Model Checking *Is* Static Analysis of Modal Logic

Flemming Nielson and Hanne Riis Nielson

DTU Informatics, Technical University of Denmark
{nielson,riis}@imm.dtu.dk

Abstract. Flow Logic is an approach to the static analysis of programs that has been developed for functional, imperative and object-oriented programming languages and for concurrent, distributed, mobile and cryptographic process calculi. In this paper we extend it to deal with modal logics and prove that it can give an exact characterisation of the semantics of formulae in a modal logic. This shows that model checking can be performed by means of state-of-the-art approaches to static analysis and allow us to conclude that the problems of model checking and static analysis are reducible to each other. In terms of computational complexity we show that model checking by means of static analysis gives the same complexity bounds as are known for traditional approaches to model checking.

1 Introduction

Model checking [10,2] is a successful approach to the validation of properties expressed in modal logics with respect to models expressed as transition systems. The transitions may originate from descriptions of hardware or more recently of software systems. Much work focuses on transition systems that are finite and have no structured data although extensions to infinite systems and structured data (e.g. allowing cryptographic terms) exist.

Static analysis [9,13] is in a similar way a successful approach to the validation of properties of programming languages. Originally used in the development of compilers it has spread to uses in editors in software development environments, program validation, program understanding and is also being applied to distributed formalisms such as process calculi. A number of "schools" exist, including Data Flow Analysis, Type and Effect Systems, and Abstract Interpretation to name but a few.

Flow Logic [17] is a particular approach to static analysis that borrows methods and techniques from Abstract Interpretation, Data Flow Analysis and Constraint Based Analysis while presenting the analysis in a style more reminiscent of Type Systems. One of the hallmarks of Flow Logic it that it makes a clear distinction between *(i)* the specification of the analysis, *(ii)* whether or not a proposed analysis result is indeed correct with respect to the semantics, and *(iii)* the computation of the best analysis result. The logical format used for

L. Ong (Ed.): FOSSACS 2010, LNCS 6014, pp. 191–205, 2010.

presenting specifications focuses on ensuring the implementability of the analyses — often this is possible in low polynomial (cubic) time. Over the years Flow Logic has proved to be a robust approach able to deal with a wide variety of programming paradigms (e.g. [17]) and calculi of computation (e.g. [4,14,18]).

The problem. The interplay between model checking and static analysis has intrigued many researchers for many years. Early comparisons focused on static analyses over-approximating the solution set while model checking similarly under-approximating the solution set (in case of non-termination). Successive enhancements within model checking and static analysis add to the complexity of understanding their interplay.

A number of papers have taken the view that static analysis *is* model checking of formulae in suitable modal logics. Possibly the first paper is [22] that focused on how to understand the construction of complex data flow equations through their characterisation in a modal μ-calculus. In [20] it is shown how abstract interpretation can be used to cast further light on the construction and this is extended in [21] where an Action Computation Tree Logic is used to express the logical properties. Finally, [11] describe a Java based software system utilising some of these ideas.

The contribution. In this paper we add to the conceptual understanding by showing that model checking really amounts to a static analysis of the modal formulae. To be concrete we choose an *Action Computation Tree Logic* (ACTL [12]) and develop a static analysis in the Flow Logic approach by means of *Alternation-free Least Fixed Point Logic* (ALFP [15]) as used in the Succinct Solver [16] — indeed this is the first paper to extend Flow Logic to deal with modal logic.

At the conceptual level we believe that the developments of Steffen and Schmidt on the one hand, and our present contribution on the other, jointly show the close relationship between model checking and static analysis – to the extent that they seem to be reducible to each other. Clearly we cannot formulate this as precisely as reducibility among computational problems for the good reason that the notions of model checking and static analysis have not been clearly formalised and there may even be lack of consensus on where the exact boundaries are.

Overview. In Section 2 we give the necessary background information on labelled transition systems and Action Computation Tree Logic. Section 3 then presents the essence of the Flow Logic approach to static analysis including the Alternation-free Least Fixed Point Logic that often plays a key role in the implementations of Flow Logics. Our main contribution is in Section 4 where we present a precise static analysis of Action Computational Tree Logic within Flow Logic and prove its correctness – thereby giving further insights on the relationship between model checking and static analysis. We conclude in Section 5.

2 Modal Logic

Labelled Transition Systems. A *labelled transition system* (LTS) has the form $(S, \mathcal{A}, \rightarrow)$ where S is a non-empty set of states, $\mathcal{A}$ is a non-empty set of actions and $\rightarrow \subseteq S \times \mathcal{A} \times S$ is the transition relation. We shall write $s \rightarrow^a s'$ whenever $(s, a, s') \in \rightarrow$. A transition system is *finite* whenever both S and $\mathcal{A}$ are finite. A transition system is *finitely branching* whenever the sets $\mathsf{sp}(s) = \{(a, s') \mid s \rightarrow^a s'\}$ are finite for all choices of s; clearly a finite transition system is also finitely branching.

A *path* π is a *maximal* sequence $(s_i \rightarrow^{a_i} s_{i+1})_i$ such that $s_i \rightarrow^{a_i} s_{i+1}$ for all $i \geq 0$. This means that a path is either infinite or ends in a state that is *stuck*; the latter means that there are no outgoing transitions for any action. Sometimes we write the path in the form $(s_i \rightarrow^{a_i} s_{i+1})_{0 \leq i < n}$ to make it clear that $i \geq 0$ and that the path has length $n \in \{0, \cdots, \infty\}$; it follows that if $n \neq \infty$ then s_n is stuck. If s_0 is a stuck state then there is exactly one path $(s_i \rightarrow^{a_i} s_{i+1})_i$ starting in s_0, namely the empty path.

One can arrange to avoid stuck states by adding a new state s_{stuck} and ensure that all stuck states, as well as s_{stuck}, have a transition to s_{stuck} for some possibly new action; in this case all paths will be infinite and this is the usual assumption in model checking where transition systems are supposed to be Kripke structures. However, this is not usually the case for transition systems generated from programming languages or process calculi; consequently we have opted for a treatment where a path may end in a stuck state.

Action Computation Tree Logic. We shall use a variant of the modal logic *Action Computation Tree Logic* (ACTL) [12] to express properties of paths in labelled transition systems. It is defined by:

$$\begin{array}{rl} \phi ::= & \mathsf{true} \mid \mathsf{false} \mid bp \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \Rightarrow \phi_2 \\ \mid & \mathbf{EX}_\Omega\ \phi \mid \mathbf{AX}_\Omega\ \phi \mid \mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] \mid \mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] \end{array}$$

The subscripts $\Omega \subseteq \mathcal{A}$ are sets of actions used to restrict the transitions taken. The base predicates bp (so far left unspecified) denote sets of states used to restrict the target state of the transitions taken. The **E** modality quantifies over the existence of paths, the **A** modality quantifies over all paths, **X** focuses on the next step and **U** is an until modality; we have chosen[1] a version of **U** that focuses on eventually taking an action in a certain set Ω_2 to reach a state satisfying a certain property ϕ_2. The interpretation of the modal operators is given in Table 1; when expressing the interpretation of a formula in the state s_0 and next choosing a path $(s_i \rightarrow^{a_i} s_{i+1})_{0 \leq i < n}$ it is intended that the state s_0 is indeed the first state in the path $(s_i \rightarrow^{a_i} s_{i+1})_{0 \leq i < n}$; we use this convention throughout the paper.

[1] This means that we do *not* obtain CTL by merely setting all Ω, Ω_1, Ω_2 to $\mathcal{A}$. Other choices of **U** are clearly possible without jeopardising our development. The present choice is from [8].

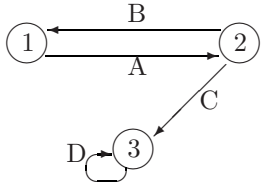
Table 1. Satisfaction relation for modal operators of ACTL: $s \models \phi$

$$
\begin{array}{rcl}
s_0 \models \mathbf{EX}_\Omega\ \phi & \underline{\text{iff}} & \exists (s_i \rightarrow^{a_i} s_{i+1})_{0 \le i < n} : n > 0 \wedge a_0 \in \Omega \wedge s_1 \models \phi \\
s_0 \models \mathbf{AX}_\Omega\ \phi & \underline{\text{iff}} & \forall (s_i \rightarrow^{a_i} s_{i+1})_{0 \le i < n} : n > 0 \wedge a_0 \in \Omega \wedge s_1 \models \phi \\
s_0 \models \mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] & \underline{\text{iff}} & \exists (s_i \rightarrow^{a_i} s_{i+1})_{0 \le i < n} : \exists k < n : \\
 & & [\bigwedge_{0 \le i < k} (a_i \in \Omega_1 \wedge s_{i+1} \models \phi_1) \wedge (a_k \in \Omega_2 \wedge s_{k+1} \models \phi_2)] \\
s_0 \models \mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2] & \underline{\text{iff}} & \forall (s_i \rightarrow^{a_i} s_{i+1})_{0 \le i < n} : \exists k < n : \\
 & & [\bigwedge_{0 \le i < k} (a_i \in \Omega_1 \wedge s_{i+1} \models \phi_1) \wedge (a_k \in \Omega_2 \wedge s_{k+1} \models \phi_2)]
\end{array}
$$

The modalities **AF**, **EF**, **AG** and **EG** are derivable in this logic[2] in the standard way. The **AG** modality is of special interest to us; its derived interpretations is:

$$s_0 \models \mathbf{AG}_\Omega\ \phi \quad \underline{\text{iff}} \quad \forall (s_i \rightarrow^{a_i} s_{i+1})_{0 \le i < n} : \bigwedge_{0 \le i < n} [a_i \in \Omega \Rightarrow s_{i+1} \models \phi]$$

Example 1. Consider a transition system with states $S = \{1, 2, 3\}$, actions $\mathcal{A} = \{A, B, C, D\}$ and transition relation $\rightarrow$ given by the diagram to the left:



ϕ	$\{s \mid s \models \phi\}$
$\mathbf{EX}_{\mathcal{A}}$ goal	$\{2, 3\}$
$\mathbf{AX}_{\mathcal{A}}$ goal	$\{3\}$
$\mathbf{E}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]$	$\{1, 2, 3\}$
$\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]$	$\{3\}$
$\mathbf{AG}_{\mathcal{A}}$ goal	$\{3\}$
$\mathbf{AG}_{\{C\}}$ goal	$\{1, 2, 3\}$

The table to the right shows the validity of some formulae when goal is the base predicate that is only satisfied in the state 3. □

Model Checking. Model checking is the problem of obtaining efficient ways of computing the set $\mathsf{mc}_\phi = \{s \mid s \models \phi\}$ of states that satisfy a given modal formula. In global model checking these algorithms usually proceed in a syntax directed manner on the formula ϕ where it is assumed that the sets of satisfying states have been computed for all sub-formulae. For the base clauses the construction is immediate; for example, $\mathsf{mc}_{\mathsf{true}} = S$ and $\mathsf{mc}_{\mathsf{false}} = \emptyset$. For the clauses involving propositional operators the construction is rather straightforward; for example, $\mathsf{mc}_{\phi_1 \wedge \phi_2} = \mathsf{mc}_{\phi_1} \cap \mathsf{mc}_{\phi_2}$ and $\mathsf{mc}_{\phi_1 \Rightarrow \phi_2} = \mathsf{mc}_{\phi_2} \cup (S \setminus \mathsf{mc}_{\phi_1})$. For the modal operators the construction is somewhat more complex and we refer to standard textbooks like [2] for an overview of existing methods and techniques.

The worst case time complexity of model checking an ACTL formula ϕ of size $|\phi|$ is $\mathcal{O}((|T| + |S|)\,|\phi|)$ where the transition system $(S, \mathcal{A}, \rightarrow)$ has a state space of size $|S|$ and a transition relation $\rightarrow$ of size $|T|$ and where we assume that the number of base predicates and actions are constant. This result has been adapted from the information given in [2] about the time complexity of model checking CTL (which essentially is ACTL without consideration about the actions).

[2] As above we do not get the CTL equivalents simply by setting Ω to $\mathcal{A}$.

3 Static Analysis

Flow Logic. The aim of static analysis is to estimate the computational behaviour of programs or systems. The idea is to capture the information of interest by elements of complete lattices; the details of the complete lattices depend on the actual property of interest. In the Flow Logic approach *logical judgements* are used to *specify* when the analysis information correctly captures the information of the program or system. The judgements are defined by a set of rules that, in general, must be interpreted co-inductively; for syntax-directed definitions this coincides with the traditional inductive interpretations.

The *correctness* of the analysis is often established as a subject reduction result – much as one will do for a type system. To ensure that the analysis is *implementable* it is customary to establish a Moore family, or model intersection property. This is one of the important points where Flow Logic distinguishes itself from traditional type systems as it ensures that we can determine a best analysis result for all programs – at the same time the result provides a strong link between Flow Logic and Abstract Interpretation.

Example 2. To illustrate the approach we shall review a small example based on the λ-calculus; we refer to [17] for more details. So let the expressions $e \in \mathsf{Exp}$ be given by

$$e ::= x \mid \lambda x.e \mid e_1\, e_2$$

where $x \in \mathsf{Var}$ denotes a variable. The notion of free variables $\mathsf{fv}(e)$ are defined as usual and we shall write $v \in \mathsf{Val}$ for the set of expressions given by $v ::= x \mid \lambda x.e$.

We shall assume that the semantics is given by a labelled transition system where $S = \mathsf{Exp}$ is the set of states, $\mathcal{A} = \mathsf{Val} \times \mathsf{Val}$ is the set of actions (recording the function being applied and its argument) and the transition relation $\rightarrow$ is given by the following axioms and rules

$$(\lambda x.e)\, v \rightarrow^{(\lambda x.e,v)} e[v/x] \qquad \frac{e_1 \rightarrow^{\beta} e_1'}{e_1\ e_2 \rightarrow^{\beta} e_1'\ e_2} \qquad \frac{e_2 \rightarrow^{\beta} e_2'}{e_1\ e_2 \rightarrow^{\beta} e_1\ e_2'}$$

where $e[v/x]$ is the expression obtained by replacing all free occurrences of x in e by v. Note that the actions are used to record the redex of the evaluation step.

A typical analysis for the λ-calculus is a *control flow analysis*. The aim is to approximate the potential values for each sub-expression and to do so it is necessary also to approximate the potential values of the variables. In order to obtain an analysis that is as precise as possible we shall assume that the variables have been α-renamed apart and furthermore we shall assign unique labels $\ell \in \mathsf{Lab}$ to each sub-expression of the expression of interest; thus expressions (and values) now take the form e^{ℓ}. The labels only serve as pointers into the syntax; they simply allow us to pinpoint the various sub-expressions so that we can speak about their values and thus have no semantic significance. So in particular we have

$$((\lambda x.e^{\ell'})^{\ell_1}\, v^{\ell_2})^{\ell} \rightarrow^{((\lambda x.e^{\ell'})^{\ell_1}, v^{\ell_2})} (e[v/x])^{\ell}$$

Table 2. Flow Logic for the λ-calculus: $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$

$$
\begin{array}{rcl}
\mathcal{C}, \mathcal{R} \vdash x^{\ell} & \underline{\text{iff}} & \mathcal{R}(x) \subseteq \mathcal{C}(\ell) \\
\mathcal{C}, \mathcal{R} \vdash (\lambda x.e^{\ell'})^{\ell} & \underline{\text{iff}} & (\mathcal{C}, \mathcal{R} \vdash e^{\ell'}) \ \wedge \ (\lambda x.e^{\ell'}) \in \mathcal{C}(\ell) \\
\mathcal{C}, \mathcal{R} \vdash (e_1^{\ell_1}\ e_2^{\ell_2})^{\ell} & \underline{\text{iff}} & (\mathcal{C}, \mathcal{R} \vdash e_1^{\ell_1}) \ \wedge \ (\mathcal{C}, \mathcal{R} \vdash e_2^{\ell_2}) \ \wedge \\
 & & \forall (\lambda x.e_0^{\ell_0}) \in \mathcal{C}(\ell_1) : (\mathcal{C}(\ell_2) \subseteq \mathcal{R}(x) \wedge \mathcal{C}(\ell_0) \subseteq \mathcal{C}(\ell))
\end{array}
$$

The two complete lattices of interest in our analysis are:

- $\mathcal{C} : \mathsf{Lab} \to \mathcal{P}(\mathsf{Val})$: the expression labelled ℓ may evaluate to a value in $\mathcal{C}(\ell)$,
- $\mathcal{R} : \mathsf{Var} \to \mathcal{P}(\mathsf{Val})$: the variable x may evaluate to a value in $\mathcal{R}(x)$.

The analysis is then defined by judgements of the form

$$\mathcal{C}, \mathcal{R} \vdash e^{\ell}$$

as shown in Table 2. The first clause expresses that any value of x is also a possible value of the expression x^{ℓ} and hence any information contained in $\mathcal{R}$ about x must also be contained in $\mathcal{C}(\ell)$. The second clause insists that the body of the λ-expression must be analysable and additionally that the λ-expression itself is a possible value of the expression $(\lambda x.e^{\ell'})^{\ell}$. The clause for function application first insists that the operator as well as the operand must be analysable. Additionally it says that if the operator (i.e. $e_1^{\ell_1}$) evaluates to a λ-expression with formal parameter x and body $e_0^{\ell_0}$ then any value that the actual parameter (i.e. the operand $e_2^{\ell_2}$) might evaluate to is also a possible value of x and any value that the body $e_0^{\ell_0}$ might evaluate to is also a possible value of the overall function application $(e_1^{\ell_1}\ e_2^{\ell_2})^{\ell}$.

Semantic correctness of this simple analysis amounts to ensuring that the judgement $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$ correctly captures the evaluation of the λ-expression e. We can express this as a *subject reduction result*:

Proposition. If $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$ and $e^{\ell} \to^{\beta} e'^{\ell}$ then $\mathcal{C}, \mathcal{R} \vdash e'^{\ell}$.

We can also express the *adequacy* of the analysis by stating that $\mathcal{C}$ correctly captures the function applications taking place:

Proposition. If $\mathcal{C}, \mathcal{R} \vdash e^{\ell}$ and $e^{\ell} \to^{(v_1^{\ell_1}, v_2^{\ell_2})} e'^{\ell}$ then $v_1 \in \mathcal{C}(\ell_1)$ and $v_2 \in \mathcal{C}(\ell_2)$.

Linking back to ACTL we can use the above propositions to show that

$$\mathbf{AG}_{\{(v_1^{\ell_1}, v_2^{\ell_2})\}} \ (v_1 \in \mathcal{C}(\ell_1) \wedge v_2 \in \mathcal{C}(\ell_2))$$

is a true formula that describes the overall adequacy of the analysis. It is observations like this one that has lead some researchers to suggest that static analysis is an instance of model checking [22,20,21].

In addition to proving semantic correctness of the analysis we shall want it to satisfy a *Moore family*, or a *model intersection* property. Recall, that a Moore family is a subset of a complete lattice that is closed under greatest lower bounds.

Proposition. $\{(\mathcal{C}, \mathcal{R}) \mid \mathcal{C}, \mathcal{R} \vdash e^{\ell}\}$ is a Moore family for all e^{ℓ}.
This result ensures that all expressions can be analysed and have a least, or best, analysis result – this links into the framework of Abstract Interpretation. □

Alternation-free Least Fixed Point Logic. The Moore family result guarantees the existence of best analysis results but in itself it does not provide any mechanism for constructing the analysis result. Here *Alternation-free Least Fixed Point Logic* (ALFP [15]) has proved very useful for obtaining efficient implementations. It is defined by:

$$\begin{aligned} v &::= c \mid x \mid f(v_1, \ldots, v_k) \\ pre &::= R(v_1, \ldots, v_k) \mid \neg R(v_1, \ldots, v_k) \mid pre_1 \wedge pre_2 \mid pre_1 \vee pre_2 \\ &\quad \mid \forall x : pre \mid \exists x : pre \\ cl &::= R(v_1, \ldots, v_k) \mid \mathsf{true} \mid cl_1 \wedge cl_2 \mid pre \Rightarrow cl \mid \forall x : cl \end{aligned}$$

The clauses are interpreted over a non-empty universe $\mathcal{U}$; indeed, a constant c is an element of $\mathcal{U}$, a function f has arity $\mathcal{U}^* \to \mathcal{U}$, a variable x ranges over $\mathcal{U}$, and a relation R is a subset of $\mathcal{U}^*$.

The interpretation of the logic is given in terms of satisfaction relations

$$(\varrho, \sigma) \ \underline{\mathsf{sat}} \ pre \qquad \text{and} \qquad (\varrho, \sigma) \ \underline{\mathsf{sat}} \ cl$$

where ϱ is an interpretation of relations and σ is an interpretation of variables which we extend to operate on values by setting $\sigma(v) = v$. The interpretation is standard. We shall say that a clause is *closed* if it contains no free variables. For closed clauses the interpretation σ of the variables is of no importance. Fixing a specific interpretation σ_0 we thus have that (ϱ, σ) $\underline{\mathsf{sat}}$ cl agrees with (ϱ, σ_0) $\underline{\mathsf{sat}}$ cl whenever cl is closed.

The presence of negation in preconditions require some care in order to ensure the existence of least models. An *occurrence* of a relation R in a clause is a sub-formula of the form $R(v_1, \cdots, v_k)$. It is a *negative use* if it has the form $\neg\, R(v_1, \cdots, v_k)$; this necessarily occurs in a precondition, i.e. to the left of an implication. It is a *positive use* if it is not a negative use but occurs in a precondition, i.e. to the left of an implication. All other occurrences are *definitions* and often occur to the right of an implication.

Example 3. Using the equality predicate EQ the clause

$$\forall s : \forall a : \forall s' : T(s, a, s') \wedge \neg EQ(s, s') \Rightarrow R(s, a, s')$$

defines the relation R to be the subset of the labelled transition system represented by T that contains no self-loops. The occurrence of T is a positive use, the occurrence of EQ a negative use and the occurrence of R is a definition. □

A clause cl is *stratified* if there is a number r, an assignment of numbers called ranks $\mathsf{rank}_R \in \{0, \cdots, r\}$ to each relation R, and a way to write cl in the form $\bigwedge_{0 \le i \le r} cl_i$ such that the following holds for all clauses:

Table 3. Flow Logic in ALFP for the λ-calculus: $\boldsymbol{C}, \boldsymbol{R} \vdash e^{\ell}$

$$
\begin{array}{rcl}
\boldsymbol{C}, \boldsymbol{R} \vdash x^{\ell} & \underline{\text{iff}} & \forall v : R_x(v) \Rightarrow C_{\ell}(v) \\
\boldsymbol{C}, \boldsymbol{R} \vdash (\lambda x.e^{\ell'})^{\ell} & \underline{\text{iff}} & (\boldsymbol{C}, \boldsymbol{R} \vdash e^{\ell'}) \wedge C_{\ell}(\lambda x.e^{\ell'}) \\
\boldsymbol{C}, \boldsymbol{R} \vdash (e_1^{\ell_1}\ e_2^{\ell_2})^{\ell} & \underline{\text{iff}} & (\boldsymbol{C}, \boldsymbol{R} \vdash e_1^{\ell_1}) \wedge (\boldsymbol{C}, \boldsymbol{R} \vdash e_2^{\ell_2}) \wedge \\
 & & [\forall (\lambda x.e_0^{\ell_0}) : [\forall v : C_{\ell_1}(\lambda x.e_0^{\ell_0}) \wedge C_{\ell_2}(v) \Rightarrow R_x(v)] \wedge \\
 & & \quad [\forall v' : C_{\ell_0}(v') \Rightarrow C_{\ell}(v')]]
\end{array}
$$

- if cl_i contains a definition of R then[3] $\mathsf{rank}_R \geq i$;
- if cl_i contains a positive use of R then $\mathsf{rank}_R \leq i$; and
- if cl_i contains a negative use of R then $\mathsf{rank}_R < i$.

Formulae without any occurrences of negation are clearly stratified as one may simply choose $r = 0$. The clause of Example 3 is stratified: simply let EQ and T have rank 0 and let R have rank 1.

Subject to the choice of ranks made above we can define a lexicographic ordering, $\sqsubseteq$, on the the interpretations of relations, ϱ, as follows: $\varrho_1 \sqsubseteq \varrho_2$ if there exists a rank $i \in \{0, \cdots, r\}$ such that (1) $\varrho_1(R) = \varrho_2(R)$ whenever $\mathsf{rank}_R < i$, (2) $\varrho_1(R) \subseteq \varrho_2(R)$ whenever $\mathsf{rank}_R = i$, and (3) either $i = r$ or $\varrho_1(R) \subset \varrho_2(R)$ for some R with $\mathsf{rank}_R = i$. This turns the set of interpretations of relations into a complete lattice. From [15] we then have:

Theorem 1. *The set* $\{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl\}$ *is a Moore Family, i.e. is closed under greatest lower bounds, whenever* cl *is closed and stratified; the greatest lower bound* $\sqcap\{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl\}$ *is the* least *model of* cl.

More generally, given ϱ_0 *the set* $\{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl \wedge \varrho_0 \sqsubseteq \varrho\}$ *is a Moore Family and* $\sqcap\{\varrho \mid (\varrho, \sigma_0)\ \underline{\mathsf{sat}}\ cl \wedge \varrho_0 \sqsubseteq \varrho\}$ *is the least model.*

Intuitively, ALFP has been defined to be the *largest subset* of predicate logic that allows us to prove the existence of least models; in particular, clauses contain no disjunction or existential quantification.

Example 4. Following the overall methodology of Flow Logic we shall now reformulate the analysis of Example 2 in ALFP. The idea is to introduce a predicate R_x for each of the variables x and corresponding to $\mathcal{R}(x)$ and similarly a predicate C_{ℓ} for each of the labels ℓ of expressions and corresponding to $\mathcal{C}(\ell)$. The analysis is then rephrased to use judgements of the form

$$\boldsymbol{C}, \boldsymbol{R} \vdash e^{\ell}$$

and the definition of Table 2 is transformed to be within ALFP as shown in Table 3. In this case we do not need the full power of ALFP; indeed we are within the Datalog fragment [5,1]. □

[3] This generalises the condition $\mathsf{rank}_i = i$ considered in [15].

The Succinct Solver. The least model guaranteed by Theorem 1 can be constructed efficiently as summarised in the following result from [15].

Theorem 2. *Under the assumptions of Theorem 1 the least model given by* $\sqcap\{\varrho \mid (\varrho, \sigma_0)\ \mathsf{sat}\ cl \wedge \varrho_0 \sqsubseteq \varrho\}$ *is computable in time* $\mathcal{O}(|\varrho_0| + |\mathcal{U}|^K|cl|)$ *whenever* $|\varrho_0|$ *is the size of* ϱ_0 *and* $|\mathcal{U}|$ *is the size of the (necessarily finite) universe* $\mathcal{U}$ *and finally* K *is the maximal nesting depth of quantifiers within* cl*.*

The Succinct Solver [15,16] computes the least model guaranteed by Theorem 1 and has a worst case time complexity as given by Theorem 2. For many clauses the worst case time complexity does not manifest itself and the Succinct Solver operates in such a way that it may then exhibit a running time substantially lower than the worst case time complexity. Indeed, [15] gives a formula estimating the less than worst case time complexity on a given clause.

In essence the Succinct Solver deals with stratification by computing the relations in increasing order on their rank and therefore the negations present no obstacles. It combines the top-down solving approach of Le Charlier and van Hentenryck [6] with the propagation of differences [7], an optimisation technique for distributive frameworks which is also known in the area of deductive databases [3] or as reduction of strength transformations for program optimisation [19]. Programmed using functional programming its efficient operation is due to the disciplined use of continuations and memoisation as well as arbitrarily branching prefix trees as a universal data-structure for storing relations and for organising sets of waiting consumers. Whenever a new tuple is inserted into a relation this makes it easy to access the continuation for traversing the parts of the clause that might be influenced by the tuple.

Example 5. The analysis specified in Table 3 can be solved in time $\mathcal{O}(N^3)$ where N is the size of the λ-expression analysed; to see this note that the maximal nesting depth of quantifiers is 2 and that the clause and universe have (within a constant factor) the same size as the λ-expression. □

4 Flow Logic for Modal Logic

Returning to ACTL we shall now use the Flow Logic approach to define for each formula ϕ a relation R_ϕ characterising those states where the formula ϕ holds. Much as in the above analysis of the λ-calculus we shall use judgements of the form $\boldsymbol{R} \vdash \phi$ to define the relations R_ϕ of interest. Since the same sub-formula may occur several times in ϕ we shall identify each sub-formula with a unique label ℓ; in examples and explanations where this problem does not arise we shall dispense with the labels.

We assume that

- for each basic predicate bp we have a corresponding relation P_{bp} on states,
- for each subset Ω of $\mathcal{A}$ we have a relation Ω on actions, and
- the transition relation $\rightarrow$ is presented by a ternary relation T.

Table 4. ACTL in ALFP: $\boldsymbol{R} \vdash \phi^{\ell}$ for defining $R_{\phi^{\ell}}$ (part 1)

$$
\begin{array}{rcl}
\boldsymbol{R} \vdash \mathsf{true}^{\ell} & \underline{\text{iff}} & [\forall s : R_{\mathsf{true}^{\ell}}(s)] \\
\boldsymbol{R} \vdash \mathsf{false}^{\ell} & \underline{\text{iff}} & \mathsf{true} \\
\boldsymbol{R} \vdash bp^{\ell} & \underline{\text{iff}} & [\forall s : P_{bp}(s) \Rightarrow R_{bp^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\phi_1^{\ell_1} \wedge \phi_2^{\ell_2})^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge [\forall s : R_{\phi_1^{\ell_1}}(s) \wedge R_{\phi_2^{\ell_2}}(s) \Rightarrow R_{(\phi_1^{\ell_1} \wedge \phi_2^{\ell_2})^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\phi_1^{\ell_1} \vee \phi_2^{\ell_2})^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge [\forall s : R_{\phi_1^{\ell_1}}(s) \vee R_{\phi_2^{\ell_2}}(s) \Rightarrow R_{(\phi_1^{\ell_1} \vee \phi_2^{\ell_2})^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\neg \phi^{\ell'})^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi^{\ell'} \wedge [\forall s : (\neg R_{\phi^{\ell'}}(s)) \Rightarrow R_{(\neg \phi^{\ell'})^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\phi_1^{\ell_1} \Rightarrow \phi_2^{\ell_2})^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge [\forall s : \neg R_{\phi_1^{\ell_1}}(s) \vee R_{\phi_2^{\ell_2}}(s) \Rightarrow R_{(\phi_1^{\ell_1} \Rightarrow \phi_2^{\ell_2})^{\ell}}(s)]
\end{array}
$$

Table 5. ACTL in ALFP: $\boldsymbol{R} \vdash \phi^{\ell}$ for defining $R_{\phi^{\ell}}$ (part 2)

$$
\begin{array}{rcl}
\boldsymbol{R} \vdash (\mathbf{EX}_{\Omega}\, \phi^{\ell'})^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi^{\ell'} \wedge \\
 & & [\forall s : [\exists a : \exists s' : T(s,a,s') \wedge \Omega(a) \wedge R_{\phi^{\ell'}}(s')] \\
 & & \quad \Rightarrow R_{(\mathbf{EX}_{\Omega}\, \phi^{\ell'})^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\mathbf{AX}_{\Omega}\, \phi^{\ell'})^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi^{\ell'} \wedge \\
 & & [\forall s : [\forall a : \forall s' : \neg T(s,a,s') \vee (\Omega(a) \wedge R_{\phi^{\ell'}}(s'))] \wedge \\
 & & \quad [\exists a : \exists s' : T(s,a,s')] \Rightarrow R_{(\mathbf{AX}_{\Omega}\, \phi^{\ell'})^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge \\
 & & [\forall s : [\exists a : \exists s' : T(s,a,s') \wedge \Omega_2(a) \wedge R_{\phi_2^{\ell_2}}(s')] \\
 & & \quad \Rightarrow R_{(\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s)] \wedge \\
 & & [\forall s : [\exists a : \exists s' : T(s,a,s') \wedge \Omega_1(a) \wedge R_{\phi_1^{\ell_1}}(s') \wedge \\
 & & \quad R_{(\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s')] \Rightarrow R_{(\mathbf{E}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s)] \\
\boldsymbol{R} \vdash (\mathbf{A}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell} & \underline{\text{iff}} & \boldsymbol{R} \vdash \phi_1^{\ell_1} \wedge \boldsymbol{R} \vdash \phi_2^{\ell_2} \wedge \\
 & & [\forall s : [\ [\exists a : \exists s' : T(s,a,s')] \wedge \\
 & & \quad [\forall a : \forall s' : \neg T(s,a,s') \vee [\Omega_2(a) \wedge R_{\phi_2^{\ell_2}}(s')] \vee \\
 & & \quad\quad [\Omega_1(a) \wedge R_{\phi_1^{\ell_1}}(s') \wedge R_{(\mathbf{A}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s')]\ ] \\
 & & \quad\quad\quad \Rightarrow R_{(\mathbf{A}[\phi_1^{\ell_1}\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2^{\ell_2}])^{\ell}}(s)]
\end{array}
$$

The interpretations of these predicates are fixed and the intention is to define the judgements $\boldsymbol{R} \vdash \phi$ such that $s \models \phi$ holds whenever $R_{\phi}(s)$ holds in the least model satisfying $\boldsymbol{R} \vdash \phi$. The definition is given in Tables 4 and 5 and is explained below.

The relation R_{true} corresponding to the ACTL formula true should hold for all states and we express this by the ALFP clause $\forall s : R_{\mathsf{true}}(s)$. The relation R_{false} corresponding to false does not hold on any state so the ALFP clause does not insist on R_{false} holding on any states; hence we simply use the ALFP clause true to reflect this meaning that in the least model there are no states where R_{false} holds. For basic predicates bp we have to make use of the predefined predicate P_{bp} and impose the constraint that if P_{bp} holds on some state s then so does the relation R_{bp}.

The four clauses for the propositional operators $\wedge$, $\vee$, $\neg$ and $\Rightarrow$ follow the same pattern so let us just explain one of them, namely conjunction $\phi_1 \wedge \phi_2$. Here the conjuncts $\boldsymbol{R} \vdash \phi_1$ and $\boldsymbol{R} \vdash \phi_2$ ensure that the relations $R_{\phi'}$ corresponding to sub-expressions of ϕ_1 and ϕ_2 correctly record which states are acceptable. The third conjunct $\forall s : R_{\phi_1}(s) \wedge R_{\phi_2}(s) \Rightarrow R_{\phi_1 \wedge \phi_2}(s)$ then caters for the relation $R_{\phi_1 \wedge \phi_2}$. Note that in the case of negation and implication we have *negative uses* of relations; we shall return to the stratification issues related to this later.

Turning to the modal operators of Table 5 let us first consider $\mathbf{EX}_{\Omega}\, \phi$. The first conjunct ensures that the sub-formula ϕ is handled correctly. The second conjunct captures the semantics of the **EX** construct: for all states s there must be an action a and a state s' such that there is a transition in the corresponding transition system (i.e. $T(s, a, s')$), that a is in Ω (i.e. $\Omega(a)$) and furthermore ϕ holds in s', i.e. that $R_{\phi}(s')$ holds. If these conditions are satisfied then it also must be the case that $R_{\mathbf{EX}_{\Omega}\, \phi}$ holds on s.

The clause for $\mathbf{AX}_{\Omega}\, \phi$ is slightly more complicated since we must cater for the possibility of stuck states in the transition system. The clause in Table 5 reflects that two conditions must be satisfied in order for the formula to hold on a given state s. First it must be the case that any transition going out of s must make use of an action from Ω and must lead to a state satisfying ϕ. The other conjunct ensures that s is not a stuck state. Only if both conditions are fulfilled the ALFP clause imposes the requirement that $R_{\mathbf{AX}_{\Omega}\, \phi}$ holds on s. Note that also in this case we have a *negative use* of a relation. Furthermore we are stepping outside the Datalog [5,1] fragment of ALFP in that we use universal quantification in a precondition of an implication.

The clause for $\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]$ captures two possibilities. There might be a transition from s using an action from Ω_2 and resulting in a state satisfying R_{ϕ_2} and if so then $R_{\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ should also hold on s. Alternatively there might be a transition out of s using an action from Ω_1 and in this case it has to lead to a state satisfying not only R_{ϕ_1} but also $R_{\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ – and only if this is the case we impose the condition that $R_{\mathbf{E}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ also holds on s. It is here worth pointing out that the ALFP clause defining the relation of interest is recursive in the sense that it contains both *uses* and *definitions* of the relation.

Finally let us consider the clause for $\mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]$. Here the first component of the premise of the implication insists that the state s of interest is not a stuck state. The second component then requires that a transition out of s either makes use of an Ω_2 action and then the next state satisfies R_{ϕ_2} or it makes use of an Ω_1 action and then the next state satisfies R_{ϕ_1} as well as $R_{\mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$. If these premises are all satisfied then the clause expresses that $R_{\mathbf{A}[\phi_1\ {}_{\Omega_1}\mathbf{U}_{\Omega_2}\ \phi_2]}$ should hold on s. This is by far the most complex of the clauses as it makes use of universal quantification in preconditions, has *negative uses* of relations and defines the relation of interest in a recursive manner as discussed above.

In order to complete the presentation of the Flow Logic of Tables 4 and 5 we should like to establish that the ALFP clause constructed for an ACTL formula is indeed *closed* and *stratified*. We shall do so below and this will allow us to

make use of Theorem 1 to show that least models exists – and that the Succinct Solver can be used to compute them.

Example 6. Consider the ACTL formula $\mathbf{AF}_{\mathcal{A}}\ \mathsf{goal}$ expressing that the transition system of interest eventually will reach a goal state. It is equivalent to $\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]$ (omitting annotations) and using the definition in Tables 4 and 5 we obtain the following ALFP clause defining the predicate $R_{\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]}$:

$$
\begin{array}{l}
[\forall s : R_{\mathsf{true}}(s)] \wedge \\
[\forall s : P_{\mathsf{goal}}(s) \Rightarrow R_{\mathsf{goal}}(s)] \wedge \\
[\forall s : [\,[\exists a : \exists s' : T(s,a,s')] \wedge \\
\qquad [\forall a : \forall s' : \neg T(s,a,s') \vee R_{\mathsf{goal}}(s') \vee R_{\mathbf{A}[\mathsf{true}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]}(s') \Rightarrow R_{\mathbf{A}[\mathsf{true}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]}(s)]]]
\end{array}
$$

Here we exploit that $\mathcal{A}$ holds on all actions and that R_{true} holds on all states. □

Remark. It is instructive to note that we *cannot* simply define $\boldsymbol{R} \vdash \mathbf{AG}_{\Omega}\,\phi$ (omitting annotations) by the conjunction of $\boldsymbol{R} \vdash \phi$ and

$$[\forall s : [\forall a : \forall s' : \neg T(s,a,s') \vee (R_{\mathbf{AG}_{\Omega}\,\phi}(s') \wedge (\neg\Omega(a) \vee R_{\phi}(s')))] \Rightarrow R_{\mathbf{AG}_{\Omega}\,\phi}(s)]$$

since the above computes the least fixed point rather than the intended greatest fixed point – which is important for transition systems with loops. Rather we need to compute the staged relations $\boldsymbol{R} \vdash \neg\mathbf{E}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\Omega}\ \neg\phi]$ and due to the use of negation this requires stratification.

Example 7. To illustrate this observation, consider the transition system of Example 1 and the formula $\mathbf{AG}_{\{C\}}\ \mathsf{goal}$ (omitting annotations). The above definition gives rise to the recursive equation

$$R = \{s \mid \forall s' : \forall a : s \rightarrow^{a} s' \Rightarrow (R(s') \wedge (\Omega(a) \Rightarrow s' \models \mathsf{goal}))\}$$

The least solution is $R = \emptyset$ whereas the correct solution is $R_{\mathbf{AG}_{\{C\}}\ \mathsf{goal}} = \{1,2,3\}$. □

Post-order labelling. It is easy to see that the clauses $\boldsymbol{R} \vdash \phi^{\ell}$ defined by Tables 4 and 5 are indeed *closed* for any ACTL formula ϕ^{ℓ}. To show that the clauses are *stratified* we shall fix the labelling scheme that so far has been unspecified.

To this end we assume that all sub-formula of an ACTL formula are annotated with their number in a *post-order traversal* of the formula. To be specific we shall say that a formula ϕ is (i,j)-*annotated* for positive integers i and j if the lowest annotation occurring within ϕ is i and the highest is j – since we consider a post-order traversal the annotation of the formula ϕ itself will necessarily be j. As an example the (5,7)-annotated version of the formula $\mathbf{A}[\mathsf{true}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}]$ is $\mathbf{A}[\mathsf{true}^{5}\ {}_{\mathcal{A}}\mathbf{U}_{\mathcal{A}}\ \mathsf{goal}^{6}]^{7}$.

Proposition 1. *For every ACTL formula ϕ there exists an $(1,\ell)$-annotated formula ϕ^{ℓ} for some ℓ. Furthermore, the ALFP clause $\boldsymbol{R} \vdash \phi^{\ell}$ generated by Tables 4 and 5 from an $(1,\ell)$-annotated ACTL formula ϕ^{ℓ} is closed and stratified.*

To see the latter assign the relations P_{bp}, Ω and T the rank 0. We may then prove by induction on the annotated sub-formulae ϕ'^j of ϕ^ℓ that ϕ'^j is closed and (i,j)-annotated for some i. Furthermore, the ALFP clause $\boldsymbol{R} \vdash \phi'^j$ will only contain definitions of relations of rank in $\{i,\cdots,j\}$, it will only contain uses of relations of rank $\{0\} \cup \{i,\cdots,j\}$ and all negative uses will involve relations of rank $\{0\} \cup \{i,\cdots,j-1\}$ and thus it is a stratified formula. We shall say that an ALFP clause is (i,j)-*stratified* when this is the case.

Correctness and precision. We are now able to establish our main theorem expressing the *correctness* of the static analysis of the modal logic with respect to the interpretation of the modal logic; this is along the same lines as the correctness result exhibited for the analysis of the λ-calculus in Example 2. Additionally, the theorem expresses that the analysis is *precise*; this is not usually the case for static analyses but does indeed hold for the so-called *collecting semantics* analyses that often are the starting point for developing coarser analyses by Abstract Interpretation.

For the remainder of this section consider some transition system $(S, \mathcal{A}, \rightarrow)$ and a ranking as described above; we shall restrict ourselves to a finitely branching transition system. Furthermore assume that $\mathcal{U} = S$, that $\varrho_0(\Omega) = \Omega$ for all $\Omega \subseteq \mathcal{A}$, that $\varrho_0(T) = \{(s,a,s') \mid s \rightarrow^a s'\}$, and that $\varrho_0(P_{bp}) = \{s \mid bp \text{ holds on } s\}$. Then we have the following main result that is proved by structural induction on the formulae:

Theorem 3. *Consider an (i,j)-annotated formula ϕ^j in ACTL and the least model ϱ of $\boldsymbol{R} \vdash \phi$ such that $\varrho \sqsupseteq \varrho_0$. We then have:*

- *Correctness of ϱ: if $s \models \phi^j$ then $\varrho(R_{\phi^j})(s)$.*
- *Precision of ϱ: if $\varrho(R_{\phi^j})(s)$ then $s \models \phi^j$.*

It follows from Proposition 1 and Theorem 3 that an implementation of the static analysis of ACTL by means of the Succinct Solver constitutes a *model checker* for ACTL.

Complexity. We may estimate the worst case time complexity of model checking as performed using our static analysis of modal logic. For this assume that S has size $|S|$ and the transition relation $\rightarrow$ has size $|T|$. Next consider an ACTL formula ϕ of size $|\phi|$; it is immediate that the ALFP clause $\boldsymbol{R} \vdash \phi$ has size $\mathcal{O}(|\phi|)$ and nesting depth 3. According to Theorem 2 the worst case time complexity is $\mathcal{O}(|T| + \sum_{bp} |P_{bp}| + 2^{|\mathcal{A}|} + |S|^3|\phi|)$ where bp ranges over all base predicates.

To compare with more traditional approaches to model checking we shall assume that both the number of base predicates and the number of action labels (as opposed to actions) is bounded by some constant. In this case we obtain the worst case time complexity $\mathcal{O}(|T| + |S|^2|\phi|)$ because in this case the calculations of nesting depth can safely ignore quantifications over actions. Indeed using a more refined reasoning than that of Theorem 2 we obtain $\mathcal{O}((|T| + |S|)\,|\phi|)$ because it is clear that the "double quantifications" over states in Table 5 really correspond to traversing all possible transitions rather than all pairs of states.

Thus our alternative model checking algorithm has the same complexity as classical model checking algorithms [2]. And indeed, in our experience the Succinct Solver operates in such a manner that it will attain this worst case time complexity.

5 Conclusion

The interplay between static analysis and model checking has intrigued researchers for many years – leading to a growing feeling that the methods are connected and can be used to strengthen each other.

At the conceptual level a number of papers have argued that various forms of static analysis are instances of model checking [22,20,21]. Our contribution completes the circle by showing the close relationship between static analysis and model checking – to the extent that one may conjecture that they are reducible to each others.

At the practical level this gives increased faith in exploring the methods and techniques developed within each of the approaches to strengthen the other. This is at the core of Research Theme 1 in MT-LAB (see below) where it is hypothesised that "static analysis and model checking fundamentally solve the same problem but using a different repertoire of techniques that must be combined in order to produce more powerful analysis techniques." In future work we hope to identify concrete methods and techniques that can be transferred between the two approaches to their mutual benefit.

Acknowledgements. This work has been supported by MT-LAB, a VKR Centre of Excellence (`http://www.mt-lab.dk`). The authors would like to thank Kim Guldstrand Larsen for provoking us to write this paper.

References

1. Apt, K., Blair, H., Walker, A.: A theory of declarative programming. In: Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan-Kaufman, San Francisco (1988)
2. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Balbin, I., Ramamohanarao, K.: A generalization of the differential approach to recursive query evaluation. Journal of Logic Programming 4(3), 259–262 (1987)
4. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Riis Nielson, H.: Static validation of security protocols. Journal of Computer Security 13, 347–390 (2005)
5. Chandra, A., Harel, D.: Computable queries for relational data bases. Journal of Computer and System Sciences 21(2), 156–178 (1980)
6. Le Charlier, B., Van Hentenryck, P.: A universal top-down fixpoint algorithm. Technical report, CS-92-25, Brown University (1992)
7. Fecht, C., Seidl, H.: Propagating differences: An efficient new fixpoint algorithm for distributive constraint systems. Nordic Journal of Computing 5(4), 304–329 (1998)

8. Hankin, C., Nielson, F., Riis Nielson, H.: Advice from Belnap policies. In: Computer Security Foundations Symposium. IEEE Computer Society, Los Alamitos (2009)
9. Hecht, M.S.: Flow Analysis of Computer Programs. North Holland, Amsterdam (1977)
10. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
11. Lamprecht, A.-L., Margaria, T., Steffen, B.: Data-flow analysis as model checking within the jABC. In: Mycroft, A., Zeller, A. (eds.) CC 2006. LNCS, vol. 3923, pp. 101–104. Springer, Heidelberg (2006)
12. De Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
13. Nielson, F., Riis Nielson, H., Hankin, C.: Principles of Program Analysis, 2nd edn. Springer, Berlin (2005)
14. Nielson, F., Riis Nielson, H., Hansen, R.R.: Validating firewalls using flow logics. Theoretical Computer Science 283(2), 381–418 (2002)
15. Nielson, F., Riis Nielson, H., Seidl, H.: A succinct solver for ALFP. Nordic Journal of Computing 9, 335–372 (2002)
16. Nielson, F., Riis Nielson, H., Sun, H., Buchholtz, M., Hansen, R.R., Pilegaard, H., Seidl, H.: The Succinct Solver Suite. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 251–265. Springer, Heidelberg (2004)
17. Riis Nielson, H., Nielson, F.: Flow Logic: a multi-paradigmatic approach to static analysis. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 223–244. Springer, Heidelberg (2002)
18. Riis Nielson, H., Nielson, F., Pilegaard, H.: Spatial analysis of BioAmbients. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 69–83. Springer, Heidelberg (2004)
19. Paige, R.: Symbolic finite differencing - part i. In: Jones, N.D. (ed.) ESOP 1990. LNCS, vol. 432, pp. 36–56. Springer, Heidelberg (1990)
20. Schmidt, D.A.: Data flow analysis is model checking of abstract interpretations. In: POPL 1998, pp. 38–48 (1998)
21. Schmidt, D.A., Steffen, B.: Program analysis *as* model checking of abstract interpretations. In: Levi, G. (ed.) SAS 1998. LNCS, vol. 1503, pp. 351–380. Springer, Heidelberg (1998)
22. Steffen, B.: Data flow analysis as model checking. In: Ito, T., Meyer, A.R. (eds.) TACS 1991. LNCS, vol. 526, pp. 346–365. Springer, Heidelberg (1991)