdots \vee \lambda_n$ is *valid* (i.e., satisfied by every assignment) iff there are $1 \leq i, j \leq n$ such that λ_i equals $\neg\lambda_j$; see e.g., [8]. Based on this fact and the vector-representation of the set of clauses which contain a variable and its negation as literals (see the end of Section 3.4), we have an easy and fast

check for validity of CNF-formulae. If φ is modeled by the relations $P : \mathcal{V} \leftrightarrow \mathcal{K}$ and $N : \mathcal{V} \leftrightarrow \mathcal{K}$, then φ is valid iff $(P \cup N)^{\mathsf{T}}\mathsf{L} = \mathsf{L}$.

Presently, we investigate how to generalize our approach to arbitrary Boolean formulae. Using matrix terminology, this is based on the fact that all possible results $\{f(a_1, \ldots, a_n) \mid a_1, \ldots, a_n \in \mathbb{B}\}$ of an n-ary Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ can be computed "in parallel" by applying a corresponding relational function to the n rows of the $n \times 2^n$ Boolean "membership matrix". Also, an interesting direction for future research is to find out how powerful our approach is in the case of practical applications, if these are reduced to Boolean logic problems as, for instance, the factorization of large numbers, asynchronous circuit synthesis, and SAT-based model checking. Besides this better insight into the weaknesses and strengths of our approach, we are also interested on further empirical comparisons with other SAT-solvers.

References

1. Asahiro Y, Iwama K, Miyano E: Random generation of test instances with controlled attributes. In: Johnson D.S., Trick M.A. (eds.): Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge. DIMACS Series on Discr. Math. and Theoret. Comput. Sci. 26, 377-394 (1996)
2. Behnke R. et al.: RelView — A system for calculation with relations and relational programming. In: Astesiano E. (ed.): Proc. 1st Conf. *Fundamental Approaches to Software Engineering*, LNCS 1382, Springer, 318-321 (1998)
3. Bryant R.E.: Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Comp. Surveys 24, 293-318 (1992)
4. Crawford J.M., Auton L.D.: Experimental results on the crossover point in random 3SAT. Artificial Intelligence 81, 59-80 (1996)
5. Dantsin E., Hirsch E.A.: Algorithms for SAT and upper bounds of their complexity. Electr. Coll. Comp. Compl., Rep. 12, http://www.eccc.uni-trier.de/eccc (2001)
6. Hoos H.H., Stützle T.: SATLIB: An online resource for research on SAT. In: Gent I.P., v. Maaren H., Walsh T (eds.): SAT 2000, 283-292, IOS Press (2000)
7. Hromkovic J.: Algorithms for hard problems. Introduction to combinatorial optimization, randomization, approximation, and heuristics. EATCS Texts in Theoret. Comput. Sci., Springer (2001)
8. Huth M.R.A., Ryan M.D.: Logic in computer science. Cambr. Univ. Press (2000)
9. Leoniuk B.: ROBDD-based implementation of relational algebra with applications (in German). Ph.D. thesis, Inst. für Inf. und Prak. Math., Univ. Kiel (2001)
10. Lind-Nielson J.: BuDDy, a binary decision diagram package, version 2.2. Techn. Univ. of Denmark, http://www.itu.dk/research/buddy (2003)
11. Milanese U.: On the implementation of a ROBDD-based tool for the manipulation and visualization of relations (in German). Ph.D. thesis, Inst. für Inf. und Prak. Math., Univ. Kiel (2003)
12. Purdam P.: A survey of average time analysis of satisfiability algorithms. J. of Inf. Processing 13, 449-455 (1990)
13. Schmidt G., Ströhlein T.: Relations and graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993)
14. Somenzi F.: CUDD: CU decision diagram package, release 2.3.1. Univ. of Colorado at Boulder, http://www.vlsi.colorado.edu/~fabio/CUDD (2001)
15. Tarski A.: On the calculus of relations. J. Symbolic Logic 6, 73-89 (1941)