Investigating Finite Models of Non-classical Logics with Relation Algebra and RelView[⋆]

Rudolf Berghammer[1] and Renate A. Schmidt[2]

[1] Institut für Informatik, Christian-Albrechts-Universität Kiel
Olshausenstraße 40, 24098 Kiel, Germany
rub@informatik.uni-kiel.de

[2] School of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, United Kingdom
Renate.Schmidt@manchester.ac.uk

Abstract. In computer science, scenarios with interacting agents are often developed using modal logic. We show how to interpret modal logic of knowledge in relation algebra. This allows the use of the RelView tool for the purpose of investigating finite models and for visualizing certain properties. Our approach is illustrated with the well-known 'muddy children' puzzle using modal logic of knowledge. We also sketch how to treat other non-classical logics in this way. In particular, we explore our approach for computational tree logic and illustrate it with the 'mutual exclusion' example.

1 Introduction

For some time now researchers in computer science have been interested in reasoning about knowledge in multi-agent systems. Here a group of interacting agents is given and it is assumed that each agent takes into account not only facts that are true about the world, but also the knowledge of other agents. Applications of this scenario can be found in many domains of computer science, for instance in distributed computing, cryptography, and robotics.

The idea of using modal logic for reasoning about knowledge goes back to J. Hintikka and has been worked out in great detail, e.g. in the textbooks [9, 12, 19]. The standard semantics of modal logic is based on the agents' accessibility relations on a global set of possible worlds. In this paper we adopt an algebraic perspective. Relation algebra, and more generally Boolean algebras with operators, provide natural settings for studying modal logics and other kinds of non-classical logics, cf. [2, 6, 13, 23] for example. A sufficient framework for interpreting modal logic of knowledge is dynamic algebra [16, 24]. However in this paper we interpret modal logics in the more expressive setting of heterogeneous relation algebras with transitive closure (see [22, 25, 26]) and their representation as Boolean matrices [25]. Representing sets (respectively, predicates on sets) by

⋆ The authors thank Harrie de Swart and the anonymous referees for their comments. The work was supported by EU COST Action 274 (Tarski).

specific relations, viz. vectors, relation-algebraic specifications can be evaluated by calculations on Boolean matrices and vectors, and properties of relations can be verified in this way. Hence, the relation-algebraic manipulation and visualization system RelView [1, 20, 4] can be applied for the purpose of model checking and similar tasks. It turns out that this can be achieved with very little effort and that the approach can be transferred to other important non-classical logics, which are embeddable into the programming language of RelView, such as temporal logic which we consider in this paper but also Peirce logic and description logic.

The case study in this paper explores a novel application of the RelView tool for which it was not originally designed. The application may be of interest to researchers working in the area of modal logics, since to our knowledge, there seem to be very few tools available for solving and visualizing computational problems of finite models in modal logic. One of the uses of RelView we explore is its use as a finite model checker. However we do not claim any superiority of the system over existing implemented model checking systems such as Mcmas [18] and Verics [14, 15]). Sophisticated model checking tools which have been developed for computational tree logic, linear temporal logic, and the process algebra CSP include Spin, Smv, Kronos, Uppall, and Fdr2. Because of the global approach that RelView takes, it cannot compete directly with systems based on local evaluations. Nevertheless, the underlying technology of RelView is based on reduced, ordered BDDs which are fast [17, 3, 20]. Furthermore, the tool has a convenient graphical user interface and provides useful capabilities for manipulating and displaying relations and graphs. Particularly attractive in the context of modal logic is the presence of the operator `trans` for computing transitive closures in the tool's programming language. This is useful for performing finite model reasoning tasks for a modal logic with the common knowledge operator and also for dynamic logic. Such logics cannot be handled directly for example by first-order logic theorem provers since the transitive closure operator and the common knowledge operator are not first-order definable.

The remainder of the paper is organized as follows. Some basic notions of modal logic and modal logic of knowledge are recalled in Sections 2 and 3. Section 4 describes how to interpret modal logic of knowledge in relation algebra and how then the RelView tool can be used for solving computational problems on finite models. The application of the approach to the well-known 'muddy children' puzzle is presented in Section 5. This example also demonstrates how RelView can be used for visualizing models, and solutions of tasks. Our method can be extended to all non-classical logics, embeddable into the programming language of RelView. Section 6 features the approach for computational tree logic and the 'mutual exclusion' example in more detail. In Section 7 some further applications of relation algebra and RelView in the context of modal logic are considered. Finally, Section 8 concludes with some further remarks about the approach and the use of RelView.

2 Modal Logic

The language of (propositional) modal logic with multiple modalities is defined over countably many propositional variables $p_1, p_2, p_3, \ldots$, and finitely many modalities $\Diamond_1, \ldots, \Diamond_n$, one for each agent $1, \ldots, n$. A *propositional atom* is a propositional variable or the constant $\top$ (the symbol for 'true') and a *modal formula* is either a propositional atom or a formula of the form $\neg\phi$, $\phi \wedge \psi$, and $\Diamond_i\phi$. We define the constant $\perp$ (the symbol for 'false') and the other propositional connectives $\vee$, $\rightarrow$, and $\leftrightarrow$ as usual, e.g. $\phi \rightarrow \psi := \neg\phi \vee \psi$. Furthermore, the dual operator of $\Diamond_i$ is defined by $\Box_i\phi := \neg\Diamond_i\neg\phi$.

The standard semantics of modal logic is given by the well-known *Kripke semantics* (or *possible world semantics*). A *frame* (or *relational structure*) for a modal logic is a pair $\mathcal{F} = (W, \{R_1, \ldots, R_n\})$, where W is a non-empty set of worlds and each R_i is a binary relation over W. W is the set of possible worlds (or states) in which the truth of formulae is evaluated. The R_i are the accessibility relations which determine the formulae deemed possible by an agent i in a given world ($1 \leq i \leq n$). A *model* is a pair $M = (\mathcal{F}, \iota)$ of a frame $\mathcal{F}$ and a valuation function ι from the set of propositional variables to 2^W, where $\iota(p_i)$ is interpreted to be the set of worlds in which p_i is true. The *truth* of a modal formula in a world x of a model M is defined as follows (where the notation $R_i(x, y)$ means that the elements x and y are related via the relation R_i).

$$\begin{array}{rcl}
M, x \models \top & & \\
M, x \models p_i & :\Longleftrightarrow & x \in \iota(p_i) \\
M, x \models \neg\phi & :\Longleftrightarrow & M, x \not\models \phi \\
M, x \models \phi \wedge \psi & :\Longleftrightarrow & M, x \models \phi \text{ and } M, x \models \psi \\
M, x \models \Diamond_i\phi & :\Longleftrightarrow & \exists\, y \in W : R_i(x, y) \text{ and } M, y \models \phi
\end{array}$$

If $M, x \models \phi$ we also say that x satisfies ϕ. A modal formula is *valid* in a model M iff the formula is true in every world of M. It is valid in a frame $\mathcal{F}$ iff it is valid in all models based on the frame, i.e. in all models $(\mathcal{F}, \iota)$.

For the purposes of this paper it suffices to consider modal logic from a semantic perspective. (The reader interested in the axiomatizations of the considered logics should refer to standard textbooks, e.g. [5, 7, 10, 11].) A modal logic L is said to be *sound* (respectively *complete*) *with respect to a class of frames* iff for any modal formula ϕ, any frame in the class validates ϕ if (respectively iff) ϕ is a theorem in L. A modal logic is said to be *complete* iff it is complete with respect to some class of frames.[1]

The basic multi-modal logic $K_{(m)}$ is complete with respect to the class of all frames. The table in Figure 1 lists the relation-algebraic correspondence properties satisfied by classes of frames for extensions of the basic logic $K_{(m)}$. This means, if L denotes an extension of the basic logic $K_{(m)}$ with a subset of the common axioms listed in the table then L is a logic (sound and) complete with

[1] Note in modal logic the notion of completeness is used differently than in other logical disciplines.

	Axiom	Correspondence property	
T	$\Box_i p \to p$	reflexivity	$\mathsf{I} \subseteq R_i$
4	$\Box_i p \to \Box_i \Box_i p$	transitivity	$R_i ; R_i \subseteq R_i$
B	$\Diamond_i \Box_i p \to p$	symmetry	$R_i \subseteq {R_i}^{\mathsf{T}}$
D	$\Box_i p \to \Diamond_i p$	seriality	$\mathsf{L} \subseteq R_i ; \mathsf{L}$
alt_1	$\Diamond_i p \to \Box_i p$	functionality	${R_i}^{\mathsf{T}} ; R_i \subseteq \mathsf{I}$
5	$\Diamond_i \Box_i p \to \Box_i p$	Euclideanness	${R_i}^{\mathsf{T}} ; R_i \subseteq R_i$

Fig. 1. Modal axioms and their frame correspondence properties

respect to the class of all frames which satisfy each of the corresponding properties. In the table, I denotes the identity relation and L denotes the universal relation. Furthermore, $R;R$ denotes the composition of R with itself and R^{T} the transpose (converse) of R. Other relation-algebraic constructions used in this paper are the empty relation O, the Boolean constructs $R \cup S$ (union), $R \cap S$ (intersection), $\overline{R}$ (complement), and the transitive closure $R^+ := \bigcup_{k \geq 1} R^k$ of R. Here we assume powers are defined inductively by $R^0 := \mathsf{I}$ and $R^{k+1} := R; R^k$ for $k \geq 0$.

3 Modal Logic of Knowledge

Modal logic lends itself to formalize informational aspects of agent-based scenarios. Consider the language defined in Section 2 in which $\Box_i \phi$, from now on written $K_i \phi$, is interpreted as 'the agent i knows that property ϕ is the case'. For this reading it is usual to assume that the following axioms of the table in Figure 1 are valid: T (axiom of true knowledge), 4 (agents are positively introspective) and 5 (agents are negatively introspective). The accessibility relations R_i associated with the knowledge operators K_i are therefore equivalence relations on the set of worlds W (because each R_i is reflexive and transitive and $R_i^{\mathsf{T}} = R_i^{\mathsf{T}}; \mathsf{I} \subseteq R_i^{\mathsf{T}}; R_i \subseteq R_i$ shows symmetry).

In order to handle the common knowledge of a group of agents two additional modal operators, E_G and C_G, are required. Let G denote a finite set of agents. Then the modal formula $E_G \phi$ is read to mean that 'each of the agents in G knows that ϕ is the case', and the modal formula $C_G \phi$ is read to mean that 'it is common knowledge among the group G of agents that ϕ is the case'. Their semantics is defined by the following equivalences, where $E_G^k \phi$ is an abbreviation of the modal formula $E_G \ldots E_G \phi$ with k occurrences of the operator E_G.

$$\begin{aligned} M, x \models E_G \phi \quad &:\Longleftrightarrow \quad \forall\, i \in G : M, x \models K_i \phi \\ M, x \models C_G \phi \quad &:\Longleftrightarrow \quad \forall\, k \geq 1 : M, x \models E_G^k \phi \end{aligned}$$

If $G = \{i_1, \ldots, i_m\}$, then we have the following equivalence.

$$M, x \models E_G \phi \quad \Longleftrightarrow \quad M, x \models K_{i_1} \phi \wedge \ldots \wedge K_{i_m} \phi$$

Thus, the formula $E_G \phi$ is true in a world of a model iff everyone in the group knows that ϕ is true. Furthermore, the formula $C_G \phi$ is true iff everyone in the group knows that ϕ is true and everyone in the group knows that everyone in the group knows that ϕ is true, and so on. The following three properties are not difficult to show for any model M and any world x of M. We assume that R is the union of the accessibility relations R_i for all $i \in G$, i.e. $R := \bigcup_{i \in G} R_i$.

$$\begin{array}{lcl} M, x \models C_G\phi & \iff & M, x \models E_G(\phi \wedge E_G C_G \phi) \\ M, x \models E_G^k \phi & \iff & \forall\, y \in W : R^k(x, y) \text{ implies } M, y \models \phi \\ M, x \models C_G\phi & \iff & \forall\, y \in W : R^+(x, y) \text{ implies } M, y \models \phi \end{array}$$

Distributed knowledge is another concept central to modal logics of knowledge. Here a group of agents can deduce a formula by pooling their knowledge together. Since this distributed knowledge is not used in the 'muddy children' puzzle of Section 5, we omit the technical details and refer to the textbooks cited in Section 2. Relation algebra does however allow us to model distributed knowledge by using the same techniques which we apply in the next section to model the modal logic of common knowledge.

4 Relational Model Checking

The term 'model checking' refers to automatic model-based verification approaches; see e.g., [21, 8]. In the case of modal logic it involves solving tasks of the following kind. Suppose that $M = (\mathcal{F}, \iota)$ is a given *finite model*, where the frame is $\mathcal{F} = (W, \{R_1, \ldots, R_n\})$, and ϕ is a given modal formula.

(1) Determine whether ϕ is true in a given world of M (satisfiability in a given world of a model).

(2) Determine whether there is a world of M in which ϕ is true (satisfiability in a model).

(3) Determine whether ϕ is true in all worlds of M (global satisfiability/validity in a model).

(4) Determine the set of all worlds of M in which ϕ is true.

In this paper, we use relation algebra and the RelView tool to compute the set of all worlds of M in which ϕ is true (i.e. to solve task (4)). This immediately leads to solutions of tasks (1)–(3), too.

Our solution is based on the representation of sets of worlds by so-called *vectors* over W. Such vectors are relations with W as the domain and a singleton set, $\{\bullet\}$ say, as the range. Since this specific range is irrelevant, in the following we omit for a vector v the second argument and write $v(x)$ instead of $v(x, \bullet)$. A vector v over W can be viewed as a Boolean column vector and *represents the set* $\{x \in W \mid v(x)\}$ of worlds.

Suppose we wish to describe an arbitrary modal formula ϕ via the vector of worlds in which it is true, that is, we want to compute the vector v_ϕ representing the set $\{x \in W \mid M, x \models \phi\}$. We start by defining for the constant $\top$ the vector $\mathsf{v}_\top$ as the universal vector L over W (the universal relation with domain

W and range $\{\bullet\}$). Then for each propositional variable p in ϕ we define a vector v_p representing the set $\iota(p)$. Using Boolean vector terminology, the latter means that we set the x-component of v_p to 1 if $x \in \iota(p)$ and we set it to 0 if $x \notin \iota(p)$. Due to the first two cases of the definition of truth in Section 2, the vector $\mathsf{v}_\top$ represents the set $\{x \in W \mid M, x \models \top\}$ and the vector v_p represents the set $\{x \in W \mid M, x \models p\}$ for every propositional variable p in ϕ. Based on these facts, we then obtain the vector v_ϕ which we are looking for by recursively applying the following properties.

$$\mathsf{v}_{\neg\psi} = \overline{\mathsf{v}_\psi} \qquad \mathsf{v}_{\psi\wedge\rho} = \mathsf{v}_\psi \cap \mathsf{v}_\rho \qquad \mathsf{v}_{\Diamond_i\psi} = R_i;\mathsf{v}_\psi$$

The proofs of these equations for arbitrary ψ and ρ use the remaining three cases of the definition of truth in Section 2 and the definition of relational complement, intersection, and composition. E.g., $\mathsf{v}_{\Diamond_i\psi} = R_i;\mathsf{v}_\psi$ holds since for all $x \in W$

$$\begin{array}{rcl} (R_i;\mathsf{v}_\psi)(x) & \iff & \exists\, y \in W : R_i(x,y) \text{ and } \mathsf{v}_\psi(y) \\ & \iff & \exists\, y \in W : R_i(x,y) \text{ and } M, y \models \psi \\ & \iff & M, x \models \Diamond_i\psi. \end{array}$$

It is obvious from the above equations, how to get the vectors for the constant $\bot$ and the other propositional connectives $\vee$, $\rightarrow$ and $\leftrightarrow$. A little reflection yields the vectors for the dual operators K_i (or $\Box_i$). With the help of the properties of Section 3 we, finally, obtain the vector-representation for the remaining modal operators E_G and C_G, too.

We present only the results for the dual operators K_i and the common knowledge operators E_G, and C_G. Here we have:

$$\mathsf{v}_{K_i\psi} = \overline{R_i;\overline{\mathsf{v}_\psi}} \qquad \mathsf{v}_{E_G\psi} = \overline{(\bigcup_{i\in G} R_i);\overline{\mathsf{v}_\psi}} \qquad \mathsf{v}_{C_G\psi} = \overline{(\bigcup_{i\in G} R_i)^+;\overline{\mathsf{v}_\psi}}$$

A proof of the first equation is

$$\mathsf{v}_{K_i\psi} = \mathsf{v}_{\neg\Diamond_i\neg\psi} = \overline{\mathsf{v}_{\Diamond_i\neg\psi}} = \overline{R_i;\mathsf{v}_{\neg\psi}} = \overline{R_i;\overline{\mathsf{v}_\psi}}\ .$$

The second equation follows from the calculation

$$\mathsf{v}_{E_G\psi} = \mathsf{v}_{\bigwedge_{i\in G} K_i\psi} = \bigcap_{i\in G} \mathsf{v}_{K_i\psi} = \bigcap_{i\in G} \overline{R_i;\overline{\mathsf{v}_\psi}} = \overline{\bigcup_{i\in G} R_i;\overline{\mathsf{v}_\psi}} = \overline{(\bigcup_{i\in G} R_i);\overline{\mathsf{v}_\psi}}\ .$$

A simple induction shows that $\mathsf{v}_{E_G^k\psi} = \overline{(\bigcup_{i\in G} R_i)^k;\overline{\mathsf{v}_\psi}}$ for all $k \geq 1$. This property is used in the following proof of the third equation.

$$\mathsf{v}_{C_G\psi} = \bigcap_{k\geq 1} \mathsf{v}_{E_G^k\psi} = \bigcap_{k\geq 1} \overline{(\bigcup_{i\in G} R_i)^k;\overline{\mathsf{v}_\psi}} = \overline{\bigcup_{k\geq 1}(\bigcup_{i\in G} R_i)^k;\overline{\mathsf{v}_\psi}} = \overline{(\bigcup_{i\in G} R_i)^+;\overline{\mathsf{v}_\psi}}$$

All the constructs of relation algebra we have used up to now are available in the programming language of the RelView tool. More specifically, we have the RelView-operators `-` for complementation (prefix operator), `^` for transposition (postfix operator), `|`, `&`, and `*` for union, intersection, and composition (infix operators), and `trans` for transitive closure (a pre-defined relational function). Furthermore, the tool allows for the definition of relational functions by the user. For instance, the box operators $\Box_i$ can be modelled by the following binary RelView-function `box`.

```
box(S,v) = -(S * -v)
```

Here `S` denotes a RelView-relation (a Boolean matrix) and `v` a RelView-vector. Consider the modal formula ϕ defined as follows.

$$K_1 p \wedge K_1 \neg K_2 K_1 p$$

In words, the formula ϕ says that agent 1 knows p and, furthermore, that agent 1 knows that agent 2 does not know agent 1 knows p. The vector-representation $\mathtt{v}_\phi$ of the set of worlds in which ϕ is true is computed by RelView as the result of the evaluation of the expression

```
box(R1,p) & box(R1,-box(R2,box(R1,p))).
```

Here it is assumed that the accessibility relations R_1, R_2 of M and the vector $\mathtt{v}_p$ are stored in the tool's workspace under the names `R1`, `R2`, and `p`.

5 Example: The Muddy Children Puzzle

By way of the well-known 'muddy children' puzzle we now illustrate the support provided by the RelView tool for solving certain problems on finite models of modal logic. Our description of the puzzle follows [9].

A group of n children play together. A number of them happen to get mud on their foreheads. Each child can see another child's forehead but it cannot see its own forehead. Since no child will tell another child whether it has mud on the forehead, the puzzle is the following. *Can a child know that it has mud on its own forehead?* Obviously, without any extra information the answer is no. But now the father comes onto the scene. He says for all to hear, that 'at least one of you has mud on your forehead'. He then asks the children over and over again: 'Do you know whether you have mud on your forehead?' with the instruction that the children have to answer the question simultaneously. Suppose the number of children with mud on their foreheads is k. Then in the first $k-1$ rounds, the father asks the question all children will answer 'no'. However, in the kth round exactly the children with muddy foreheads will answer 'yes'; the remaining will answer 'no'.

This puzzle can be modelled and solved within the modal logic of knowledge defined in Section 3. The common knowledge operator C_G is particularly crucial for the solution.

As a concrete example of the 'muddy children' puzzle, in the following we elaborate an instance of the problem with three children. The possible states

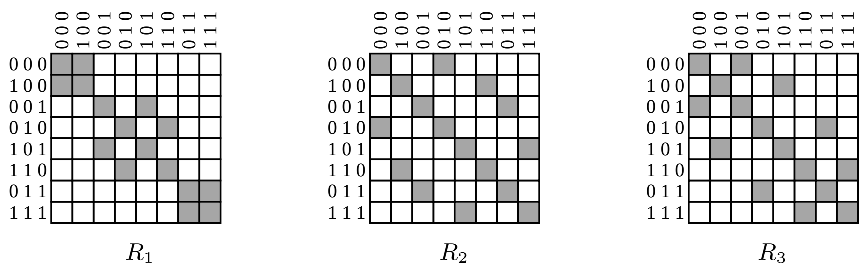

Fig. 2. The accessibility relations in the case of three children

(worlds) of the model are given by triples of 0's and 1's, where (s_1, s_2, s_3) is the state in which child i has mud on its forehead iff $s_i = 1$ and is clean iff $s_i = 0$ $(1 \leq i \leq 3)$. The model, hence, consists of 8 states representing all combinations of associating 0 or 1 with the three children. Let us now consider what each child knows in a given state. For instance, in the state $(1, 0, 1)$ child 1 sees the foreheads of child 2 and child 3 but not its own, it therefore knows that child 2 does not have a muddy forehead but child 3 does. Initially the child does not know if its own head is muddy. Hence, $(0, 0, 1)$ and $(1, 0, 1)$ are the only possible successor states of the state $(1, 0, 1)$ with respect to the accessibility relation R_1. Similar considerations apply to the other children and states.

The three pictures in Figure 2 show the accessibility relations R_1, R_2, and R_3. This is how RelView displays the relations as Boolean matrices (with labeled rows and columns). A black square in the matrix R_i means that the corresponding states are related via this relation and a white square means that they are not related. E.g., the above considerations on the knowledge of child 1 in the state $(1, 0, 1)$ correspond to the two black squares in the fifth row of R_1.

Suppose that the relation R is the union of the three accessibility relations R_1, R_2, and R_3. In Figure 3 it is shown how the RelView tool displays the irreflexive part $R \cap \overline{\mathsf{I}}$ of R as a labelled graph. This graph is the disjoint union of three subgraphs. These correspond to the possibilities of child 1 (boldface arcs), child 2 (dotted arcs), and child 3 (remaining arcs), but neglecting all self-loops. (We have omitted the self-loops in order to avoid cluttering in the graph.)

Now, we assume the propositional variable $p_i, 1 \leq i \leq 3$, denotes that 'child i has mud on its forehead'. Then RelView depicts the vector v_{p_i} representing the set $\iota(p_i)$ as a Boolean column vector as in Figure 4, where we have again used the tool's labeling mechanism to enhance understandability.

The three accessibility relations and these three vectors (Figures 2 and 4) provide a complete specification of the model M which we use as input to RelView. We assume that these are stored in the tool's workspace under the names `R1`, `R2`, `R3` and `p1`, `p2`, `p3`. Furthermore, we use the relational function `box` of Section 3.

In order to determine satisfiability of a formula ϕ in a state or set of states all that is required is to let RelView evaluate the expression corresponding to ϕ, since this returns the set of worlds/states in which the formula is true as

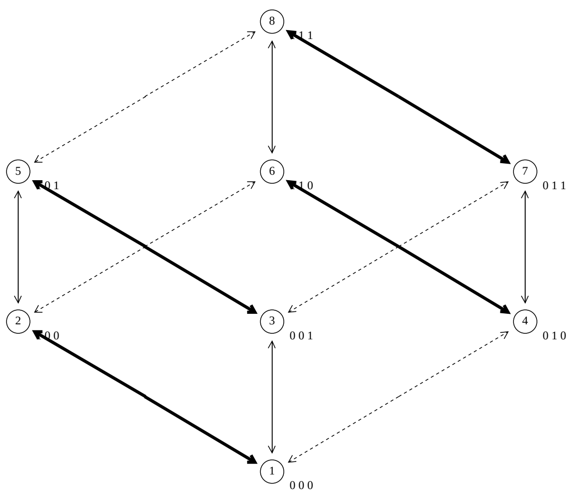

Fig. 3. Graphical representation of the accessibility relations

a vector. As first examples consider the following two statements.

$$M, x \models \neg K_1(p_1 \vee p_2) \qquad M, x \models K_1(p_2 \vee K_3 p_1)$$

The formula on the left says that child 1 does not know whether it or child 2 is muddy and the formula on the right says that child 1 knows that child 2 is muddy or that child 3 knows that child 1 is muddy. The RelView-expressions representing the modal formulae $\neg K_1(p_1 \vee p_2)$ and $K_1(p_2 \vee K_3 p_1)$ are `-box(R1,p1 | p2)` respectively `box(R1,p2 | box(R3,p1))`. Evaluating these two expressions with the tool yields the vectors in Figure 5.

The labelling of the rows is as in Figure 4. Hence, the interpretation of the vectors is that $\neg K_1(p_1 \vee p_2)$ is true in the states $(0, 0, 0)$, $(1, 0, 0)$, $(0, 0, 1)$ and $(1, 0, 1)$ and $K_1(p_2 \vee K_3 p_1)$ is true in all states except $(0, 0, 0)$, $(1, 0, 0)$, $(0, 0, 1)$ and $(1, 0, 1)$. As a consequence, the statement

$$M, x \models K_1(p_1 \vee p_2) \leftrightarrow K_1(p_2 \vee K_3 p_1)$$

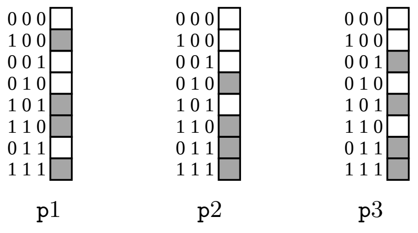

Fig. 4. The vectors for ‘child i has mud on its forehead’

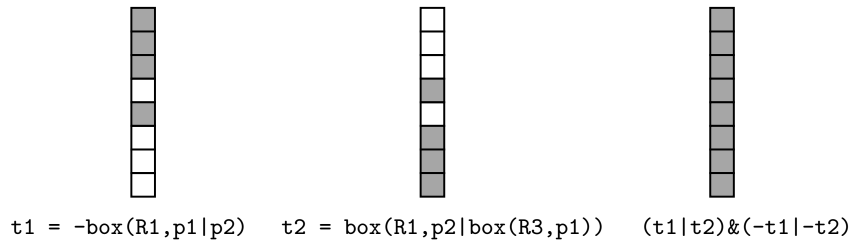

Fig. 5. Satisfiability of $t_1 = \neg K_1(p_1 \vee p_2)$, $t_2 = K_1(p_2 \vee K_3 p_1)$ and $\neg t_1 \leftrightarrow t_2$

is true for all states x of the model M, which menas that its formula is valid in M. This can be easily determined with the aid of RelView by evaluating the expression `(t1 | t2) & (-t1 | -t2)` (i.e. $\neg$`t1` $\leftrightarrow$ `t2`) where `t1` and `t2` denote $\neg K_1(p_1 \vee p_2)$ and $K_1(p_2 \vee K_3 p_1)$, respectively. This produces the universal vector which confirms that the equivalence is valid. In words, the equivalence says that child 1 knows that itself or child 2 is muddy iff it knows that either child 2 is muddy or that child 3 knows that child 1 is muddy.

The next example involves the common knowledge operator C_G. Consider the following statement.

$$M, x \models C_{\{1,2,3\}}(p_2 \rightarrow K_1 p_2)$$

Because $\mathsf{v}_{C_G\psi} = \overline{(\bigcup_{i \in G} R_i)^+ ; \overline{\mathsf{v}_\psi}}$ (see Section 4) and the definition of implication in terms of negation and disjunction, the RelView-expression for the formula $C_{\{1,2,3\}}(p_2 \rightarrow K_1 p_2)$ is

```
box(trans(R1 | R2 | R3),-p2 | box(R1,p2)).
```

The RelView result for this expression is the universal vector, which means that the formula $C_{\{1,2,3\}}(p_2 \rightarrow K_1 p_2)$ holds in all the worlds of the model M under consideration. Indeed, as is easy to verify, in this model it is common knowledge of all children that, if child 2 is muddy then child 1 knows this.

The above illustrates that RelView has two modes for displaying relations: graph representations and matrix representations. Graph representations are particularly well suited for visualization. RelView allows for the edges and nodes of graphs to be distinctively marked. For example, different edge styles can be used as in Figure 3 to specify designated (sub)relations and the nodes can be labelled. Matrix representations are in general less well-suited for visualization, but provide efficient representations of graphs and are easy to process by relation-algebraic (matrix) operations. In addition, certain properties have natural illustrations in matrices. E.g., it is easy to recognize at one glance from the matrices representing R_1, R_2 and R_3 that all three relations are reflexive and symmetric (because each matrix includes the diagonal, the identity relation, and is a mirror image in the diagonal). Also validity of a formula in the model is immediately recognizable when the evaluation returns a vector with all squares marked.

6 Treatment of Other Non-classical Logics

Until now, we have shown how to interpret modal logic of knowledge in relation algebra and how then the RelView tool can be used for investigating finite models of this logic, for visualizing them and for computing solutions to certain computational tasks. This method can be extended to all non-classical logics, embeddable into the programming language of RelView. Prominent examples are logics such as linear-time logic LTL, Hennessy-Milner logic HML, the modal μ-calculus, and the computational tree logic CTL. These are used in computer science for describing properties of computer systems, and model checking for these logics can then serve as a verification method.

In all the logics we have just mentioned some modalities are specified via fixed point constructions. This is no problem for RelView. Far from it! Its programming language allows to formulation of while-loops. These can be used immediately to compute extremal fixed points of monotone functions f on finite lattices as limit of the finite ascending chain $0 \leq f(0) \leq f(f(0)) \leq \ldots$ in the case of the least fixed point (0 is the least element of the lattice) and of the finite descending chain $1 \geq f(1) \geq f(f(1)) \geq \ldots$ in the case of the greatest fixed point (1 is the greatest element of the lattice), respectively.

In the following, we consider computational tree logic CTL in more detail. Formulae of this logic are constructed using the propositional atoms and connectives of modal logic as introduced in Section 2 and the specific operators AX, EX, AU, EU, AF, EF, AG, and EG. The meaning of the operators AX (respectively EX) is the same as the meaning of the $\Box$-modality (respectively the $\Diamond$-modality) in classical modal logic. Hence, if we use again v_ϕ as vector representation of the set $\{M, x \models \phi\}$ we obtain the relation-algebraic specifications

$$\mathsf{v}_{AX(\phi)} = \overline{R;\overline{\mathsf{v}_\phi}} \qquad\qquad \mathsf{v}_{EX(\phi)} = R;\mathsf{v}_\phi,$$

where R is the transition relation of the model M. A formula of the form $AU(\phi, \psi)$ holds in a state x if for *all computation paths* $x_1, x_2, x_3, \ldots$ beginning with $x(= x_1)$ we have that ψ holds in some future state x_i and ϕ holds for all states x_j, $j < i$. Furthermore, a formula $EU(\phi, \psi)$ holds in a state x if there *exists a computation path* $x_1, x_2, x_3, \ldots$ beginning with $x(= x_1)$ such that ψ holds in some future state x_i and ϕ holds in all states x_j, $j < i$. Formally these properties can be described by least fixed point constructions (cf. [21]). These yield the following vector representations, where again R is the transition relation of the model M.

$$\begin{array}{ll} \mathsf{v}_{AU(\phi,\psi)} = \mu_f & \text{where} \;\; f(w) = \mathsf{v}_\psi \cup (\mathsf{v}_\phi \cap \overline{R;\overline{w}} \cap R;\mathsf{L}) \\ \mathsf{v}_{EU(\phi,\psi)} = \mu_g & \text{where} \;\; g(w) = \mathsf{v}_\psi \cup (\mathsf{v}_\phi \cap R;w) \end{array}$$

The remaining four operators can be reduced to AU and EU. We have $AF(\varphi) := AU(\top, \varphi)$, $EF(\varphi) := EU(\top, \varphi)$, $AG(\varphi) := \neg EF(\neg\varphi)$, and $EG(\varphi) := \neg AF(\neg\varphi)$ (see e.g., [21]). From these definitions we obtain the corresponding vector representations as follows:

$$\begin{array}{llll} \mathsf{v}_{AF(\varphi)} = \mathsf{v}_{AU(\top,\varphi)} & \qquad & \mathsf{v}_{AG(\varphi)} & = \overline{\mathsf{v}_{EF(\neg\varphi)}} \\ \mathsf{v}_{EF(\varphi)} = \mathsf{v}_{EU(\top,\varphi)} & \qquad & \mathsf{v}_{EG(\varphi)} & = \overline{\mathsf{v}_{AF(\neg\varphi)}} \end{array}$$

```
AX(R,p) = -(R * -p).

AU(R,p,q)
  DECL w, v
  BEG  w = O(p);
       v = q | (p & -(R * -w) & R * L(p));
       WHILE -eq(w,v) DO
         w = v;
         v = q | (p & -(R * -v) & R * L(p)) OD
       RETURN w
  END.

AF(R,p) = AU(R,L(p),p).
```

Fig. 6. Programs to compute AX, AU, AF.

A RelView-implementation of CTL essentially consists of RelView-programs for the operators of this logic. The code in Figure 6 shows the programs for the three operators AX, AU, and AF as they arise from the above vector representations. Guided by this code the reader should have no difficulties to obtain the RelView-programs for the remaining five CTL-operators EX, EU, EF, AG, and EG from the corresponding vector representations.

We have experimented with a RelView-implementation of CTL using standard examples from the literature. One of them is the 'mutual exclusion' of two processes P_1 and P_2. In the textbook [12] this example is modelled by a transition system in two ways and in each case some important properties (such as safety and liveness) are verified using CTL. The remainder of this section treats the first attempt of [12] with the aid of RelView.

We assume six propositional variables. For $i \in \{1,2\}$ the variable n_i denotes that the process P_i is in a non-critical section, the variable t_i denotes that P_i tries to enter a critical section, and the variable c_i denotes that P_i is in a critical section. Based on these variables, a protocol for managing the admission to a critical section is given by a transition relation R on a set of states and a valuation of the propositional variables. A RelView-description of the protocol is presented in the Figures 7 and 8. Figure 7 shows the transition relation R on the

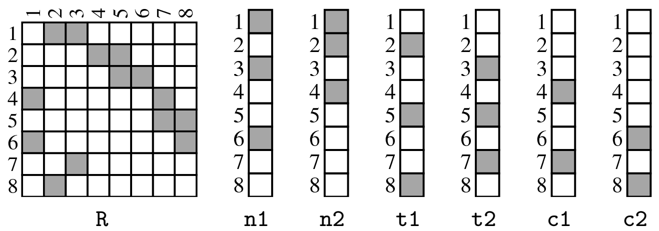

Fig. 7. Relational model of a mutual exclusion protocol

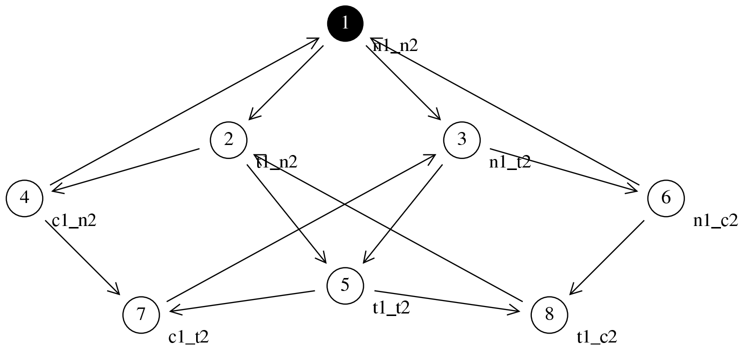

Fig. 8. Graphical representation of a mutual exclusion protocol

protocol's states as a Boolean matrix `R` and the valuation of the propositional variables as six Boolean vectors `n1`, `n2`, `t1`, `t2`, `c1`, and `c2`. The graph representation of the model is shown in Figure 8. In this picture a node corresponds to a state and the labels of a node indicate which propositional variables are defined to be true in the corresponding state. E.g., the first node corresponds to the initial state where both processes are in a non-critical section and the second node corresponds to the state where P_1 tries to enter a critical section and P_2 remains in a non-critical section. Usually, the initial state of a transition system is indicated as a node with an incoming arrow without a source. Since in RelView such 'partial arrows' are not possible, we have drawn the initial node as a black circle.

Having the RelView-description of the protocol at hand, we have used the tool to verify fundamental properties of the protocol. For example, safety, liveness, and that a process can always request to enter a critical section are described by the following three CTL-formulae.

safety:	$AG(\neg(c_1 \wedge c_2))$
liveness:	$AG(t_1 \rightarrow AF(c_1))$
non-blocking:	$AG(n_1 \rightarrow EX(t_1))$

If we evaluate the three corresponding RelView-expressions `AG(R,-(c1 & c2))`, `AG(R,-t1 | AF(R,c1))`, and `AG(R,-n1 | EX(R,t1))`, we obtain in the first case and the third case the 8×1 universal vector and in the second case the 8×1 empty vector. This means that the properties of safety and non-blocking are satisfied in every state but liveness is satisfied in no state. This conclusion is in agreement with the results of [12].

7 Further Uses of RelView

Suppose q is a propositional variable in a modal formula ϕ and suppose the valuations $\iota(p)$ of all propositional variables p in ϕ with the exception of q are defined in M. A problem which we might be interested in is the following:

(5) Compute a valuation $\iota(q)$ to q so that ϕ is satisfiable in a world of M.

Task (5) may be generalized to an optimization problem as follows:

(6) Compute a valuation $\iota(q)$ for q so that ϕ is satisfiable in a maximal number of worlds of M.

A solution to the first problem is possible by applying the 'is-member-of' relation between W and the powerset 2^W. The 'is-member-of' relation ε relates a world x and a set of worlds X iff $x \in X$. It is available in RelView via a pre-defined relational function called `epsi`. Problem (6), the generalization, can also be solved with RelView. The solution uses besides the 'is-member-of' relation also the 'size-comparison' relation on 2^W, and the vector-representation of greatest elements with respect to a quasi-order. The 'size-comparison' relation relates two sets X and Y iff $|X| \leq |Y|$ and can be computed via a call of the pre-defined function `cardrel`.

In an array-like implementation of relations the memory consumption of the 'is-member-of' relation and the 'size-comparison' relation is exponential in the size of the base set. However, BDDs allow a very efficient implementation of these two relations. In [17] for the 'is-member-of' relation a BDD-implementation is developed that uses $O(n)$ BDD-nodes and [20] presents for the 'size-comparison' relation a BDD-implementation with $O(n^2)$ BDD-nodes. In both cases n is the number of elements of the base set, i.e., the cardinality of the set of worlds W in our case.

To give an impression of how to solve problem (5) by means of RelView, we consider the formula $\neg K_1(p_1 \vee p_2)$ of Section 5 and replace the propositional variable p_2 by the (uninterpreted) propositional variable q. We assume again that the relation R_1 is as shown in Figure 2 and that the propositional variable p_1 denotes 'child 1 has mud on its forehead', i.e., the vector representation v_{p_1} of $\iota(p_1)$ is as shown in Figure 4. Then RelView computes exactly 240 possible valuations $\iota(q)$ for q such that the modal formula

$$\neg K_1(p_1 \vee q)$$

becomes true in a world of the model M with relation R_1 and valuation function ι. The key to obtaining this result is the relation Q between the set of worlds W and the powerset 2^W, defined by

$$Q := R_1; \overline{\mathsf{v}_{p_1}; \mathsf{L} \cup \varepsilon}\,.$$

This definition implies that for all $x \in W$ and $X \in 2^W$ we have that $Q(x, X)$ iff $X = \iota(q)$ implies $M, x \models \neg K_1(p_1 \vee q)$. In matrix terminology this means: If $\iota(q)$ is represented by column c of ε, then $\{x \in W \mid M, x \models \neg K_1(p_1 \vee q)\}$ is represented by the same column of Q. Hence, the vector $Q^\mathsf{T}; \mathsf{L}$ (defined over 2^W) represents the 240 solutions of problem (5) with inputs $\neg K_1(p_1 \vee q)$, R_1, and $\iota(p_1)$. A column-wise description of these solutions is $\varepsilon; \mathit{inj}(Q^\mathsf{T}; \mathsf{L})^\mathsf{T}$, where the

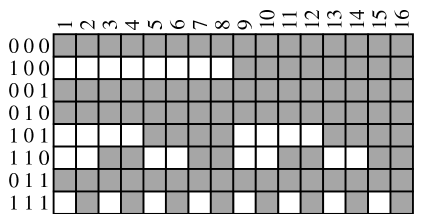

Fig. 9. Valuations not leading to satisfiability of $\neg K_1(p_1 \vee q)$

relational function *inj* computes the injective mapping generated by a vector. (If the vector v over X represents the subset Y of X, then *inj*(v) is the relation between Y and X such that *inj*$(v)(x, y)$ iff $x = y$.) This standard technique for representing sets of subsets is explained in, e.g. [3, 4]. In our example it yields a 8×240 RelView-matrix, which is too large to be presented here. Therefore, we show in Figure 9 a much smaller RelView-matrix that column-wisely represents the non-solutions, i.e., the 16 valuations $\iota(q)$ for q which do not lead to satisfiability. For example, from the last column of this picture we see that no world of M satisfies the formula $\neg K_1(p_1 \vee q)$ if ι defines the variable q to be true in all worlds of M.

We have also used RelView to solve problem (6) for the same three inputs $\neg K_1(p_1 \vee q)$, R_1, and $\iota(p_1)$. The system computes that exactly 16 of the 240 solutions of problem (5) maximize the number of worlds which satisfy the formula, there is only one such maximal set of worlds, and its cardinality is 4. The 16 solutions of problem (6) are column-wisely described by the 8×16 matrix of Figure 10. E.g., the last column of this matrix states that $\neg K_1(p_1 \vee q)$ is true in a maximal number of worlds if q is true in $\langle 1, 0, 0\rangle$, $\langle 1, 0, 1\rangle$, $\langle 1, 1, 0\rangle$, and $\langle 1, 1, 1\rangle$. The only 4 worlds which satisfy $\neg K_1(p_1 \vee q)$, if $\iota(q)$ is one of the 16 solutions of problem (6), are $\langle 1, 0, 0\rangle$, $\langle 1, 0, 1\rangle$, $\langle 1, 1, 0\rangle$, and $\langle 1, 1, 1\rangle$. This property follows from the RelView-vector of Figure 10.

Like the solutions (respectively non-solutions) of problem (5) for the inputs $\neg K_1(p_1 \vee q)$, R_1, and $\iota(p_1)$, also the solutions of problem (6) can be specified by simple relation-algebraic expressions. Crucial to the solution is the vector

$$v := ge(C, syq(\varepsilon, Q); \mathsf{L})$$

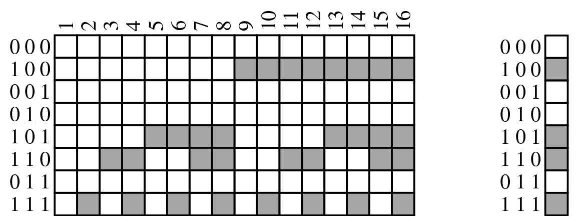

Fig. 10. Maximum satisfiability of $\neg K_1(p_1 \vee q)$

over the powerset 2^W that represents the set of all maximal subsets X of W such that $M, x \models \neg K_1(p_1 \vee q)$ holds for all $x \in X$. In this definition the relations Q and ε are as above, C denotes the 'size-comparison' relation on 2^W, and the relational functions

$$ge(R, w) = w \cap \overline{\overline{R}^{\mathsf{T}}; w} \qquad syq(R, S) = \overline{R^{\mathsf{T}}; \overline{S}} \cap \overline{\overline{R}^{\mathsf{T}}; S}$$

compute the vector of the greatest elements of the vector w with respect to the quasi-order R and the symmetric quotient of R and S, respectively. In the present case the column-wise description $\varepsilon; inj(v)^{\mathsf{T}}$ of the maximal subsets consists of only one column and coincides with the vector of Figure 10. From it we obtain the vector representation of the set of 16 valuations leading to the only maximal subset[2] $\{\langle 1,0,0\rangle, \langle 1,0,1\rangle, \langle 1,1,0\rangle, \langle 1,1,1\rangle\}$ via the vector $syq(Q, \varepsilon; inj(v)^{\mathsf{T}})$ over 2^W, and the 8×16 matrix of Figure 10, finally, is exactly the column-wise description of this set of valuations.

So far we have used RelView only for computing sets of worlds or for solving related tasks. But the application domain of the system is larger. For example, the tool can also be used for the following important task:

(7) Determine whether a relation R in a given finite frame possesses certain properties.

The kinds of properties RelView can express and handle are rather general. In particular, these are all properties which can be written as Boolean combinations of inclusions between relation-algebraic expressions. This includes all the correspondence properties of Section 2 (reflexivity of a relation R, transitivity or R, etc), and also properties such as irreflexivity ($\overline{\mathsf{I}} \subseteq R$) and acyclicity ($R^+ \subseteq \overline{\mathsf{I}}$) as well as Boolean combinations of these. For example, R is an equivalence relation iff it satisfies the conjunction of the first three correspondence properties of Section 2. In the syntax of the RelView tool a corresponding evaluation test looks as follows:

```
incl(I(R),R) & incl(R*R,R) & incl(R,R^).
```

Let us consider a last application. For a given finite frame $\mathcal{F}$ with set of worlds W and a closure system[3] $\mathcal{C} \subseteq 2^{W \times W}$ of relations (like the Euclidean or the transitive relations), the RelView tool very often allows us to solve the following task:

(8) Compute the corresponding closure operator $\mathrm{cl} : 2^{W \times W} \to 2^{W \times W}$, defined by $\mathrm{cl}(R) = \bigcap\{S \in \mathcal{C} \mid R \subseteq S\}$.

The condition which is to be fulfilled is that the conjunction of $S \in \mathcal{C}$ and $R \subseteq S$ is equivalent to $f(S) \subseteq S$, with f being a monotone function on the

[2] In words, this vector marks exactly the 16 columns of Q each of which represent a set of worlds with the maximal cardinality 4.

[3] A subset $\mathcal{C}$ of a powerset 2^X is a closure system on X if $\bigcap Y \in \mathcal{C}$ for all $Y \subseteq \mathcal{C}$.

```
euclid(R)
  DECL S, fS
  BEG  S = O(R);
       fS = R;
       WHILE -eq(S,fS) DO
         S = fS;
         fS = R | fS^ * fS OD
       RETURN S
  END.
```

Fig. 11. Program to compute the Euclidean closure of a relation R

set $2^{W \times W}$ of all relations over W. In this case $\mathrm{cl}(R)$ coincides with the least fixed point μ_f of the function f, due to Tarski's fixed point theorem [27]. The frame $\mathcal{F}$ is finite. Hence f is even $\cup$-continuous and we get the representation $\mu_f = \bigcup_{i \geq 0} f^i(\mathsf{O})$, where the chain $\mathsf{O} \subseteq f(\mathsf{O}) \subseteq f^2(\mathsf{O}) \subseteq \ldots$ eventually becomes stationary. To give an example, the Euclidean closure of a relation R is computed by the RelView-program `euclid` of Figure 11, because obviously a relation S is Euclidean (i.e.,$S^{\mathsf{T}}; S \subseteq S$) and contains R iff $R \cup S^{\mathsf{T}}; S \subseteq S$.

Finally, it is worth mentioning that RelView has some file input/output interfaces. Especially ASCII formatted files can be used to exchange data with other systems.

8 Concluding Remarks

Based on the interpretation of non-classical logics in relation algebra, in this paper we have shown how the RelView tool can be used for investigating finite models of such logics and for visualizing them and solutions of certain computational tasks. Modal logic of knowledge and computational tree logic have been treated in detail and illustrated with two well-known examples, viz. the 'muddy children' puzzle and a 'mutual exclusion' protocol.

We believe that the attraction of RelView in respect to the applications we have discussed in this paper lies in its flexibility, the concise form of its programs, and the various possibilities for manipulation, testing, and visualization. Because of these properties it is an excellent tool for prototyping, experimenting, and for university teaching. It can be programmed to handle different logics and perform typical tasks on them while avoiding unnecessary overhead. We found it very attractive to use RelView also for producing good examples. Concerning teaching, its visualization possibilities can be used to demonstrate the meaning of logical operators and formulae for example.

To illustrate this point, consider the picture in Figure 12. It explains the meaning of the *AF*-operator of the logic CTL. The squares denote the states where a certain property, p say, holds and the black vertices (including the squares) denote the states x such that for all computation paths $x_1, x_2, \ldots$ beginning in x somewhere along the path p holds. Visualization is of particular

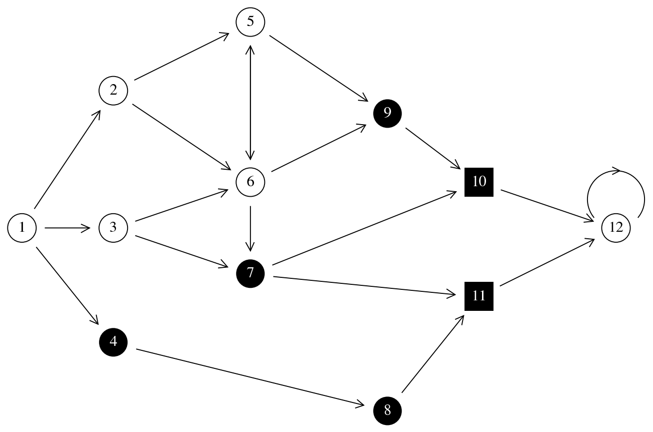

Fig. 12. Visualization of the meaning of the *AF*-operator

importance when combined with the evaluation of RelView-expressions in a stepwise fashion. All this can help students, and even be key to their fully understanding of an advanced concept.

References

1. R. Behnke, R. Berghammer, E. Meyer, and P. Schneider. RelView - A system for calculating with relations and relational programming. In E. Astesiano, editor, *Fundamental Approaches to Software Engineering*, volume 1382 of *LNCS*, pages 318–321. Springer, Berlin, 1998.
2. R. Berghammer, P. Kempf, G. Schmidt, T. Ströhlein. Relational algebra and logic of programs. In H. Andréka, J.D. Monk, and I. Németi, editors, *Algebraic Logic*, volume 54 of *Colloq. Math. Soc. J. Bolyai*, pages 37–58. North-Holland, Amsterdam, 1991.
3. R. Berghammer, B. Leoniuk, and U. Milanese. Implementation of relational algebra using binary decision diagrams. In H. de Swart, editor, *Relational Methods in Computer Science*, volume 2561 of *LNCS*, pages 241–257. Springer, Berlin, 2002.
4. R. Berghammer, F. Neumann. RelView – An OBDD-based Computer Algebra system for relations. In V.G. Gansha, E.W. Mayr, and E.V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, volume 3718 of *LNCS*, pages 40–51. Springer, Berlin, 2005.
5. P. Blackburn, M. de Rijke, and V. Venema. *Modal Logic.* Cambridge Univ. Press, Cambridge, 2001.
6. C. Brink, K. Britz, and R. A. Schmidt. Peirce algebras. *Formal Aspects of Computing*, 6(3):339–358, 1994.
7. B. F. Chellas. *Modal Logic: An Introduction.* Cambridge Univ. Press, Cambridge, 1980.
8. E. Clarke, O. Grumberg and D. Peled. *Modal Checking.* MIT Press, Cambridge, 2000.
9. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about knowledge.* MIT Press, Cambridge, 1995.

10. R. Goldblatt. *Logics of Time and Computation*, volume 7 of *CSLI Lecture Notes*. Chicago Univ. Press, Chicago, 1987.
11. G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, London, 1996.
12. M. Huth and M. Ryan. *Logic in Computer Science. Modelling and Reasoning About Systems*. Cambridge Univ. Press, Cambridge, 2000.
13. B. Jónsson and A. Tarski. Boolean algebras with operators, Part I. *American Journal of Mathematics*, 73:891–939, 1951.
14. W. Nabialek, A. Niewiadomski, W. Penczek, A. Polrola, and M. Szreter. VERICS 2004: A model checker for real time and multi-agent systems. In *Proc. International Workshop on Concurrency, Specification, and Programming*, volume 170 of Informatik-Berichte, pages 88-99, Humbold Universität Berlin, 2004.
15. M. Kacprzak, A. Lomusico, A. Niewiadomski, M. Szreter, W. Penczek, and F. Raimondi. Comparing BDD and SAT based techniques for model checking Chaum's Dining Cryptographers protocol. To appear in *Fundamenta Informaticae*, 2006.
16. D. Kozen. A representation theorem for models of ∗-free PDL. In J. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *LNCS*, pages 351–362. Springer, Berlin, 1980.
17. B. Leoniuk. ROBDD-based implementation of relational algebra with applications (in German). Ph.D. thesis, Institut für Informatik und Praktische Mathematik, Universität Kiel, 2001.
18. A. Lomuscio and F. Raimondi. MCMAS: a tool for verifying multi-agent systems. In H. Hermanns and J. Palsberg, editors, *Tools and Algorithms for the Aonstruction and Symposium of Systems*, volume 3920 of *LNCS*, pages 450–454. Springer, Berlin, 2006.
19. J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge Univ. Press, Cambridge, 1995.
20. U. Milanese. On the implementation of a ROBDD-based tool for the manipulation and visualization of relations (in German). Ph.D. thesis, Institut für Informatik und Praktische Mathematik, Universität Kiel, 2003.
21. M. Müller-Olm, D. Schmidt, B. Steffen. Model-checking: A tutorial introduction. In A. Cortesi and G. Filé, editors, *Static Analysis Symposium 1999*, volume 1694 of *LNCS*, pages 330–354. Springer, Berlin, 1999.
22. K. C. Ng and A. Tarski. Relation algebras with transitive closure, abstract 742-02-09. *Notices Amer. Math. Soc.*, A29–A30, 1977.
23. E. Orlowska. Relational formalisation of nonclassical logics. In C. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, Advances in Computing, pages 90–105. Springer, Wien, 1997.
24. V. R. Pratt. Dynamic algebras: Examples, constructions, applications. Technical Report MIT/LCS/TM-138, MIT Laboratory for Computer Science, 1979.
25. G. Schmidt and T. Ströhlein. *Relations and Graphs. Discrete Mathematics for Computer Scientists*. Springer, Berlin, 1993.
26. A. Tarski. On the calculus of relations. *J. Symbolic Logic*, 6(3):73–89, 1941.
27. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.