Model Checking and Transitive-Closure Logic*

Neil Immerman**[1] and Moshe Y. Vardi***[2]

[1] Computer Science Dept., University of Massachusetts, Amherst, MA 01003, http://www.cs.umass.edu/~immerman, immerman@cs.umass.edu
[2] Computer Science Dept., Rice University, Houston, TX 77005-1892, http://www.cs.rice.edu/~vardi, vardi@cs.rice.edu

Abstract. We give a linear-time algorithm to translate any formula from computation tree logic (CTL or CTL*) into an equivalent expression in a variable-confined fragment of transitive-closure logic FO(TC). Traditionally, CTL and CTL* have been used to express queries for model checking and then translated into μ-calculus for symbolic evaluation. Evaluation of μ-calculus formulas is, however, complete for time polynomial in the (typically huge) number of states in the Kripke structure. Thus, this is often not feasible, not parallelizable, and efficient incremental strategies are unlikely to exist. By contrast, evaluation of any formula in FO(TC) requires only NSPACE[$\log n$]. This means that the space requirements are manageable, the entire computation is parallelizable, and efficient dynamic evaluation is possible.

1 Introduction

Model checking, proposed first as a paradigm for computer-aided verification of finite-state programs in [CE81] and developed further in [BCM92, CES86, LP85, QS81, VW86] has been gaining widespread acceptance lately (see [BBG94]). The approach is especially appropriate for the design and verification of circuits and distributed protocols. The detailed, low-level design can be automatically translated into a logical structure called a Kripke structure $\mathcal{K}$. We can then write a series of short correctness conditions $\varphi_1, \varphi_2, \ldots$ concerning the behavior of the Kripke structure. The conditions are written in a formal language such as computation tree logic (CTL) or the more expressive CTL*. Given $\mathcal{K}$ and φ_i, the model-checking program will automatically test whether or not $\mathcal{K}$ satisfies φ_i. If it does, then confidence in the design is improved. If $\mathcal{K}$ does not satisfy some φ_i, then the checking program will usually present a counter example which thus exposes a bug in the design.

The Kripke structures used in model checking usually have a state for each possible configuration of the circuit or protocol being designed. For this reason they are often of size exponential in the size of the design. In this case,

* Part of the research reported here was conducted while the authors were visiting DIMACS during the Special Year on Logic and Algorithm.
** Research partly supported by NSF grant CCR-9505446.
*** Research partly supported by NSF grant CCR-9628400

one usually represents the Kripke structure symbolically rather than explicitly, often using ordered binary decision diagrams (OBDDs). The model checking performed using these symbolic representations is called symbolic model checking [BCM92, McM93].

The correctness conditions φ_i described above can be thought of as queries to the Kripke structure. In fact, in this paper we emphasize the close relationship between model checking and database query evaluation (cf. [Var97]). Optimization of the queries is crucial. For this reason, the tradeoff between the expressive power of the query language and the complexity of doing model checking is important.

A powerful query language for model checking is the branching-time logic CTL*. Consider the model checking problem for CTL* in which we fix a query $\varphi \in$ CTL* and vary the Kripke structure $\mathcal{K}$. The complexity of this problem, called *program complexity* in [VW86] and *data complexity* in [Var82], is known to be NSPACE[$\log n$] [BVW94] for CTL*. Here n is the size of the Kripke structure – as we have mentioned, n is often exponential in the size of the design being verified.

The standard way to perform symbolic model checking using CTL* is to translate the query to the modal μ-calculus [Koz83, EL86]. A problem with this is that the data complexity of the modal μ-calculus is polynomial-time complete [BVW94] (cf. [I86, Var82]). This means that evaluation of modal μ-calculus queries most likely requires polynomial space, is not parallelizable, and efficient incremental evaluation strategies are unlikely to exist.

We give here a linear-time algorithm to translate any formula from CTL* into an equivalent expression in a variable-confined fragment of transitive-closure logic FO(TC). In fact, the resulting formulas have only two first-order variables. The resulting logic, denoted FO^2(TC), is known to have a data complexity of NSPACE[$\log n$] [I87, Var82]. This means that the space requirements are manageable, the entire computation is parallelizable, and efficient incremental evaluation is possible (see, for example, [PI94, ZSS94]). Thus, it is very promising to do model checking and symbolic model checking using the language FO^2(TC) rather than the more complex modal μ-calculus.

2 Background on Temporal Logic and the Modal μ-calculus

Let $\Phi = \{p_1, \ldots, p_r\}$ be a finite set of propositional symbols. A *propositional Kripke structure,* $\mathcal{K} = (S, R, \pi)$, is a tuple consisting of a finite set of states S, a binary transition relation $R \subseteq S^2$, and a labeling function $\pi : \Phi \to 2^S$, where intuitively, $\pi(p_i)$ is the set of states at which p_i is true. S is often called the set of possible worlds, but we call it the set of states because in model checking applications it usually represents the set of global states of the circuit or protocol being designed. Typically, we are interested in infinite computation paths, so in this paper we restrict our attention to Kripke structures in which every state has at least one successor, which may be itself. We can meet this condition by adding the loop $R(s,s)$ to each state that has no other successors. A Kripke structure may be thought of as a directed graph whose vertices are the states, labeled by the set of propositional symbols they satisfy.

The propositional Kripke structure $\mathcal{K}$ may also be thought of as a finite relational structure, i.e., relational database, $\mathcal{K}^\star = (S, R, p_1^{\mathcal{K}^\star}, \ldots, p_r^{\mathcal{K}^\star})$. The universe of $\mathcal{K}^\star$ is the set of states S. The binary relation $R \subseteq S^2$ is the transition relation, and a unary relation $p_i^{\mathcal{K}^\star} = \pi(p_i)$ is the set of states at which the proposition p_i holds. For any first-order formula φ, we will use the notation $\mathcal{K}^\star \models \varphi$ to mean that φ is true in $\mathcal{K}^\star$.

We use in this paper the computation tree logics CTL and CTL*. For definitions of syntax and semantics of these logics see [Eme90].

The modal μ-calculus is a propositional modal logic that includes the least-fixed point operator (μ) [Koz83, Eme97]. The modal μ-calculus is strictly more expressive than CTL*, and has polynomial-time data complexity (see next section). As an example, we can write the CTL formula $\mathbf{EF}p$ as a least fixed point,

$$\mathbf{EF}p \quad \equiv \quad \mu Y(p \vee \langle R \rangle Y) \tag{1}$$

Equation 1 can be generalized to show that all of CTL* can be interpreted in the modal μ-Calculus. See [Eme97, Var97] for details.

Fact 2

- *There is a linear time algorithm that translates any formula in* CTL *into an equivalent formula in the modal μ-calculus.*
- *There is an exponential time algorithm that translates any formula in* CTL* *into an equivalent formula in the modal μ-calculus.*

Symbolic model checking is typically carried out by first translating the CTL correctness condition into the μ-calculus [McM93]. A drawback of this approach is that model checking of the μ-calculus uses space polynomial in the size of the usually huge Kripke structure. In the next section we describe transitive-closure logic. We will see that although transitive-closure logic has lower complexity than the μ-calculus, it still suffices to interpret CTL*.

3 Background on Descriptive Complexity

In descriptive complexity, we study finite logical structures — relational databases — such as the Kripke structures,

$$\mathcal{K}^\star \quad = \quad (S, R, p_1^{\mathcal{K}^\star}, \ldots, p_r^{\mathcal{K}^\star}) \ .$$

The complexity of computing queries on such structures is intimately tied to the power of variants of first-order logic needed to describe these queries. This has been studied in great detail. See for example [EF95, I89, LR96, Var82].

Let FO be the set of first-order expressible properties. For example, consider the first-order formula,

$$\varphi \quad \equiv \quad (\forall x)(p(x) \rightarrow (\exists y)(R(x,y) \wedge p(y)))\ .$$

A Kripke structure $\mathcal{K}^\star$ satisfies φ — in symbols, $\mathcal{K}^\star \models \varphi$ — iff every state satisfying p has a successor state that also satisfies p.

The class FO captures the complexity class AC^0 consisting of those properties checkable by bounded depth polynomial-size circuits. This is equal to the set of properties computable in constant time on a concurrent parallel random access machine that has at most polynomially many processors [I89a].

To obtain a richer class of queries, let FO(LFP) be first-order logic extended by a least-fixed-point operator. This is the closure of first-order logic under the power to define new relations by induction. We can view the modal μ-calculus as a restriction of FO(LFP) in which all fixed points are taken over monadic relations, and such that only two domain variables are used. Let FO^k be the restriction of FO such that the only domain variables are $x_1, \ldots, x_k$. Let LFP^r be the restriction of LFP to act only on inductive definitions of arity at most r. Then there is a linear-time mapping of each formula from the modal μ-calculus to an equivalent formula in $\mathrm{FO}^2(\mathrm{LFP}^1)$ [Var97].

As an example, consider the μ-calculus formula, $\psi \equiv \mu Y(p \vee \langle R \rangle Y)$. Recall from Equation 1 that ψ is equivalent to the CTL formula $\mathbf{EF}p$. This can be interpreted in FO(LFP) as the formula,

$$\psi' \quad \equiv \quad \mathrm{LFP}_{Y,y}(p(y) \vee \exists y'(R(y,y') \wedge Y(y')))\,(y)\ . \tag{3}$$

The equivalence between ψ and ψ' is that for any propositional Kripke structure $\mathcal{K}$ and state s,

$$(\mathcal{K}, s) \models \psi \quad \Leftrightarrow \quad (\mathcal{K}^\star, s/y) \models \psi'\ .$$

It is well known that FO(LFP) captures polynomial time. The following facts assume that structures in question are finite and include a total ordering on their universes.

Fact 4 ([I86, Var82]) *The queries computable in polynomial time are exactly those expressible in* FO(LFP).

While the modal μ-calculus is a proper subset of FO(LFP), it still contains problems complete for polynomial-time [BVW94]. Since the model checking problem for $\mathrm{CTL}^\star$ is contained in NSPACE[$\log n$], it would be much better to interpret $\mathrm{CTL}^\star$ in a logic with this lower complexity.

Let the formula $\varphi(x_1, \ldots x_k, y_1, \ldots y_k)$ represent a binary relation on k-tuples. We express the reflexive, transitive closure of this relation using the transitive-closure operator (TC), as follows: $\mathrm{TC}_{\overline{x},\overline{y}}\varphi$. Let FO(TC) be the closure of first-order logic under the transitive-closure operator. For example, the following formula is equivalent to ψ' (Equation 3) and thus interprets the CTL formula,

EFp. It does so directly, by saying that there is an R-path to a state satisfying p.

$$\psi'' \quad \equiv \quad (\exists y')[(\mathrm{TC}_{y,y'} R(y,y'))(y,y') \ \wedge \ p(y')]$$

We will see in the next section that every formula in CTL* can be so interpreted.

Transitive closure logic exactly captures nondeterministic logspace:

Fact 5 ([I87, I88]) *The queries computable in* NSPACE[$\log n$] *are exactly those expressible in* FO(TC).

The number of variables used is an important descriptive resource. Each domain variable x_i ranges over the universe of its input structure. In the definition of FO^k, we allow an unbounded number of *boolean variables*, $b_1, \ldots, b_c$ in addition to the k domain variables. Boolean variables are essentially first-order variables that are restricted to range only over the first two elements of the universe, which we fix as 0 and 1. Including also boolean variables makes the definition of FO^k more robust [I91]. As a simple example, we can interpret the conjunction $\mathbf{EF}p \wedge \mathbf{EF}q$ using a universally quantified boolean variable,

$$(\forall b)(\exists y')[(\mathrm{TC}_{y,y'} R(y,y'))(y,y') \ \wedge \ (b \wedge p(y') \ \vee \ \neg b \wedge q(y'))] \ .$$

We note, however, that the inclusion of boolean variables has a nontrivial complexity-theoretic consequence. While "pure" (i.e., boolean-variable-free) queries in FO^k(TC) can be evaluated in *uniform* polynomial time, the space required to evaluate queries in FO^k(TC) is polynomial in the number of boolean variables.

We sometimes want a *strict* transitive closure operator: $\mathrm{TC}^s(\varphi)$ denotes the transitive closure of φ, as opposed to the reflexive, transitive closure of φ. The strict and reflexive transitive closure operators are definable from each other, as follows. Note that no extra variables are needed:

$$\begin{aligned}
\mathrm{TC}(\varphi(y,y'))(y,y') &\equiv y = y' \quad \vee \quad \mathrm{TC}^s(\varphi(y,y'))(y,y') \\
\mathrm{TC}^s(\varphi(y,y'))(y,y') &\equiv (y \neq y' \quad \wedge \quad \mathrm{TC}(\varphi(y,y'))(y,y')) \quad \vee \\
&\qquad (y = y' \quad \wedge \quad (\exists y')(\varphi(y,y') \ \wedge \ \mathrm{TC}(\varphi(y,y'))(y',y)))
\end{aligned}$$

4 Transitive Closure Logic Suffices

In this section we present an algorithm that translates any formula in CTL* to an equivalent formula in FO^2(TC), i.e., first-order logic with only two first-order variables, extended by the transitive-closure operator. We first do the case of CTL, which is significantly simpler.

Theorem 6. *There is a transformation f from state formulas in* CTL *to formulas in* $\mathrm{FO}^2(\mathrm{TC})$ *that preserves meaning. That is, for all state formulas $\varphi \in$ CTL and all Kripke structures $\mathcal{K}$, and states s,*

$$(\mathcal{K}, s) \models \varphi \quad \Leftrightarrow \quad (\mathcal{K}^\star, s/y) \models f(\varphi) \tag{7}$$

Proof. We define f by induction on φ,

- $f(p) \ = \ p(y)$, for predicate symbol p
- $f(\neg\varphi) \ = \ \neg f(\varphi)$
- $f(\varphi \wedge \psi) \ = \ f(\varphi) \wedge f(\psi)$
- $f(\mathbf{E}(\varphi\mathbf{U}\psi)) \ = \ (\exists y')(\mathrm{TC}(M_{f(\varphi)})(y, y') \ \wedge \ f(\psi)(y'))$
  where, $M_\alpha(y, y') \ \equiv \ R(y, y') \ \wedge \ \alpha(y)$
- $f(\mathbf{E}(\varphi\mathbf{B}\psi)) \ = \ (\exists y')(\mathrm{TC}(M_{f(\psi)})(y, y') \wedge ((f(\varphi)(y') \wedge f(\psi)(y')) \vee \mathrm{TC}^s(M_{f(\psi)})(y', y')))$

It is easy to show by induction that Equation 7 holds. The interesting cases are the last two: For "Until", note that there is a path starting at y along which $\varphi\mathbf{U}\psi$ holds iff there is some point y' at which ψ holds and there is a path from y to y' along which φ holds. For "Before", there is a path starting at y along which $\varphi\mathbf{B}\psi$ holds iff there is some point y' for which there is a path from y to y' along which ψ holds, and either φ and ψ both hold at y', or there is an infinite path, i.e., a cycle, starting at y' along which ψ remains true. □

Note that the formulas $f(\varphi)$ does not use boolean variables. We would be happier if the above f were linear-time computable. The problem is that the formula $f(\psi)$ occurs more than once in the definition of $f(\mathbf{E}(\varphi\mathbf{B}\psi))$. This could cause an exponential blowup in the size of the resulting formula. We will defer this problem to Corollary 11.

The difficulty in extending Theorem 6 to $\mathrm{CTL}^\star$ occurs in a formula such as

$$\alpha \quad = \quad \mathbf{E}((p \to q\mathbf{U}r)\mathbf{U}t)$$

As before, we can express that at some state y', t holds, and that there is a path from y to y' along which $(p \to q\mathbf{U}r)$ holds. The problem is that we must remember our obligations along this path, i.e., whether we need to preserve $q\mathbf{U}r$, and we may need to preserve this along the same path, beyond y'.

To solve this problem we introduce a new boolean variable b, whose purpose is to remember our obligation concerning the formula $q\mathbf{U}r$. The following formula $\alpha^\star$ asserts that there is a path to a future state y' and a boolean value b' such that $(p \to q\mathbf{U}r)$ holds along the path, t holds at y', and if b' holds, i.e., we are still obliged to fulfill $q\mathbf{U}r$, then there is a continuation of the path along which $q\mathbf{U}r$ holds,

$$\begin{aligned} \alpha^\star \equiv (\exists y'b')(\mathrm{TC}(\gamma)(y, \mathbf{false}, y', b') \ \wedge \ t(y') \ \wedge \\ b' \to (\exists y)(\mathrm{TC}(M_q)(y', y) \wedge r(y))) \\ \gamma(y, b, y', b') \equiv ((p \vee b) \to (r \vee (q \wedge b'))) \ \wedge \ R(y, y') \end{aligned}$$

Observe that as desired, for all Kripke structures $\mathcal{K}$ and states s,

$$(\mathcal{K}, s) \models \alpha \quad \Leftrightarrow \quad (\mathcal{K}^\star, s/y) \models \alpha^\star$$

Reiterating the main point, in addition to the $\log n$ bits needed to name y – the current state in our n-state Kripke structure – we use one additional bit b to record our obligations concerning the truth of a formula along the remainder of a path.

We now describe this construction in general so that we may extend Theorem 6 to CTL*. Let $\mathbf{E}(\varphi)$ be a CTL* formula. Define the *closure* of φ ($cl(\varphi)$) to be the set of path subformulas of φ. We introduce a boolean variable b_α for each $\alpha \in cl(\varphi)$. Intuitively, we use the boolean variables to encode the state of the automaton that runs along a path and checks that the path satisfies a path formula (see [VW94]).

We inductively define a mapping g from state formulas $\mathbf{E}(\varphi)$ in CTL* to equivalent formulas in FO^2(TC). Let $\bar{b}$ be a tuple of all the boolean variables b_α, for $\alpha \in cl(\varphi)$. Define the transition relation $R^0_\varphi(y, \bar{b}, y', \overline{b'})$ as follows. In each case, the comment on the right is the condition underwhich the given conjunct is included in the formula. (We assume that φ is written in positive-normal form.)

	$R(y, y')$	
$\wedge$	$b_\alpha \rightarrow g(\alpha)(y)$	for any state formula $\alpha \in cl(\varphi)$
$\wedge$	$b_{\alpha\wedge\beta} \rightarrow b_\alpha \wedge b_\beta$	for any path formula $\alpha \wedge \beta \in cl(\varphi)$
$\wedge$	$b_{\alpha\vee\beta} \rightarrow b_\alpha \vee b_\beta$	for any path formula $\alpha \vee \beta \in cl(\varphi)$
$\wedge$	$b_{\mathbf{X}\alpha} \rightarrow b'_\alpha$	for any path formula $\mathbf{X}\alpha \in cl(\varphi)$
$\wedge$	$b_{\alpha\mathbf{U}\beta} \rightarrow b_\beta \vee (b_\alpha \wedge b'_{\alpha\mathbf{U}\beta})$	for any path formula $\alpha\mathbf{U}\beta \in cl(\varphi)$
$\wedge$	$b_{\alpha\mathbf{B}\beta} \rightarrow b_\beta \wedge (b_\alpha \vee b'_{\alpha\mathbf{B}\beta})$	for any path formula $\alpha\mathbf{B}\beta \in cl(\varphi)$

It follows by an inductive proof from the definition of R^0_φ that if the structure $\mathcal{K}^\star$ satisfies the formula,

$$(\exists y'\, \bar{b}\, \overline{b'})(b_\varphi \quad \wedge \quad \mathrm{TC}(R^0_\varphi)(y, \bar{b}, y', \overline{b'}) \quad \wedge \quad \mathrm{TC}^s(R^0_\varphi)(y', \overline{b'}, y', \overline{b'})) \tag{8}$$

then there is a path from y to y' along which φ may be true. The reason we say, "may be," is that there may be some booleans $b'_{\alpha\mathbf{U}\beta}$ that are true, promising that eventually β will become true, but in fact as we walk around the cycle, α remains true but β never becomes true. Essentially, the boolean variables encode only the states of the "local automaton" in [VW94], which does not guarantee the satisfaction of "Until" formulas.

In order to solve this problem, let $\overline{m}$ be a tuple of bits $m_{\alpha\mathbf{U}\beta}$, one for each "Until" formula, $\alpha\mathbf{U}\beta \in cl(\varphi)$. We use the "memory bit" $m_{\alpha\mathbf{U}\beta}$ to check that β actually occurs on the path from y' back to itself. We do this by starting the cycle with $m_{\alpha\mathbf{U}\beta}$ being false and only letting it become true when β holds. Essentially, the memory bits encode the state of the "eventuality automaton" in [VW94].

Define the relation $R_\varphi(y, \bar{b}, \bar{m}, y', \overline{b'}, \overline{m'})$ as follows,

$$
\begin{aligned}
& R^0_\varphi(y, \bar{b}, y', \overline{b'}) \\
\wedge\ & m'_{\alpha \mathbf{U} \beta} \rightarrow (m_{\alpha \mathbf{U} \beta} \ \vee \ b_\beta) \qquad \text{for any formula } \alpha \mathbf{U} \beta \in cl(\varphi)
\end{aligned}
$$

Finally, we define the desired mapping g from CTL* state formulas to FO^2(TC) as follows:

$$
\begin{aligned}
g(p) &= p(y) \\
g(\alpha \wedge \beta) &= g(\alpha) \wedge g(\beta) \\
g(\neg \alpha) &= \neg g(\alpha) \\
g(\mathbf{E}(\varphi)) &= (\exists y' \, \bar{b} \, \overline{b'} \, \overline{m})(\quad b_\varphi \\
& \qquad \wedge \mathrm{TC}(R_\varphi)(y, \bar{b}, \overline{\mathbf{false}}, y', \overline{b'}, \overline{\mathbf{false}}) \\
& \qquad \wedge \mathrm{TC}^s(R_\varphi)(y', \overline{b'}, \overline{\mathbf{false}}, y', \overline{b'}, \overline{m}) \\
& \qquad \wedge b'_{\alpha \mathbf{U} \beta} \ \rightarrow \ m_{\alpha \mathbf{U} \beta} \qquad \text{for any formula } \alpha \mathbf{U} \beta \in cl(\varphi)
\end{aligned}
$$

The following can now be proved by induction on φ,

Theorem 9. *The map g defined above translates* CTL* *state formulas to equivalent formulas in* FO^2(TC)*. That is, for all Kripke structures $\mathcal{K}$, states s, and* CTL* *state formulas φ,*

$$
(\mathcal{K}, s) \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}^\star, s/y) \models g(\varphi) \tag{10}
$$

The transformation g suffers from a similar problem as the transformation f of Theorem 6. The problem is that the formula R_φ is written twice in the definition of $g(\mathbf{E}(\varphi))$. This may cause the size of the formula $g(\gamma)$ to grow exponentially in the nesting depth of path quantifiers $(\mathbf{E}, \mathbf{A})$ in γ. In practice there is little reason for this nesting depth to be greater than one or two. We can, however, alleviate this problem in general as follows:

Corollary 11. *The mapping g above may be modified to run in linear time and thus produce linear size output, in any of the following ways:*

1. *Modify the mapping allowing another variable, that is, map to* FO^3(TC)*.*
2. *Allow the definition of R_φ to be written once and reused, that is, we represent the formula as a first-order circuit.*
3. *Allow the construction, "$R' := \mathrm{TC}^s(R_\varphi)$," that is, whenever we compute the transitive closure of a relation we may reuse it.*

Proof. Items 2 and 3 simply change our mode of representation and are thus obvious. The idea in item 1 is that with an extra state variable t and a universal quantifier, we can eliminate the extra occurrence of R_φ. For example, we can rewrite the definition of $G(\mathbf{E}(\varphi))$ as follows:

$$
\begin{aligned}
g'(\mathbf{E}(\varphi)) = (\exists y' \, \bar{b} \, \overline{b'} \, \overline{m})(\forall t \bar{c} \bar{d})(b_\varphi \ \wedge \\
& (t = y \wedge \bar{c} = \bar{b} \wedge \bar{d} = \overline{\mathbf{false}} \ \vee \ t = y' \wedge \bar{c} = \overline{b'} \wedge \bar{d} = \overline{m}) \quad \rightarrow \\
& \quad \mathrm{TC}^s(R_\varphi)(t, \bar{c}, \overline{\mathbf{false}}, y', \overline{b'}, \overline{d}) \\
& \wedge \quad b'_{\alpha \mathbf{U} \beta} \ \rightarrow \ m'_{\alpha \mathbf{U} \beta} \qquad \text{for any formula } \alpha \mathbf{U} \beta \ \in cl(\varphi)
\end{aligned}
$$

Note: In all of the cases of Corollary 11, the resulting formulas remain in FO(TC) and thus have data complexity NSPACE$[\log n]$. In addition, conditions 2 and 3 are quite feasible from a symbolic model checking point of view: we would naturally compute the OBDD for the relation R_φ and its transitive closure only once.

The use of the finiteness of the Kripke structures, $\mathcal{K}^\star$, in our proofs of Theorems 6 and 9 is crucial. It is known that CTL cannot be translated to FO(TC) over all structures [Ott].

5 Applications to Symbolic Model Checking

The main application of this work is to symbolic model checking. In this situation, the Kripke model is too large to be represented in memory and is instead represented symbolically, often via an OBDD.

From a descriptive point of view, this corresponds to a Kripke structure determined by a set of n boolean variables. Let,

$$
\mathcal{A} \quad = \quad \langle \{x_1, x_2, \ldots, x_n\}, \delta, p_1, \ldots, p_r \rangle \ .
$$

Here the universe of $\mathcal{A}$ is a set of n boolean variables. A state in the corresponding Kripke structure $\mathcal{K}(\mathcal{A})$ is a unary relation S over $\mathcal{A}$, i.e., a truth assignment to the elements of $|A|$. The formula, δ, which might be represented as an OBDD, expresses the transition relation, $\delta(S_1, S_2)$, on states of $\mathcal{K}(\mathcal{A})$. Similarly, the formulas $p_1, \ldots, p_r$ represent the relevant unary relations that are true or false at each state S of $\mathcal{K}(\mathcal{A})$.

Above, we expressed CTL or CTL* conditions concerning a Kripke structure $\mathcal{K}$ in FO^2(TC), that is, in first-order logic with two variables, and a transitive closure operator. In the symbolic setting, such a formula concerning $\mathcal{K}(\mathcal{A})$ is best thought of as a second-order, monadic formula concerning the structure $\mathcal{A}$. That is, the elements of the universe of $\mathcal{K}(\mathcal{A})$ are unary relations over $\mathcal{A}$. Thus, the correctness conditions in question are queries to $\mathcal{A}$ in the language MSO^2(TC) — monadic, second-order formulas, with only two second-order variables, and a transitive closure operator.

It is not hard to see that

Fact 12 NSPACE$[n]$ $=$ MSO(TC).

Thus, the CTL and CTL* queries are all checkable in nondeterministic linear space [BVW94]. Here the space is linear in n, the size of the design of the circuit or protocol to be verified, not 2^n, the size of the Kripke structure $\mathcal{K}(\mathcal{A})$.

It is important in our simulations that we used as few variables as possible. With two second-order, monadic variables, the paths to be checked can have length at most 2^n. Each boolean variable that we add, can at most double the length of such a path, whereas adding another second-order, monadic variable is essentially n boolean variables, and could thus increase the length of paths to be searched by a factor of 2^n. We suspect that the number of boolean variables needed for typical CTL* queries is quite small. It is an interesting open question how many boolean variables are needed in the worst case. (For example, in the context of linear temporal logic, analogous translations are known that use no boolean variables [EVW97].)

Experiments need to be performed concerning practical aspects of using FO(TC) as a language for expressing correctness queries. While the straightforward approach for adopting transitive-clousre algorithms to symbolic model checking have failed [TBK95], more sophisticated transitive-closure algorithms (see [Ya90]) might be quite useful for symbolic model checking.

This work suggests a new paradigm for model checking: One can write the conditions to be checked in a very expressive language, e.g., second-order logic or first-order logic with least-fixed point operators, FO(LFP). Next, if the Kripke structure is small, we may be able to check this condition automatically. If not, we may need to break our correctness conditions down into simpler conditions which may be expressed in simpler languages, e.g., FO(TC), which can be automatically checked in a feasible amount of time. Even within FO(TC), there is a hierarchy of how many varables we need, and how many boolean variables in $\mathrm{FO}^2(\mathrm{TC})$. There is a well-developed theory in the context of finite-model theory of the relationship between descriptive complexity and computational complexity [I89]. This understanding could be also important in computer-aided verification.

6 Conclusions and Future Work

We have shown that every formula in CTL* may be translated in linear time to an equivalent formula in transitive closure-logic, FO(TC). Since the language FO(TC) has data complexity NSPACE[$\log n$], it admits more efficient model checking algorithms than the modal μ-calculus, which has a polynomial-time-complete data complexity

There are several open questions concerning the number of variables needed for the resulting formulas in FO(TC):

1. We have shown that the resulting formulas are linear size when we allow three domain variables, that is they are in $\mathrm{FO}^3(\mathrm{TC})$. It is open whether linear size can be maintained when we map to $\mathrm{FO}^2(\mathrm{TC})$, or indeed, whether an exponential blow-up is required.

2. We would like to know how many boolean variables are needed to interpret $\mathrm{CTL}^\star$ in $\mathrm{FO}^2(\mathrm{TC})$ (our construction allows a linear number of such boolean variables).

Finally, our approach of using transitive-closure logic rather than the much more complex μ-calculus for model checking might be useful in practice. This requires further investigation and testing. Part of the program of Descriptive Complexity is that the computational complexity of query evaluation should be apparent just from looking at the syntax of the query under consideration. Translating $\mathrm{CTL}^\star$ queries into transitive-closure logic rather than μ-calculus facilitates this approach.

Acknowledgements: Thanks to Kousha Etessami and Thomas Wilke for helpful comments, corrections, and suggestions.

References

[BBG94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman and M. Yoel, "Methodology and System for Practical Formal Verification of Reactive Hardware," in *Computer Aided Verification, Proc. 6th Int. Conference,* D. L. Dill, ed., LNCS 818, 1994, Springer-Verlag, 182–193.

[BVW94] O. Bernholtz, M.Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Branching-Time Model Checking," in *Computer Aided Verification, Proc. 6th Int. Conference,* D. L. Dill, ed., LNCS 818, 1994, Springer-Verlag, 142–155.

[BCM92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," *Information and Computation* 98(2) (1992), 142–170.

[CE81] E.M. Clarke and E.A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic," in *Proc. Workshop on Logic of Programs*, LNCS 131, 1981, Springer-Verlag, 52–71.

[CES86] E.M. Clarke, E.A. Emerson and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Transactions on Programming Languages and Systems,* 8(2) (1986), 244-263.

[EF95] H.-D. Ebbinghaus, J. Flum, *Finite Model Theory* 1995, Springer 1995.

[Eme90] E.A. Emerson, "Temporal and modal logic," in *Handbook of theoretical computer science*, 1990, 997-1072.

[Eme97] E. A. Emerson, "Model Checking and the Mu-Calculus," in *Descriptive Complexity and Finite Models,* N. Immerman and Ph. Kolaitis, eds., 1997, American Mathematical Society.

[EL86] E.A. Emerson and C.-L. Lei, "Efficient Model Checking in Fragments of the Propositional mu-Calculus," *Proc. 1st Symp. on Logic in Computer Science* (1986), 267-278.

[EVW97] K. Etessami, M.Y. Vardi, and T. Wilke, "First-Order Logic with Two Variables and Unary Temporal Logic," *Proc. 12th IEEE Symp. on Logic in Computer Science*, July 1997.

[I86] N. Immerman, "Relational Queries Computable in Polynomial Time," *Information and Control,* 68 (1986), 86-104.

[I87] N. Immerman, "Languages That Capture Complexity Classes," *SIAM J. Comput.* 16(4) (1987), 760-778.
[I88] N. Immerman, "Nondeterministic Space is Closed Under Complementation," *SIAM J. Comput.* 17(5) (1988), 935-938.
[I89] N. Immerman, "Descriptive and Computational Complexity,"in *Computational Complexity Theory,* ed. J. Hartmanis, Lecture Notes for AMS Short Course on Computational Complexity Theory, *Proc. Symp. in Applied Math.* 38, American Mathematical Society (1989), 75-91.
[I89a] N. Immerman, *Expressibility and Parallel Complexity, SIAM J. of Comput* **18** (1989), 625-638.
[I91] N. Immerman, "DSPACE$[n^k]$ = VAR$[k+1]$," *Sixth IEEE Structure in Complexity Theory Symp.* (July, 1991), 334-340.
[Koz83] D. Kozen, "Results on the Propositional μ-Calculus," *Theoretical Computer Science*, 27 (1983), 333–354.
[LR96] R. Lassaigne and M. de Rougemont, *Logique et Complexité,* 1996, Hermes.
[LP85] O. Lichtenstein and A. Pnueli, "Checking that Finite State Concurrent Programs Satisfy their Linear Specification" *Proc. 12th ACM Symp. on Principles of Programming Languages* (1985), 97-107.
[McM93] K. McMillan, *Symbolic Model Checking,* 1993, Kluwer.
[Ott] M. Otto, private communication.
[PI94] S. Patnaik and N. Immerman, "Dyn-FO: A Parallel, Dynamic Complexity Class," *Proc. ACM Symp. on Principles of Database Systems* (1994), 210-221.
[QS81] J.P. Queille and J. Sifakis, "Specification and Verification of Concurrent Systems in Cesar," *Proc. 5th Int'l Symp. on Programming,* LNCS 137, 1981, Springer-Verlag, 337–351.
[TBK95] H. J. Touati, R. K. Brayton, and R. P. Kurshan, "Testing language containment for ω-automata using BDD's," *Information and Computation*, 118(1):101–109, 1995.
[Var82] M.Y. Vardi, "Complexity of Relational Query Languages," *ACM Symp. Theory Of Comput.* (1982), 137-146.
[Var97] M.Y. Vardi, "Why is Modal Logic So Robustly Decidable?"in *Descriptive Complexity and Finite Models,* N. Immerman and Ph. Kolaitis, eds., 1997, American Mathematical Society.
[VW84] M.Y. Vardi and P. Wolper, "Yet Another Process Logic," in *Logics of Programs,* LNCS 164, 1984, Springer-Verlag, 501–512.
[VW86] M.Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification," *Proc. 1st Symp. on Logic in Computer Science* (1986), 322–331.
[VW94] M.Y. Vardi and P. Wolper, "Reasoning about Infinite Computations," *Information and Computation* 115(1) (1994), 1-37.
[Ya90] M. Yannakakis, "Graph-theoretic methods in database theory", *Proc. 9th ACM Symp. on Principles of Database Systems*, 230–242, 1990.
[ZSS94] S. Zhang, S.A. Smolka, and O. Sokolsky, "On the Parallel Complexity of Model Checking in the Modal μ-Calculus," *Proc. 9th IEEE Symp. on Logic in Computer Science*, 1994, 154-163.