Reachability Logic: Efficient Fragment of Transitive Closure Logic

Natasha Alechina, *School of Computer Science and IT, University of Nottingham, UK, email nza@cs.nott.ac.uk*

Neil Immerman, *Computer Science Dept., UMass, Amherst, USA, email immerman@cs.umass.edu*

Abstract

We define reachability logic ($\mathcal{RL}$), a fragment of FO^2(TC) (with boolean variables) that admits efficient model checking – linear time with a small constant – as a function of the size of structure being checked. $\mathcal{RL}$ is expressive enough so that modal logics PDL and CTL* can be linearly embedded in it. The model checking algorithm is also linear in the size of the formula, but exponential in the number of boolean variables occurring in it. In practice this number is very small. In particular, for CTL and PDL formulas the resulting model checking algorithm remains linear. For CTL* the complexity of model checking — which is PSPACE complete in the worst case — can be read from the face of the translated formula.

Keywords: graph query languages, model checking, PDL, CTL*

1 Introduction

Many problems in computer science can be reduced to asking questions about paths in a graph: for example, is there a path from one vertex to another; is the graph connected; is it acyclic; is there a path comprised of a certain kind of steps (e.g. described by a regular expression). We are interested in a logical language that can express these kinds of path queries, or reachability queries, and at the same time where expressions can be efficiently evaluated.

The complexity of query evaluation which we are striving for is linear time in the size of the query times the size of the graph (that is, the number of vertices plus the number of edges in the graph). The graphs under consideration are usually large and sparse.

Several different modal logics have been proposed for talking about paths in a graph, for example, PDL, CTL, CTL*, modal μ calculus and others. It is not surprising that modal languages are appropriate for expressing reachability queries. They are usually less complex than a 'classical' logic which encompasses them (e.g. basic modal logic vs first order logic) but they can quantify over reachable objects and thus express reachability queries.

We believe that a natural 'encompassing' logic for path queries is first order logic extended with the transitive closure operator [9, 10]. We are looking for a fragment of it which admits efficient model checking and which can express both PDL and CTL*

expressible properties of graphs (note that these two logics are complementary in the sense that neither is more expressive than the other) [5].

In the next section, we formally introduce transitive closure logic and the structures it is interpreted on. Then we introduce a 'modal' fragment $\mathcal{RL}_0$ of $\mathrm{FO}^2(\mathrm{TC})$ where quantifiers are restricted by path descriptions and which admits linear time model checking. However, PDL and CTL* cannot be linearly embedded in it.

In order to express all interesting reachability queries (and provide a linear embedding for PDL and CTL*) $\mathcal{RL}_0$ needs to be extended with additional variables, which range over the set $\{0,1\}$ and are called boolean variables, or booleans. The resulting fragment is called *Reachability Logic* ($\mathcal{RL}$). The model checking algorithm for $\mathcal{RL}$ is linear in the size of the formula and the size of the model, but exponential in the number of booleans in the formula.

We show that PDL and CTL* can be linearly embedded in $\mathcal{RL}$, which results in a new model checking algorithm for these logics. There is no polynomial embedding of CTL* in $\mathcal{RL}_0$, i.e., without the booleans [1]. The booleans are an important indicator of the complexity of a given CTL* query: they make it possible to distinguish 'difficult' queries (which require lots of booleans) from the 'easy' ones just by looking at the syntax of the query.

It can be shown that there is no embedding of PDL in $\mathcal{RL}$ without the booleans. However, embedding PDL uses at most logarithmically many booleans. We show that the model checking algorithm which translates a PDL formula to $\mathcal{RL}$ and evaluates the result is still linear time in the size of the initial PDL formula. The subset CTL of CTL* can be linearly embedded in $\mathcal{RL}_0$ so a linear model checking algorithm results in this case as well.

2 Transitive Closure Logic

The structures of interest to us in this paper are finite labeled graphs, sometimes called Kripke structures. Let $L = \{a, b, \cdots\}$ be a finite set of edge labels and $\Phi = \{p, q, p_1, q_1, \ldots\}$ be a finite set of propositional symbols (vertex labels).

The language $\mathcal{L}(\Phi, L)$ consists of first-order logic with unary relation symbols $\{p : p \in \Phi\}$, binary relation symbols: $\{R_a : a \in L\}$ and equality $=$.

A Kripke structure of vocabulary (Φ, L) is a finite labeled directed graph:

$$\mathcal{K} \quad = \quad \langle S;\ p^{\mathcal{K}} : p \in \Phi;\ R_a^{\mathcal{K}} : a \in L \rangle$$

where S is the set of states (vertices), each $p^{\mathcal{K}} \subseteq S$ is a unary relation on S and each $R_a^{\mathcal{K}} \subseteq S^2$ is a binary relation on S: the set of edges labeled a. Sometimes when it is clear which graph we are talking about we will omit the superscript $^{\mathcal{K}}$.

For any first-order formula φ, we will write $\mathcal{K} \models \varphi$ to mean that φ is true in $\mathcal{K}$.

The following first-order formula says that there is an edge labeled a from the vertex x to some vertex that satisfies p: $\exists y(R_a(x,y) \wedge p(y))$.

Since the reachability relation is not expressible in first-order logic, we cannot yet write such simple formulas as,

"There is a path of a-edges from x to a vertex where p holds." (*)

We next add a transitive closure operator to first-order logic to allow us to express reachability, cf. [9].

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: $\mathrm{TC}_{\overline{x},\overline{y}}\varphi(\overline{x},\overline{y})$ (or $\mathrm{TC}\,\varphi$ if no confusion is likely to arise). Strict transitive closure is denoted by TC^s. Note that strict transitive closure is definable in terms of TC.

Let FO(TC) be the closure of first-order logic under the transitive-closure operator. For example, the following formula expresses (*):

$$\exists y((\mathrm{TC}_{x,y}R_a(x,y))(x,y) \ \wedge \ p(y)).$$

Let $\mathrm{FO}^2(\mathrm{TC})$ be the restriction of first-order logic with transitive closure in which only two variables may appear in a formula (we call them x and y).

3 $\mathcal{RL}_0$, a Modal fragment of $FO^2(TC)$

To illustrate the idea behind the reachability logic $\mathcal{RL}$ we first introduce a weaker fragment $\mathcal{RL}_0$. In $\mathcal{RL}_0$, quantifiers are restricted by formulas describing paths. More precisely,

DEFINITION 3.1
An *adjacency formula* (without booleans) is a quantifier-free formula that is a disjunction of conjunctions where each conjunct contains at least one of $x = y$, $R_a(x,y)$ or $R_a(y,x)$ for some edge label a; in addition, the conjuncts may contain unary atomic formulas of the form $p(x)$.

Observe that an adjacency formula necessarily implies that there is an edge from x to y or an edge from y to x, or x is equal to y. We allow quantification restricted by an adjacency formula or by a transitive closure of an adjacency formula.

DEFINITION 3.2
$\mathcal{RL}_0$ is the smallest language that satisfies the following:

1. Boolean constants $\top$, $\bot$ are members of $\mathcal{RL}_0$.
2. If p is a unary relation symbol, then $p \in \mathcal{RL}_0$.
3. If $\varphi, \psi \in \mathcal{RL}_0$, then $\neg\varphi \in \mathcal{RL}_0$ and $\varphi \wedge \psi \in \mathcal{RL}_0$.
4. If $\varphi, \psi \in \mathcal{RL}_0$ and q is a new unary predicate symbol, then (**let** $q = \varphi$ **in** ψ) is in $\mathcal{RL}_0$.
5. If $\varphi \in \mathcal{RL}_0$ and δ is an adjacency formula, then the following formulas are in $\mathcal{RL}_0$:
   (a) $\mathrm{NEXT}(\delta)\varphi$
   (b) $\mathrm{REACH}(\delta)\varphi$
   (c) $\mathrm{CYCLE}(\delta)$

Semantics of $\mathcal{RL}_0$: We give the semantics of $\mathcal{RL}_0$ by interpreting each $\mathcal{RL}_0$ construct (on the left) by its meaning in $\mathrm{FO}^2(\mathrm{TC})$ on the right. Here the free variable x always refers to the current position.

$$\begin{aligned} p &\equiv p(x) \\ (\textbf{let } q = \varphi \textbf{ in } \psi) &\equiv \psi[\varphi/q] \end{aligned}$$

$$\begin{aligned} \mathrm{NEXT}(\delta)\varphi &\equiv \exists y(\delta(x,y) \wedge \varphi[y/x]) \\ \mathrm{REACH}(\delta)\varphi &\equiv \exists y(\mathrm{TC}\,\delta)(x,y) \wedge \varphi[y/x]) \\ \mathrm{CYCLE}(\delta) &\equiv (\mathrm{TC}^s\,\delta)(x,x) \end{aligned}$$

The idea behind the logic $\mathcal{RL}_0$ is that we may speak about the current vertex (x), steps out of x, paths out of x, and cycles from x back to itself. $\mathcal{RL}_0$ may be thought of as a modal logic, or as a fragment of $\mathrm{FO}^2(\mathrm{TC})$ in which only the variable x may occur free. Clearly $\mathcal{RL}_0$ is a proper fragment of $\mathrm{FO}^2(\mathrm{TC})$ because $\mathcal{RL}_0$ may not discuss vertices unreachable from the current vertex. For example, the following formula is not expressible in $\mathcal{RL}_0$,

$$\forall y(\mathrm{TC}\,R_a)(x,y)\ .$$

The **let** construct allows for simpler and more modular formulas. It allows us to use a new symbol q for a formula φ and thus may save space in a formula in which we had to write out φ several times. Furthermore, without the **let** construct we could not restrict adjacency formulas to be quantifier free.

The following are a few sample $\mathcal{RL}_0$ formulas and their meanings. Note that the third formula asserts that there exists an infinite chain. Thus $\mathcal{RL}_0$ does not have the finite model property. However, we do restrict our attention in this paper to finite structures.

1. $\neg\mathrm{REACH}(R(x,y))\mathrm{CYCLE}(R(x,y) \wedge p(x) \wedge \neg q(x))$ means that there is no reachable cycle along which p is always true and q is always false. Over finite structures this expresses weak fairness, i.e, there is no infinite path along which a resource is always requested (p) but never granted (q).
2. $\neg\mathrm{REACH}(R(x,y))(p \wedge \mathrm{CYCLE}(R(x,y) \wedge \neg q(x)))$ means that there is no reachable cycle along which p occurs at least once but q never holds. Over finite structures this expresses strong fairness, i.e., there is no infinite path along which a request is made infinitely often but granted only finitely often.
3. $\neg\mathrm{REACH}(R(x,y))(\neg\mathrm{NEXT}(R(x,y))\top \ \vee \ \mathrm{CYCLE}(R(x,y)))$ means that every reachable point has a successor and is not involved in a cycle.
4. **let** $r = \mathrm{REACH}(R_a(x,y))p$ **in** $\mathrm{REACH}(R(x,y) \wedge r)\mathrm{CYCLE}(R(x,y) \wedge r)$ means there is an infinite path for which at all times we can take a path of a-edges to a point where p holds.

$\mathcal{RL}_0$ is unexpectedly similar to other 'bounded quantifier fragments' (see [3]). However, instead of quantifying over objects accessible by an atomic step, in $\mathcal{RL}_0$ we quantify over objects reachable by a path. The reason for restricting quantifiers in $\mathcal{RL}_0$ is not the quest for decidability, as in bounded fragments, but for efficient evaluation. The following theorem illustrates this point.

THEOREM 3.3
There is an algorithm that given a graph G and a formula $\varphi \in \mathcal{RL}_0$ marks the vertices in G that satisfy φ. This algorithm runs in time $O(|G||\varphi|)$.

PROOF. We inductively mark the vertices of G according to whether they satisfy each subformula of φ.

We assume that G is represented by a sorted adjacency list. That is, for each vertex s we have the list of all adjacent vertices, $v_1, \ldots, v_d$. For each v_i on the list we have a

label indicating exactly which edge relations hold, e.g. $R_a(s, v_1)$ and $R_b(v_1, s)$. This structure is linear in the size of G (number of vertices + double the number of edges).

We also assume that we have a list of all subformulas of φ in the order of increasing complexity. We iterate through the list, and for every vertex s mark it according to the following rules:

1. **Base case:** Mark vertex s for p, iff $s \in p^G$.
2. Mark s for $\neg\varphi$ iff s is not marked for φ. Mark s for $\varphi \wedge \psi$ iff s is marked for φ and s is marked for ψ.
3. There are three cases in the remaining part of the marking algorithm:
   (a) Mark s for $\mathrm{NEXT}(\delta(x, y))\varphi$ iff there is an edge from s to a vertex s' that is marked for φ and such that $\delta(s, s')$ holds. Note that the time required to check all adjacent vertices for every vertex is at most the size of δ times the size of the adjacency list.
   (b) Mark s for $\mathrm{REACH}(\delta)\varphi$ iff there is a δ-path to some vertex s' that is marked for φ. This can be tested in linear time by doing a depth first search of G starting from all points marked for φ and proceeding backwards along edges for which δ holds.
   (c) Mark s for $\mathrm{CYCLE}(\delta)(x, x)$ iff s is in a non-trivial strongly connected component of the δ-graph. This can also be checked in linear time using depth first search of G [2].

∎

Although we feel that $\mathcal{RL}_0$ is an interesting fragment of $\mathrm{FO}^2(\mathrm{TC})$ in itself, and it admits linear time model checking, we need to show that $\mathcal{RL}_0$ can express interesting queries. Unfortunately, we cannot show this by (linearly) embedding modal logics PDL and CTL* in $\mathcal{RL}_0$. Indeed, we next show that that PDL is embeddable neither in $\mathcal{RL}_0$ nor in full $\mathrm{FO}^2(\mathrm{TC})$ without booleans.

PROPOSITION 3.4
PDL cannot be embedded in $\mathcal{RL}_0$, nor in $\mathrm{FO}^2(\mathrm{TC})$ without booleans.

PROOF. Consider the PDL formula $\mathrm{EVEN} \equiv \langle (a; a)^\star \rangle \neg \langle a \rangle \top$ meaning that there is an even length path of a's from where we are to a point that has no a-edge out of it. If $\mathrm{FO}^2(\mathrm{TC})$ without booleans is interpreted over finite successor structures, then every $\mathrm{FO}^2(\mathrm{TC})$ formula is equivalent to a two-variable first order formula with order. (Note that no such formula can express the property that the distance from x to y is two.) But over finite orderings, EVEN is not expressible in first order logic [1]. ∎

4 Reachability Logic

In order to provide linear embeddings of PDL and CTL* in $\mathrm{FO}^2(\mathrm{TC})$ we need to introduce additional expressive power. In addition to ordinary (or domain variables, which range over the domain of the input structure, we allow *boolean variables* $b, c, d, b_1, \ldots$. 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.

[1] We thank the anonymous referee who suggested this simpler version of our original proof.

In what follows, we will assume that $\mathrm{FO}^2(\mathrm{TC})$ may contain boolean variables. We modify the definition of an adjacency formula as follows:

DEFINITION 4.1
An *adjacency formula* (with booleans) is a disjunction of conjunctions where each conjunct contains at least one of $x = y$, $R_a(x, y)$ or $R_a(y, x)$ for some edge label a; in addition, the conjuncts may contain expressions of the form $(\neg)(b_1 = b_2)$, $(b_1 = 0)$, $(b_1 = 1)$ and $p(x)$, where b_1 and b_2 are boolean variables.

DEFINITION 4.2
$\mathcal{RL}$ is the smallest fragment of $\mathrm{FO}^2(\mathrm{TC})$ that satisfies the following:

1. If p is a unary relation symbol then $p \in \mathcal{RL}$; also $\top, \bot \in \mathcal{RL}$.
2. If $\varphi, \psi \in \mathcal{RL}$, then $\neg\varphi \in \mathcal{RL}$ and $\varphi \wedge \psi \in \mathcal{RL}$.
3. If $\varphi \in \mathcal{RL}$ and b is a boolean variable, then $\exists b\varphi \in \mathcal{RL}$.
4. If $\varphi, \psi \in \mathcal{RL}$ and q is a new unary predicate symbol, then (**let** $q = \varphi$ **in** ψ) is in $\mathcal{RL}$.
5. If $\varphi \in \mathcal{RL}$ and $\delta(x, \overline{b}, y, \overline{b'})$ is an adjacency formula (a binary relation between two n-tuples $\langle x, b_1, \ldots, b_{n-1}\rangle$ and $\langle y, b'_1, \ldots, b'_{n-1}\rangle$), then the following formulas are in $\mathcal{RL}$:
   (a) $\mathrm{NEXT}(\delta)\varphi$
   (b) $\mathrm{REACH}(\delta)\varphi$
   (c) $\mathrm{CYCLE}(\delta)$

Semantics of $\mathcal{RL}$: The semantics of $\mathcal{RL}$ is similar to that of $\mathcal{RL}_0$, the only difference being the use of booleans in adjacency formulas. In each case below assume that $\delta(x, \overline{b}, y, \overline{b'})$ is an adjacency formula.

$$\begin{aligned} p &\equiv p(x) \\ (\textbf{let } q = \varphi \textbf{ in } \psi) &\equiv \psi[\varphi/q] \\ \mathrm{NEXT}(\delta)\varphi &\equiv \exists y(\delta(x, \overline{0}, y, \overline{1}) \wedge \varphi[y/x]) \\ \mathrm{REACH}(\delta)\varphi &\equiv \exists y(\mathrm{TC}\,\delta)(x, \overline{0}, y, \overline{1}) \wedge \varphi[y/x]) \\ \mathrm{CYCLE}(\delta) &\equiv (\mathrm{TC}^s\,\delta)(x, \overline{0}, x, \overline{0}) \end{aligned}$$

Here are some examples of formulas in $\mathcal{RL}$:

- $\mathrm{REACH}(\delta)p$ where $\delta(x, b_1, b_2, y, b'_1, b'_2)$ is $(R_a(x, y) \wedge b_1b_2 = 00 \wedge b'_1b'_2 = 01) \vee (R_b(x, y) \wedge b_1b_2 = 01 \wedge b'_1b'_2 = 11)$ (this is $\langle a; b\rangle p$ of PDL, see Section 6).
- $\varphi_1 = \mathrm{REACH}(R)p$ (**EF**p of CTL*, see Section 5);
- $\varphi_2 = \mathrm{REACH}(\delta)\mathrm{CYCLE}(\delta)$, where δ is $R(x, y) \wedge q(x)$ (**EG**q of CTL*);
- (**let** $q = \varphi_1$ *in* φ_2) (**EGEF**p of CTL*).

$\mathcal{RL}$ is a logical language and it is a fragment of $\mathrm{FO}^2(\mathrm{TC})$. However, because of the 'let' construct, when we talk about size in the representation of $\mathcal{RL}$, we are really talking about circuits. Thus the size of an $\mathcal{RL}$-circuit may be logarithmic in the size of the smallest equivalent $\mathrm{FO}^2(\mathrm{TC})$ formula. This allows the linear size embedding of

CTL* which presumably does not hold for $\mathrm{FO}^2(\mathrm{TC})$ (without a circuit representation or an extra domain variable cf. [10]).

Boolean variables however add extra complexity, which is not surprising since model checking CTL* is PSPACE complete [13].

THEOREM 4.3
There is an algorithm that given a graph G and a formula $\varphi(x) \in \mathcal{RL}$ marks the vertices in G that satisfy φ. This algorithm runs in time $O(|G||\varphi|2^{n_b})$ where n_b is the number of boolean variables occurring in φ.

PROOF. As for $\mathcal{RL}_0$, we inductively mark the vertices of G according to whether they satisfy each subformula of φ. For subformulas that include a free boolean variable, we include this subformula with both substitutions of the boolean variable, thus with n_b booleans there could be as many as 2^{n_b} copies of some subformulas.

In $\mathcal{RL}$, formulas talk about transitions between tuples of the form (node, sequence of booleans). In addition to G, we need to maintain an adjacency list corresponding to the extended graph $\mathcal{G}$, where nodes correspond to old nodes followed by sequences of 0s and 1s of length n_b. $\mathcal{G}$ is exponentially (in the number of booleans) larger than G.

We also assume that we have a list of all subformulas of φ in the order of increasing complexity. We iterate through the list, and for every vertex s mark it as we did for $\mathcal{RL}_0$. The new cases are:

- Mark s for $(\exists b)\varphi$ iff s is marked for at least one of $\varphi(0/b)$ or $\varphi(1/b)$.
- For the formula (**let** $q = \varphi$ **in** ψ), we have inductively marked states according to whether they satisfy φ. Thus, our Kripke structure is expanded to interpret the new predicate symbol q, true of those states marked for φ. Now we evaluate the smaller formula ψ on this expanded structure.
- The cases of $\mathrm{NEXT}(\delta)\varphi$, $\mathrm{REACH}(\delta)\varphi$, and $\mathrm{CYCLE}(\delta)$ are the same as for $\mathcal{RL}_0$, but we do depth first search of $\mathcal{G}$ instead of G. For the last clause, we check that $s, \overline{0}$ is in a non-trivial strongly connected component of the δ-subgraph of $\mathcal{G}$.

It is easy to see that the above marking algorithm is correct and runs in the required time. ∎

5 Embedding CTL* in $\mathcal{RL}$

A popular and quite expressive language for Model Checking is computation tree logic CTL*. CTL* is a version of temporal logic that combines linear and branching time. CTL* has two kinds of formulas: *state formulas*, which are true or false at each state, and *path formulas*, which are true or false with respect to an infinite path through $\mathcal{K}$. The following is an inductive definition of the state and path formulas of CTL*.

DEFINITION 5.1
(Syntax of CTL***)** State formulas $\mathcal{S}$ and path formula $\mathcal{P}$ of CTL* are the smallest sets of formulas satisfying the following:
State Formulas, $\mathcal{S}$:
the boolean constants $\top$ and $\bot$ are elements of $\mathcal{S}$;
if $p_i \in \Phi$, then $p_i \in \mathcal{S}$;
if $\varphi \in \mathcal{P}$, then $\mathbf{E}\varphi \in \mathcal{S}$.

Intuitively, $\mathbf{E}\varphi$ means that there exists an infinite path starting at the current state and satisfying φ.

Path Formulas, $\mathcal{P}$:
if $\alpha \in \mathcal{S}$ then $\alpha \in \mathcal{P}$;
if $\varphi, \psi \in \mathcal{P}$, then $\neg\varphi, \varphi \wedge \psi$, $\mathbf{X}\varphi$, and $\varphi\mathbf{U}\psi$ are in $\mathcal{P}$.

Intuitively, $\mathbf{X}\varphi$ means that φ holds at the next moment of time and $\varphi\mathbf{U}\psi$ means that at some time now or in the future, ψ holds, and from now until then, φ holds.

Next, we formally define the semantics of the above operators. A path $\rho = \rho_0, \rho_1, \ldots$ is a mapping of the natural numbers to states in $\mathcal{K}$ such that for all i, $\mathcal{K} \models R(\rho_i, \rho_{i+1})$. We use the notation ρ^i for the tail of ρ, with states $\rho_0, \rho_1, \ldots, \rho_{i-1}$ removed.

DEFINITION 5.2
(Semantics of CTL*) The following are inductive definitions of the meaning of CTL* formulas:
State Formulas:

$$\begin{aligned}(\mathcal{K}, s) \models p_i \quad &\text{iff} \quad \mathcal{K} \models p_i(s)\\ (\mathcal{K}, s) \models \mathbf{E}\varphi \quad &\text{iff} \quad \exists \text{ path } \rho(\rho_0 = s \wedge (\mathcal{K}, \rho) \models \varphi)\end{aligned}$$

Path Formulas:

$$\begin{aligned}(\mathcal{K}, \rho) \models \alpha \quad &\text{iff} \quad (\mathcal{K}, \rho_0) \models \alpha \text{ for } \alpha \in \mathcal{S}\\ (\mathcal{K}, \rho) \models \varphi \wedge \psi \quad &\text{iff} \quad (\mathcal{K}, \rho) \models \varphi \text{ and } (\mathcal{K}, \rho) \models \psi\\ (\mathcal{K}, \rho) \models \neg\varphi \quad &\text{iff} \quad (\mathcal{K}, \rho) \not\models \varphi\\ (\mathcal{K}, \rho) \models \mathbf{X}\varphi \quad &\text{iff} \quad (\mathcal{K}, \rho^1) \models \varphi\\ (\mathcal{K}, \rho) \models \varphi\mathbf{U}\psi \quad &\text{iff} \quad \exists i((\mathcal{K}, \rho^i) \models \psi \wedge (\forall j < i)(\mathcal{K}, \rho^j) \models \varphi)\end{aligned}$$

THEOREM 5.3
There is a linear-time computable function g that maps any CTL* formula φ to an equivalent formula $g(\varphi) \in \mathcal{RL}$. While $g(\varphi)$ has only two domain variables, it may have a linear number of boolean variables.

PROOF. We review the proof from [10] that CTL* is linearly embeddable in FO^2(TC) and show that the embedding lands in $\mathcal{RL}$.

Let $\mathbf{E}(\varphi)$ be a CTL* formula in which the "$\neg$"s have been pushed inside as far as possible subject to the fact that all path quantifiers should be $\mathbf{E}$s. For this purpose we will need the temporal operator $\mathbf{B}$, the dual of $\mathbf{U}$,

$$\varphi\mathbf{B}\psi \quad \equiv \quad \neg(\neg\varphi\mathbf{U}\neg\psi)$$

The intuitive meaning of $\varphi\mathbf{B}\psi$ is that "φ holds before ψ fails."[2]

Inductively assume that we have computed $g(\alpha)$ for every state subformula of φ. We can then use the "let" rule of $\mathcal{RL}$ to replace $g(\alpha)$ by a new unary relation symbol. We can thus assume that φ has no path quantifiers.

[2] Other authors use "$\mathbf{R}$" for the dual of $\mathbf{U}$ and say that, "φ releases ψ," (from the obligation of holding in the future).

Define the *closure* of φ ($cl(\varphi)$) to be the set of all 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 [14]). We do not need booleans for state formulas but we use them just to make the following inductive definition simpler.

Let $\overline{b}$ be a tuple of all the boolean variables b_α, for $\alpha \in cl(\varphi)$. Define the transition relation $\delta^0_\varphi(y, \overline{b}, y', \overline{b'})$ as follows. In each case, the comment on the right is the condition under which the given conjunct is included in the formula. (We assume that φ is written in positive-normal form.)

$$
\begin{array}{lcll}
 & R(x,y) & & \\
\wedge & b_\alpha = 1 \ \rightarrow g(\alpha)(x) & & \text{for any state formula } \alpha \in cl(\varphi) \\
\wedge & b_{\alpha\wedge\beta} = 1 \ \rightarrow b_\alpha = 1 \wedge b_\beta = 1 & & \text{for any path formula } \alpha \wedge \beta \in cl(\varphi) \\
\wedge & b_{\alpha\vee\beta} = 1 \ \rightarrow b_\alpha = 1 \vee b_\beta = 1 & & \text{for any path formula } \alpha \vee \beta \in cl(\varphi) \\
\wedge & b_{\mathbf{X}\alpha} = 1 \ \rightarrow b'_\alpha = 1 & & \text{for any path formula } \mathbf{X}\alpha \in cl(\varphi) \\
\wedge & b_{\alpha\mathbf{U}\beta} = 1 \ \rightarrow b_\beta = 1 \vee (b_\alpha = 1 \wedge b'_{\alpha\mathbf{U}\beta} = 1) & & \text{for any path formula } \alpha\mathbf{U}\beta \in cl(\varphi) \\
\wedge & b_{\alpha\mathbf{B}\beta} = 1 \ \rightarrow b_\beta = 1 \wedge (b_\alpha = 1 \vee b'_{\alpha\mathbf{B}\beta} = 1) & & \text{for any path formula } \alpha\mathbf{B}\beta \in cl(\varphi)
\end{array}
$$

It follows that if $b_\varphi = 1$, then an infinite δ^0_φ-path starting at $(\overline{b}, x)$ may satisfy φ. However, there could 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 a cycle, α remains true but β never becomes true.

In order to solve this problem, let $\overline{m}$ be a tuple of bits m_β, one for each "Until" formula, $\alpha\mathbf{U}\beta \in cl(\varphi)$. We use the "memory bit" m_β to check that β actually occurs by starting it at 0 and only letting it become 1 when β becomes true.

Let $\overline{d}$ be a set of $|cl(\varphi)|$ "destination" bits and let c_0, c_1 be "control" bits. Define the adjacency formulas δ_1 and δ_2 as follows. They imply that δ_1 starts with $b_\varphi = 1$ and ends at $\overline{b} = \overline{d}$. Similarly δ_1 starts with $\overline{b} = \overline{d}$ and the memory bits all zero and ends with $\overline{b} = \overline{d}$ and the memory bits all one:

$$
\begin{array}{rcl}
\delta_1(\overline{c}, \overline{b}, x, \overline{c'}, \overline{b'}) & \equiv & (\overline{c} = 00 \wedge \overline{c'} = 01 \wedge b'_\varphi = 1 \wedge x = y) \\
 & & \vee \quad (\overline{c} = \overline{c'} = 01 \wedge \delta^0_\varphi(\overline{b}, x, \overline{b'}, y)) \\
 & & \vee \quad (\overline{c} = 01 \wedge \overline{c'} = 11 \wedge \overline{b} = \overline{d} \wedge \overline{b'} = \overline{1} \wedge x = y) \\
 & & \\
\delta_2(\overline{c}, \overline{b}, \overline{m}, x, \overline{c'}, \overline{b'}, \overline{m'}) & \equiv & (\overline{c} = 00 \wedge \overline{c'} = 01 \wedge \overline{b'} = \overline{d} \wedge \overline{m} = \overline{0} \wedge x = y) \\
 & & \vee \quad (\overline{c} = \overline{c'} = 01 \wedge \delta^0_\varphi(\overline{b}, x, \overline{b'}, y) \\
 & & \qquad \wedge (m'_\beta = 1 \rightarrow (m_\beta = 1 \vee b_\beta = 1)) \quad \text{for any } \alpha\mathbf{U}\beta \in cl(\varphi)\ ) \\
 & & \vee \quad (\overline{c} = 01 \wedge \overline{b} = \overline{d} \wedge \overline{c'} = \overline{0} \wedge \overline{m'} = \overline{b'} = \overline{0} \wedge x = y \\
 & & \qquad \wedge (b_{\alpha\mathbf{U}\beta} = 1 \rightarrow m_\beta = 1) \quad \text{for any } \alpha\mathbf{U}\beta \in cl(\varphi)\ )
\end{array}
$$

We define the desired mapping g from CTL* state formulas to $\mathcal{RL}$ as follows:

$$g(\mathbf{E}\varphi) \equiv \exists\overline{d}\mathrm{REACH}(\delta_1)\mathrm{CYCLE}(\delta_2)$$

By construction, $g(\mathbf{E}\varphi)$ asserts that there is an infinite path along which φ holds, as desired. ∎

6 Embedding PDL in $\mathcal{RL}$

Propositional Dynamic Logic (PDL) was introduced in [12] as a logic to reason about programs. The language of PDL includes two sets of primitive symbols: a set of propositional symbols and a set of atomic transitions. Propositional symbols stand for properties that can be true or false for a node in a graph (in the original interpretation of PDL, they are properties of states in the execution of a program). Atomic transitions (edge labels) are interpreted in PDL as basic instructions, e.g., assignment statements.

DEFINITION 6.1
(Syntax of PDL)
Transition terms, $\mathcal{T}$:
elements of L (edge labels) are transition terms;
if $t \in \mathcal{T}$, then $t^* \in \mathcal{T}$;
if $t_1, t_2 \in \mathcal{T}$, then $t_1; t_2$ and $t_1 \cup t_2$ are in $\mathcal{T}$;
if φ is a formula, then $\varphi? \in \mathcal{T}$.
Intuitively, ';' corresponds to sequential composition, '$\cup$' to non-deterministic choice, '$*$' to finite iteration of unspecified length and '$\varphi?$' to test for φ.
Formulas, $\mathcal{F}$:
the boolean constants $\top$ and $\bot$ are elements of $\mathcal{F}$;
if $p_i \in \Phi$, then $p_i \in \mathcal{F}$;
if $\varphi, \psi \in \mathcal{F}$ and $t \in \mathcal{T}$, then $\neg\varphi, \varphi \wedge \psi$, $\langle t \rangle \varphi$ are in $\mathcal{F}$.
The formula $\langle t \rangle \varphi$ means 'after some transition t, φ holds' .

For example, $\langle a* \rangle \neg \langle b \rangle \top$ means that after 0 or finitely many a links, one can reach a node that has no outgoing links labeled b.

The language of PDL is interpreted by Kripke structures.

DEFINITION 6.2
(Semantics of PDL)
The meaning of transition terms is given by the following function tr:
$tr(a) = R_a$
$tr(t_1 \cup t_2) = tr(t_1) \cup tr(t_2)$
$tr(t*)$ is the reflexive, transitive closure of $tr(t)$
$tr(t_1; t_2) = \{(u, v) : \exists z (tr(t_1)(u, z) \wedge tr(t_2)(z, v))\}$
$tr(\varphi?) = \{(u, u) : \varphi \text{ is true at } u\}$
The following are inductive definitions of the meaning of PDL formulas:

$$\begin{aligned}
(\mathcal{K}, s) \models p_i \quad &\text{iff} \quad \mathcal{K} \models p_i(s) \\
(\mathcal{K}, s) \models \varphi \wedge \psi \quad &\text{iff} \quad (\mathcal{K}, s) \models \varphi \text{ and } (\mathcal{K}, s) \models \psi \\
(\mathcal{K}, s) \models \neg\varphi \quad &\text{iff} \quad (\mathcal{K}, s) \not\models \varphi \\
(\mathcal{K}, s) \models \langle t \rangle \varphi \quad &\text{iff} \quad \exists s' ((s, s') \in tr(t) \text{ and } (\mathcal{K}, s') \models \varphi)
\end{aligned}$$

PDL model checking is linear time, i.e., $O(|K| \cdot |\varphi|)$. This follows from the fact that PDL can be linearly embedded into alternation-free mu-calculus circuits [5], and the latter can be model checked in linear time [4]. In this section we show how to linearly embed PDL into a portion of $\mathcal{RL}$ which admits linear time model checking.

We begin by informally showing how to model check PDL in linear time and then how to preserve this linear-time algorithm as we first map PDL to $\mathcal{RL}$ and then model-check the resulting formula.

We are given a PDL formula φ, and a Kripke structure, $\mathcal{K}$. We want to mark each state of $\mathcal{K}$ according to whether it satisfies each subformula of φ.

The only tricky case is $\langle\alpha\rangle\psi$, where we assume that we have inductively marked all states satisfying ψ. The formula α is a regular expression which can be translated to an NFA N_α of size $|N_\alpha| = O(|\alpha|)$.

Next, we can mark all states of $\mathcal{K}$ such that a string in $\mathcal{L}(N_\alpha)$ can take them to a state marked ψ. This is just a depth-first search of $\mathcal{K} \times N_\alpha$, taking time $O(|K| \cdot |N|)$.[3] The desired time $O(|K| \cdot |\varphi|)$ model checking algorithm for PDL results.

The above linear-time model checking algorithm for PDL suggests a natural way to translate PDL to $\mathcal{RL}$. Define a linear translation h from PDL to $\mathcal{RL}$ as follows:

$$\begin{array}{rcl} h(p) & = & p \\ h(\alpha \wedge \beta) & = & h(\alpha) \wedge h(\beta) \\ h(\neg\alpha) & = & \neg h(\alpha) \\ h(\langle\alpha\rangle\psi) & = & \mathrm{REACH}(\delta_\alpha)h(\psi) \end{array}$$

The only interesting case in the above definition of h is the last. Here, δ_α is a translation of the transition relation of N_α using $\log|\alpha| + O(1)$ pairs of booleans to encode the state. Clearly δ_α can be written in disjunctive normal form in size and time that is linear in the size of N_α.[4] Each clause of δ_α is of the form

$$\bar{a} = \bar{c} \ \wedge \ \overline{a'} = \bar{d} \wedge R_a(x, y)$$

where N_α has a transition from state $\bar{c}$ to state $\bar{d}$ reading the letter a. We encode the start state as $\bar{0}$ a tuple of zero's and similarly we can have N_α have a unique final state and encode it as $\bar{1}$.

It is easy to see that,

LEMMA 6.3
The mapping h from PDL to $\mathcal{RL}$ defined above is computable in linear time. Furthermore, for all PDL formulas φ, and Kripke structures $\mathcal{K}$ with states s, we have that,

$$\mathcal{K}, s \models \varphi \quad \Leftrightarrow \quad \mathcal{K}, s \models h(\varphi) \ ,$$

It is intuitively clear that the resulting formula $h(\varphi) \in \mathcal{RL}$ can be model checked in linear time.

LEMMA 6.4
Model checking for formulas in the image of PDL under the mapping h is linear.

PROOF. Let $\varphi \in \mathcal{RL}$ be a translation of a PDL formula. Observe that all adjacency formulas δ in φ are in complete DNF, meaning that for each clause t, and each boolean variable b, one of $b = 0$ or $b = 1$ occurs in t.

[3] Note that if α includes a subformula of the form "$\gamma?$" then inductively we have marked the states of $\mathcal{K}$ according to whether or not they satisfy γ. For a state that satisfies γ, $\gamma?$ is a possible move that leaves the state of $\mathcal{K}$ fixed, whereas for a state that does not satisfy γ, $\gamma?$ is not possible.

[4] Here we are assuming that the size of an equation of the form $\bar{a} = \bar{c}$ is $O(1)$ where $\bar{a}$ and $\bar{c}$ are $\log n$-tuples of booleans. This is consistent with the unit cost for operations on $\log n$-bit words that underly most linear-time algorithms [2].

The model checking algorithm is simpler than the algorithm for the full $\mathcal{RL}$ with no free boolean variables. Again the only interesting case is (5). We know that for all formulas which are translations of PDL formulas under h, there are no occurrences of NEXT or CYCLE, and all occurrences of REACH are of the form

$$\varphi \equiv \text{REACH}(\delta_\alpha)h(\psi)$$

We may assume that we have inductively marked the states satisfying $h(\psi)$. As in the model checking algorithm for $\mathcal{RL}$, we replace the graph G with a larger graph $\mathcal{G}$ where each node is an original node of G followed by a sequence of 0s and 1s. Note that $\mathcal{G} = G \times N_\alpha$ is of size at most $O(|G| \cdot |\delta|)$. The linear-time model checking algorithm results. ∎

7 Conclusions

We have shown that PDL and CTL* can be embedded in reachability logic, $\mathcal{RL} \subseteq \text{FO}^2(\text{TC})$. $\mathcal{RL}$ can be efficiently model checked in time $O(\|K\||\varphi|2^{n_b})$: linear in the size of the structure and the formula, but exponential in the number of booleans in the formula. Furthermore, for PDL and CTL this algorithm is linear, i.e., $O(\|K\||\varphi|)$.

This is useful, both because n_b tends to be tiny, and because the language involved is closely tied to reachability queries which are the bread and butter of model checking. One nice feature is that we can look at the formula, count the number of booleans, and automatically say whether the query can be checked efficiently or not. (Recall that model checking CTL* is PSPACE complete [13]. Thus there presumably are some CTL* queries that are not feasible. An advantage of translating to $\mathcal{RL}$ is that we can see whether or not it is feasible on the face of the resulting query.)

We believe that model checking using $\mathcal{RL}$ may be more efficient than using the μ-calculus in many practical cases. The following directions should be investigated concerning model checking via transitive closure logic:

- Dynamic model checking strategies are needed, i.e. we should often be able to efficiently recompute a model checking query after a small change in the design being checked.
- We have only shown that explicit model checking for $\mathcal{RL}$ is efficient. We believe the same will be true of symbolic model checking but this needs to be investigated.
- From Savitch's theorem, we know that reachability is contained in DSPACE[$\log^2 n$]. The time/space tradeoff for computing reachability should be investigated and exploited in model checking.

Acknowledgement:

This research was supported in part by NSF grant CCRR-9877078.

References

[1] M. Adler and N. Immerman. An $n!$ Lower Bound On Formula Size. Manuscript, 1999.

[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[3] H. Andréka, J. van Benthem, and I. Németi, Modal Languages and Bounded Fragments of Predicate Logic. Technical Report ML-96-03, ILLC, University of Amsterdam, 1996.

[4] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. *Formal Methods in System Design 2*, 121-147, 1993.

[5] E.A. Emerson and C.-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proc. LICS'86*, 267-278, 1986.

[6] K. Etessami and N. Immerman. Tree Canonization and Transitive Closure. To appear in *Information and Computation*. A preliminary version appeared in *Proc. LICS'95*, 331-341, 1995.

[7] K. Etessami and T. Wilke. An Until Hierarchy for Temporal Logic. In *Proc. LICS'96*, 1996.

[8] N. Immerman. *Descriptive Complexity*, Springer Graduate Texts in Computer Science, New York, 1998.

[9] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16(4):760–778, 1987.

[10] N. Immerman and M. Vardi, Model Checking and Transitive Closure Logic. *Proc. 9th Int'l Conf. on Computer-Aided Verification* (CAV'97), Lecture Notes in Computer Science, Springer-Verlag 291 - 302, 1997.

[11] N. Immerman, DSPACE[n^k]=VAR[$k+1$]. *Sixth IEEE Structure in Complexity Theory Symposium*, 334-340, 1991.

[12] V. R. Pratt, Semantical considerations on Floyd-Hoare logic. In *Proc. 17th IEEE Symposium on Foundations of Computer Science*, pages 109–121, 1976.

[13] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. In *JACM,* 32(3):733-749, 1985.

[14] M. Y. Vardi and P. Wolper. Reasoning about Infinite Computations. In *Information and Computation*, 115(1):1-37, 1994.

Received February 22, 2000