On Solving Temporal Logic Queries

S. Hornus* and Ph. Schnoebelen

Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex France
email: phs@lsv.ens-cachan.fr

Abstract. Temporal query checking is an extension of temporal model checking where one asks what propositional formulae can be inserted in a temporal query (a temporal formula with a placeholder) so that the resulting formula is satisfied in the model at hand.
We study the problem of computing all minimal solutions to a temporal query without restricting to so-called "valid" queries (queries guaranteed to have a unique minimal solution). While this problem is intractable in general, we show that deciding uniqueness of the minimal solution (and computing it) can be done in polynomial-time.

1 Introduction

Temporal model checking. Pnueli pioneered the use of *temporal logic* as a formal language for reasoning about reactive systems [Pnu77]. Temporal logic allows *model checking*, i.e. the automatic verification that a finite-state model of the system satisfies its temporal logic specifications [CGP99,BBF$^+$01]. One limitation of model checking is that it gives simple "yes-or-no" answers: either the system under study satisfies the temporal formula, or it does not [1].

Temporal queries. In a recent paper [Cha00], Chan proposed *temporal logic queries* as an extension of model checking (inspired by database queries). A temporal query is a temporal formula $\gamma(?)$ where the special symbol ? occurs as a placeholder (a query can have at most one ?). A propositional formula f is a *solution* to the query if the system satisfies $\gamma(f)$ (γ with f in place of ?), and *temporal query evaluation* is the process of computing the solutions to a temporal query.

When applicable, query answering extends model checking and is a more powerful and versatile tool for validating, debugging, and more generally understanding the formal model of a system [BG01].

---

* Now at Lab. GRAVIR, INRIA Rhône-Alpes, Montbonnot, France. Email: `Samuel.Hornus@inria.fr`. The research described in this paper was conducted while S. Hornus was at LSV.

[1] When the system does not satisfy the temporal formula, most model checkers provide a *diagnostic*, e.g. as a trace showing one possible way of violating the property.

Let us illustrate this with an example. Assume we are currently designing some system and rely on model checking to verifies that it satisfies a typical temporal specification such as

$$\mathsf{G}(\texttt{request} \Rightarrow \mathsf{F}\,\texttt{grant}) \tag{S}$$

stating that "all requests are eventually granted". Model checking will provide a yes-or-no answer: either the system satisfies (S) or it does not.

Temporal queries lead to a finer analysis of the system. The query

$$\mathsf{G}(? \Rightarrow \mathsf{F}\,\texttt{grant}) \tag{Q}$$

asks for the conditions that always inevitably lead to `grant`. Computing the solutions to (Q) for our system will tell us, among other things, whether the system satisfies its specification: (S) is satisfied iff `request` is a solution to (Q). But it will tell us more. For example, if (S) is not satisfied, answering (Q) can lead to the discovery that, in reality, the property that our system satisfies is $\mathsf{G}((\texttt{request} \land \texttt{free}) \Rightarrow \mathsf{F}\,\texttt{grant})$. Dually, if (S) is satisfied, query answering can lead to the discovery that, say, $\top$ is a solution to (Q), i.e. $\mathsf{G}\,\mathsf{F}\,\texttt{grant}$ is satisfied (grants need no request, likely to be a severe bug in our design).

We refer to [Cha00,BG01] for more motivations and examples on the role temporal queries can play in the modeling and validation of systems.

Minimal solutions. In general, evaluating a query γ on a system S will lead to many solutions. However, these solutions can be organized because of a fundamental monotonicity property: Say that γ is a *positive* (resp. a *negative*) query if the placeholder ? occurs under an even (resp. odd) number of negations in γ. Assume f is a solution to a positive γ and f' is a logical consequence of f, then f' too is a solution. (For negative queries, the implication goes the other way around.) Therefore, the set of solutions to a positive γ can be represented by its minimal elements, the *minimal solutions* (called *maximally strong* solutions in [BG01]). From a practical viewpoint, these solutions are the most informative [Cha00,BG01]. In our earlier example, (Q) is a negative query, and the minimal solutions are the weakest conditions that always ensure an eventual `grant`.

Unique minimal solutions. Chan proved monotonicity for CTL queries and introduced a notion of so-called *valid* queries, i.e. queries that always have a *unique minimal solution* for every system. He further defined CTL^v, a special subset that only contains valid CTL queries, and proposed a special-purpose algorithm for answering CTL^v queries.

While uniqueness of the minimal solution is a desirable feature (it certainly provides more intelligible answers), the fragment CTL^v is quite restricted. To begin with, CTL^v does not contain all valid queries. But the very notion of valid queries is a limitation by itself: for example, queries as simple as $\mathsf{EF}?$ and $\mathsf{AF}?$ are not valid. This limitation comes from the fact that valid queries have a unique minimal solution when evaluated over any system, while we are more interested in queries that have a unique minimal solution *when evaluated over the system S at hand*. And when a query does not have a unique minimal solution over some S, we might be interested in knowing what are these several minimal solutions.

This prompted Bruns and Godefroid to study how one can compute, in a systematic way, the set of all minimal solutions to a query [BG01]. They showed how the automata-theoretic approach to model checking can be adapted to provide a *generic* algorithm computing all minimal solutions (generic in the sense that it works for all temporal logics). From a computational complexity viewpoint, their solution is in principle as costly as trying all potential solutions.

Our contribution. In this paper we study the problem of computing the set of minimal solutions to arbitrary temporal queries and look for feasible solutions. We show this problem can be computationally expensive: a query can have a huge number of minimal solutions and simply counting them is provably hard.

This does not mean all problems related to temporal query evaluation are intractable. Indeed, our main contribution is to show that deciding whether a given query has a unique minimal solution over a given system, and computing this solution, can be solved efficiently. Here "efficiently" means that we can reduce this to a linear number of model checking problems.

These methods are useful in more general situations. When there is not a unique minimal solution, we can compute a first minimal solution with a linear number of model checking calls, then compute a second minimal solution with a quadratic number of calls, then a third with a cubic number, etc.: every additional solutions costs more. (We provide hardness results showing that some form of increasing costs is inescapable.)

We see these result as an *a posteriori* explanation of why it is desirable that a query only has a few minimal solutions: not only this makes the answers easier to understand and more informative, but *it also makes it easier to compute them*. Finally, we believe our approach has some advantages over Chan's since it is not restricted to CTL^v (or more generally to valid queries) and it extends nicely to situations where a query only has a small number of minimal solutions.

Plan of the paper. We recall the formal definitions of temporal queries, their solutions, etc. in Section 2. Then we study the structure of the set of solutions in Section 3 and prove a few key results that provide the basis for the polynomial-time algorithms we exhibit in Section 4. Hardness results are given in Section 5.

2 Temporal logic queries

We quickly recall the syntax and semantics of CTL, LTL, and CTL*, three of the main temporal logics used in model checking (we refer the reader to [Eme90,CGP99,BBF$^+$01] for more background). We assume the reader is familiar with the main complexity classes (see e.g. [Pap94]).

2.1 Syntax of temporal logics

Assume $\mathcal{P} = \{P_1, P_2, \ldots\}$ is a countable set of *atomic propositions*, CTL* formulae are given by the following syntax:

$$\varphi, \psi ::= \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{E}\varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \psi \mid P_1 \mid P_2 \mid \ \ldots$$

where E is the *existential path quantifier*, X is the *next-time* temporal modality, and U is the *until* temporal modality. Standard abbreviations are $\bot$ (for $P_1 \wedge \neg P_1$), $\top$ (for $\neg\bot$), $\mathsf{A}\varphi$ (for $\neg\mathsf{E}\neg\varphi$), $\mathsf{F}\varphi$ (for $\top\mathsf{U}\varphi$), $\mathsf{G}\varphi$ (for $\neg\mathsf{F}\neg\varphi$), as well as $\varphi \vee \psi$, $\varphi \Rightarrow \psi$, $\varphi \Leftrightarrow \psi$, ...

LTL is the fragment of CTL* where the path quantifiers E and A are forbidden.

CTL is the fragment of CTL* where every modality U or X must appear immediately under the scope of a path quantifier E or A. CTL formulae are *state formulae*, that is, whether $\pi \models \varphi$ only depends on the current state $\pi(0)$. When φ is a state formula, we often write $q \models \varphi$ and say that φ holds in state q of S.

Boolean combinations of atomic propositions (i.e. no temporal combinator is allowed) are a special case of vacuously temporal formulae, called *propositional formulae*, and ranged over by $f, g, \ldots$ Assuming the set of *atomic propositions* is $\mathcal{P} = \{P_1, P_2, \ldots\}$, the propositional formulae are given by the abstract syntax

$$f, g ::= f \wedge g \mid \neg f \mid P_1 \mid P_2 \mid \ \ldots$$

2.2 Semantics

A *Kripke structure* (shortly, a KS) is a tuple $S = \langle Q, q_{\text{init}}, \rightarrow, l\rangle$ where $Q = \{q, q', \ldots\}$ is a finite set of *states*, $q_{\text{init}} \in \mathcal{Q}$ is the *initial state*, $\rightarrow \subseteq Q \times Q$ is a total *transition relation* between states, and $l : Q \mapsto 2^{\mathcal{P}}$ labels the states with (finitely many) propositions. In the following we assume a given KS S.

A run π of S is an infinite sequence $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \cdots$ of states linked by transitions (i.e. $(q_i, q_{i+1}) \in \rightarrow$ for all i). For π of the form $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \cdots$, we write $\pi(i)$ for the i-th state q_i, and π^i for the i-th suffix $q_i \rightarrow q_{i+1} \rightarrow q_{i+2} \rightarrow \cdots$ of π. Observe that π^i is a run. We write $\Pi(q)$ for the set of runs that start from q: since we assume $\rightarrow$ is total, $\Pi(q)$ is never empty.

CTL* formulae describe properties of runs in a KS. We write $\pi \models \varphi$ when π satisfies φ in S. The definition is by induction on the structure of the formula:

$$\begin{array}{ll}
\pi \models P_i & \overset{\text{def}}{\Leftrightarrow} P_i \in l(\pi(0)), \\
\pi \models \neg\varphi & \overset{\text{def}}{\Leftrightarrow} \pi \not\models \varphi, \\
\pi \models \varphi \wedge \psi & \overset{\text{def}}{\Leftrightarrow} \pi \models \varphi \text{ and } \pi \models \psi, \\
\pi \models \mathsf{E}\varphi & \overset{\text{def}}{\Leftrightarrow} \text{there is a } \pi' \in \Pi(\pi(0)) \text{ s.t. } \pi' \models \varphi, \\
\pi \models \mathsf{X}\varphi & \overset{\text{def}}{\Leftrightarrow} \pi^1 \models \varphi, \\
\pi \models \varphi \mathsf{U} \psi & \overset{\text{def}}{\Leftrightarrow} \text{there is some } i \text{ s.t. } \pi^i \models \psi \text{ and } \pi^j \models \varphi \text{ for all } 0 \leq j < i.
\end{array}$$

We write $S \models \varphi$ when $\pi \models \varphi$ for all $\pi \in \Pi(q_{\text{init}})$ and say that the KS S satisfies φ.

Semantical equivalences between temporal formulae are denoted $\varphi \equiv \psi$, while entailments are written $\varphi \supset \psi$.

2.3 Temporal queries

Let TL be some temporal logic (e.g. CTL, LTL, CTL* or some other fragment of CTL*). A TL *query* γ is a TL formula in which the special placeholder symbol ? may appear in place where a propositional formula is allowed. Note that ? may appear at most once in γ. We say a query is *positive* (resp. *negative*) if ? appears under an even (resp. odd) number of negations. E.g. (Q) above is negative since "$? \Rightarrow$ `grant`" really is an abbreviation for $\neg ? \vee$ `grant`.

If f is a propositional formula, we write $\gamma(f)$ for the TL-formula obtained by replacing ? by f in γ. A *solution* of a query γ in a given KS S is a propositional formula f such that $S \models \gamma(f)$.

Lemmas 2.1 and 2.2 below state fundamental properties of the sets of solutions. Chan stated and proved them for CTL queries [Cha00] but they hold more generally.

Lemma 2.1 (Monotonicity). *Let γ be a CTL* query and f and g two propositional formulae:*
— if γ is positive then $f \supset g$ entails $\gamma(f) \supset \gamma(g)$,
— if γ is negative then $f \supset g$ entails $\gamma(g) \supset \gamma(f)$.

There is a duality principle linking positive and negative queries: let γ be a negative query. Then $\neg\gamma$ is a positive query and it is possible to compute the solutions of γ from the solutions of $\neg\gamma$: these are all f that are not solutions of $\neg\gamma$. This duality justifies the policy we now adopt: *from now on, we consider positive queries only.*

Lemma 2.2. *Let S be a Kripke structure and γ a CTL* query, then*
— $S \models \gamma(\bot)$ iff $S \models \gamma(f)$ for all propositional formulae f,
— $S \models \gamma(\top)$ iff $S \models \gamma(f)$ for some propositional formula f.

Lemma 2.2 shows that deciding whether a query admits a solution in a given KS can be reduced to a model checking question: does $S \models \gamma(\top)$?

2.4 Minimal solutions

A conclusion we draw from Lemmas 2.1 and 2.2 is that not all solutions to a temporal query problem are equally informative. The most informative solutions are the minimal ones:

Definition 2.3. *A* minimal *solution to a query γ in some KS S is a solution f that is not entailed by any other solution.*

Since the set of solutions to γ in S is closed w.r.t. entailment (also, *upward-closed*), the minimal solutions characterize the set of solutions.

The problem we are concerned with is, given a Kripke structure S and a temporal query γ, compute the set of minimal solutions to γ in S.

2.5 Relevant solutions

There exists an important generalization of temporal query answering. Here one considers a given subset $\mathcal{RP} \subseteq \mathcal{P}$ of so-called *relevant* atomic propositions and only looks for *relevant solutions*, i.e. solutions that only use propositions from $\mathcal{RP}$. Note that, by Lemma 2.2, a query has relevant solutions iff it has solutions.

A *minimal relevant solution* is minimal among relevant solutions only. This is not the same as being minimal and relevant: a query γ may have a unique minimal solution and several minimal relevant solutions, or, dually, several minimal solutions and a unique minimal relevant solution.

Being able to look for relevant solutions greatly improves the scope of temporal query checking. Going back to our example in the introduction, it is very likely that a minimal solution to (Q) involves a lot of specific details about the system at hand. We will probably only get the informative solution $\texttt{request}\wedge\texttt{free}$ if we restrict to a set of a few important propositions.

In the rest of this paper, we only look at the basic query answering problem. As the reader will see, all the results and algorithms we propose generalize directly to relevant solutions: it suffices to restrict the set of valuations one considers to valuations of $\mathcal{RP}$. While the generalization is crucial in practical applications, it is trivial and uninteresting in the more abstract analysis we develop.

3 The lattice of solutions

Assume we are given a fixed KS $S = \langle Q, q_{\text{init}}, \to, l\rangle$ where the states are labeled with propositions from a finite set $\mathcal{P} = \{P_1, \ldots, P_m\}$ of m atomic propositions. Assuming finiteness of $\mathcal{P}$ is not a limitation since anyway only the propositions appearing in S are useful.

A *valuation* is a mapping $v : \mathcal{P} \mapsto \{\bot, \top\}$ assigning a truth value to each atomic proposition. We write $\mathcal{V}$ for the set of valuations on $\mathcal{P}$. Observe that each state q of S can be seen as carrying a valuation denoted v_q and given by $v_q(P_i) = \top \overset{\text{def}}{\Leftrightarrow} P_i \in l(q)$.

In the standard way, a propositional formula f denotes a set of valuations $[\![f]\!] \subseteq \mathcal{V}$, the valuations that satisfy f. We write $|f|_\#$ for the size of $[\![f]\!]$, i.e. the number of valuations that satisfy f.

3.1 The lattice of propositional formulae

It is well know that the Boolean lattice $\langle 2^{\mathcal{V}}, \subseteq \rangle$ of subsets of $\mathcal{V}$ is isomorphic to the lattice of propositional formulae ordered by entailment (when we do not distinguish between equivalent propositional formulae). We write $\mathcal{L}_m$ for this lattice when $\mathcal{P}$ contains m atomic propositions.

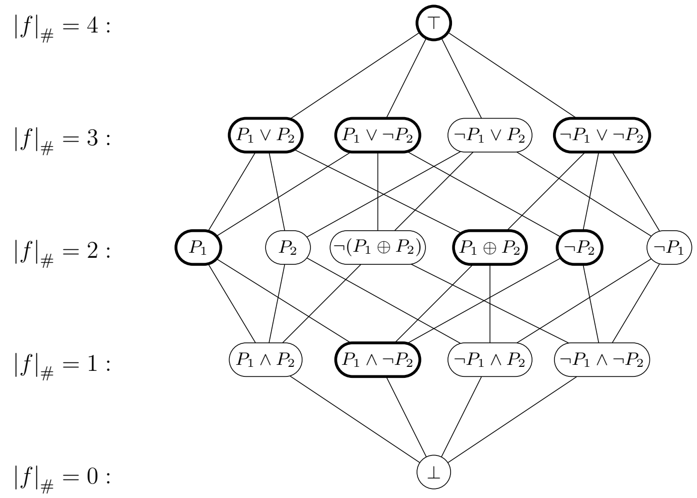

Fig. 1. $\mathcal{L}_2$, the lattice of propositional formulae for $\mathcal{P} = \{P_1, P_2\}$

Fig. 1 displays $\mathcal{L}_2$, the lattice of propositional formulae for $\mathcal{P} = \{P_1, P_2\}$. We use the exclusive-or operator, denoted $\oplus$, to shorten the notations of some formulae. In Fig. 1, the bold nodes delineate the sub-lattice of all propositional formulae entailed by $P_1 \wedge \neg P_2$.

There are 16 non-equivalent formulae in $\mathcal{L}_2$. The huge size of $\mathcal{L}_m$ can be explained with a few numbers: m atomic propositions allow 2^m valuations, so that there are 2^{2^m} non-equivalent propositional formulae.

$\mathcal{L}_m$ can be sliced in $2^m + 1$ levels in the natural way, where a level collects all formulae f with same $|f|_\#$. The central level is the largest, containing

$$\Gamma_m \overset{\text{def}}{=} \binom{2^m}{2^{m-1}}$$

distinct formulae. Therefore a temporal query cannot have more than Γ_m minimal solutions (where m is the number of relevant propositions).

3.2 Bounding the number of minimal solutions

We start by showing that the Γ_m upper bound for the number of minimal solutions is not very tight. Note that Γ_m is $2^{2^{\Omega(m)}}$.

In practice, rather than considering the number m of propositions, a better parameter is the number of different valuations that appear in the states of the Kripke structure. This is explained by the following lemma, where our notation "$f \vee v$" treats valuation v as a propositional formula with the obvious meaning:

Lemma 3.1. *Let f be a propositional formula and v a valuation. If v does not label any state of S, then $S \models \gamma(f \vee v)$ iff $S \models \gamma(f)$.*

Proof. Obviously, for any state $q \in S$, $q \models f \vee v$ iff $q \models f$. This equivalence carries over to the whole of γ by structural induction. $\square$

Proposition 3.2. *Let γ be a CTL* query, and S a KS with n nodes. Then γ has at most $\binom{n}{\lfloor n/2 \rfloor}$ minimal solutions, i.e. less than 2^n. Furthermore, $|f|_\# \leq n$ for any minimal solution f.*

Proof. By Lemma 3.1, a minimal solution is a disjunction of the form $\bigvee_{q \in Q'} v_q$ for some subset $Q' \subseteq Q$ of states of S. Thus a minimal solution can account for at most n valuations. Also, there are 2^n subsets of Q, but if we want a large number of distinct Q' subsets s.t. none contains another one, then at most $\binom{n}{\lfloor n/2 \rfloor}$ subsets can be picked. $\square$

Now there does exist simple situations where a query has an exponential number of minimal solutions.

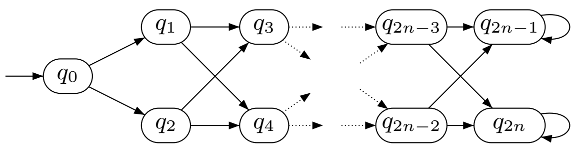

Fig. 2. S_n, a KS with 2^n minimal solutions to the query $\mathsf{EG}\,?$

Consider the KS S_n from Fig. 2: it has $2n+1$ states. If these $2n+1$ states are labeled with different valuations, then we have:

Proposition 3.3. *In S_n, the query $\gamma = \mathsf{EG}\,?$ has 2^n minimal solutions.*

Proof (Idea). Write $v_0, v_1, \ldots, v_{2n}$ for the valuations labeling $q_0, q_1, \ldots, q_{2n}$ The minimal solutions are all f of the form $v_0 \vee \bigvee_{i=1}^{n} v_{k_i}$ where $k_i \in \{2i-1, 2i\}$. $\square$

This example partly explains why it is difficult to compute the set of minimal solutions to an arbitrary query on an arbitrary KS: simply listing these minimal solutions takes exponential time. We show below that simply counting the minimal solutions is intractable (see Theorem 5.1).

4 Polynomial-time algorithms for temporal queries

In this section we exhibit two temporal-query problems that can be reduced efficiently to corresponding model checking problems.

Formally, these problems can be solved by a polynomial-time algorithm that uses model checking (for the underlying temporal logic) as an oracle. Using an oracle abstracts away from the specific details of the model checking algorithms for different temporal logics (and their different complexity).

Informally, we say that these problems can be solved "in P^{MC}". In practice, this means that they can be solved by simple extensions of standard model checking algorithms with little extra cost. In particular, *for CTL queries, these problems can be solved in polynomial-time.* More generally, this further holds of any temporal logic for which model checking is polynomial-time, like the alternation-free mu-calculus [CS92], or some other extensions of CTL like [KG96,LST00,LMS02,PBD+02]. Obviously, since model checking is a special case of temporal query solving, we cannot expect to do much better and provide efficient query solving for temporal logics where model checking is intractable.

Remark 4.1. We do not provide a fine-grained evaluation of our algorithm and say they are in polynomial-time when it is sometimes obvious that, e.g., only a linear number of oracle calls is used.

4.1 Deciding minimality

Minimality of solutions can be reduced to model checking. This requires that the candidate solutions be given in a manageable form:

Theorem 4.2. *The problem of deciding, given some KS $S = \langle Q, q_{\text{init}}, \rightarrow, l\rangle$, query γ and propositional formula f, whether f is* a minimal solution *to γ in S, is in* P^{MC} *when f is given in disjunctive normal form, or as an OBDD* [2]*.*

Proof. One uses the following algorithm:
(1) First check that $S \models \gamma(f)$, otherwise f is not a solution.
(2) Then check whether $|f|_{\#} > |Q|$. If so, then f is not a minimal solution (by Prop. 3.2).
(3) Otherwise, enumerate $[\![f]\!]$ as some $\{v_1, \ldots, v_k\}$. f is minimal iff $S \not\models \gamma(f \wedge \neg v_i)$ for $i = 1, \ldots, k$.
Therefore minimality can be decided with at most $1+|Q|$ invocations to a model checking algorithm. □

[2] I.e. *Ordered Binary Decision Diagrams* [Bry92]. They now are a standard way of efficiently storing and handling boolean formulae in model checking tools.

Here, the assumption that f is in disjunctive normal form, or an OBDD, permits efficient enumeration of $[\![f]\!]$ and decision of whether $|f|_\# > |Q|$ as we now explain.

Computing $|f|_\#$ when f is an OBDD uses simple dynamic programming techniques (see, e.g., [Bry92, § 5.4]). When f is a disjunctive normal form, computing $|f|_\#$ is a bit more difficult. However, we just need to check whether $|f|_\# > |Q|$, which can be done by enumerating $[\![f]\!]$ assuming we stop as soon as we see that $|f|_\#$ is too large.

Remark 4.3. If f can be an arbitrary propositional formula, then minimality of f is NP-hard since satisfiability can easily be reduced to minimality.

4.2 Computing unique minimal solutions

Uniqueness of the minimal solution can be reduced to model checking. Furthermore, when it is unique, the minimal solution can be computed efficiently.

Theorem 4.4. *The problem of deciding, given some KS $S = \langle Q, q_{\mathrm{init}}, \to, l\rangle$ and query γ, whether γ admits a unique minimal solution in S (and computing that solution) is in* P^{MC}.

Proof. One uses the following algorithm:
(1) Let $\mathcal{V}_Q$ be the set $\{v_q \mid q \in Q\}$ of valuations labeling a state of S. Write f_Q for $\bigvee_{v\in\mathcal{V}_Q} v$.
(2) Let $\mathcal{V}'$ be the set of all $v \in \mathcal{V}_Q$ s.t. $S \models \gamma(f_Q \wedge \neg v)$. Write $f_\wedge$ for $\bigvee_{v\in\mathcal{V}_Q\setminus\mathcal{V}'} v$.
(3) If $S \models \gamma(f_\wedge)$ then $f_\wedge$ is the unique minimal solution. Otherwise, there is no solution, or the minimal solution is not unique.

The correctness of this algorithm is easily proved: by construction, $f_\wedge$ is the infimum of all solutions and can only be a solution itself if it is the unique minimal solution.

Therefore uniqueness of the minimal solution can be decided (and that solution be computed) with at most $1 + |Q|$ invocations to a model checking algorithm. □

Remark 4.5. Theorem 4.4 is not in contradiction with Theorem 1 of [Cha00], where the validity of a CTL query is shown to be EXPTIME-complete. Indeed, we ask here the question of the existence and uniqueness of a minimal solution for one query in the *given* model, and not in all models as in [Cha00].

4.3 Minimal solutions without uniqueness

The ideas behind Theorem 4.4 can be adapted to situations where there is not a unique minimal solution.

Assume we are given a KS $S = \langle Q, q_{\mathrm{init}}, \to, l\rangle$, a temporal query γ, and k (distinct) minimal solutions $f_1, \ldots, f_k$. Then it is possible to tell whether

$f_1, \ldots, f_k$ form the complete set of minimal solutions with $O(|Q|^k + |Q|)$ invocations to the underlying model checking algorithm. If the answer is negative, it is furthermore possible to compute an additional f_{k+1} minimal solution.

The algorithm proceeds as follows: assume each minimal solution f_i is given under the form $f_i = \bigvee_{j=1}^{n_i} v_{i,j}$ of a disjunction of n_i single valuations [3]. Assume f is a solution to γ that is not implied by any f_i. Then $f_i \not\supset f$ for every $i = 1, \ldots, k$, i.e. there exists a valuation v_{i,r_i} s.t. $f \supset \neg v_{i,r_i}$, and $\neg v_{1,r_1} \wedge \cdots \wedge \neg v_{k,r_k}$ is a solution. Finally the k minimal solutions $f_1, \ldots, f_k$ do not form a complete set iff there is a choice function $(r_i)_{i=1,\ldots,k}$ with $1 \leq r_i \leq n_i$ s.t. $f_Q \wedge \neg v_{1,r_1} \wedge \cdots \wedge \neg v_{k,r_k}$ is a solution. Observe that there are at most $n_1 \times \cdots \times n_k$, which is $O(|Q|^k)$, candidate solutions.

Once we have a candidate solution f (with $f \supset f_Q$) we can compute a minimal solution f' that entails f with a linear number of invocation to the underlying model checking algorithm. One lists $[\![f]\!]$ as $\{v_1, \ldots, v_n\}$, sets $f' := f$ and repeats the following for $i = 1$ to n: if $S \models \gamma[f' \wedge \neg v_i]$ then $f' := f' \wedge \neg v_i$. When the loop is finished, the resulting f' is a minimal solution. This algorithm is non-deterministic and what f' we finally obtain depends on what enumeration of $[\![f]\!]$ was started with.

In summary, our methods allow computing a first minimal solution with a linear number of model checking calls, a second one with a quadratic number, a third one with a cubic number, etc.

Every additional minimal solution costs more, and it seems that some form of increasing cost cannot be avoided, as we show in the next section.

5 Counting the minimal solutions

In section 4.3 we saw that, for any fixed k, one can decide with a polynomial number of model checking calls, whether there are at most k minimal solutions to a query in some KS (and compute these minimal solutions). Here the degree of the polynomial grows with k so that when k is not fixed (is an input) the reduction is not polynomial-time any more.

This seems inescapable: counting the number of minimal solutions is hard. For a formal proof, we introduce the following two problems: one is a decision problem (yes-or-no output) while the other is a counting problem (numeric output).

MANY_SOLS:
Input: a KS S, a query γ, an integer k *in unary.*
Answer: yes iff there are at least k minimal solutions to γ in S.

COUNT_SOLS:
Input: a KS S, a query γ.
Answer: the number of minimal solutions to γ in S.

[3] It is easy to obtain these valuations from S if f_i is given in some other form.

Theorem 5.1 (Hardness of temporal query solving.). *When considering CTL queries:*
1. **Many_Sols** *is NP-complete.*
2. **Count_Sols** *is #P-complete.*
Furthermore, hardness already appears with the fixed query $\mathsf{EG}\,?$.

Proof. That **Many_Sols** is in NP, and **Count_Sols** is in #P, is easy to see since a minimal solution can be presented succinctly (it is associated with a subset of the set of states), and minimality is in P for CTL queries (Theorem 4.2).

Hardness of **Many_Sols** for NP is proved by reducing from SAT [4]. Assume $\mathcal{I}$ is a SAT instance with n Boolean variables in the form of a CNF with m clauses. With $\mathcal{I}$ we associate a KS $S_{\mathcal{I}}$.

We illustrate the construction on an example: with the following instance

$$\mathcal{I}: \quad \underbrace{x_2 \vee x_3}_{C_1} \;\wedge\; \underbrace{x_1 \vee x_2 \vee \overline{x_3}}_{C_2} \;\wedge\; \underbrace{\overline{x_2} \vee \overline{x_3}}_{C_3} \tag{1}$$

we associate the KS depicted in Fig. 3.

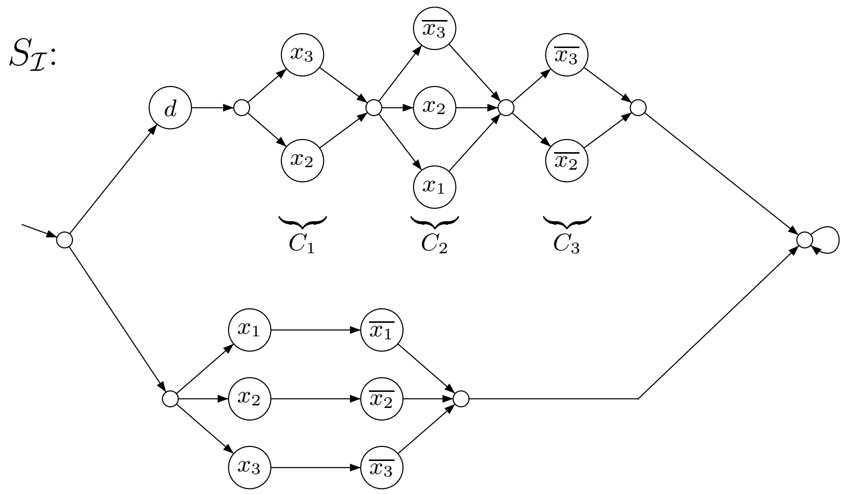

Fig. 3. The KS $S_{\mathcal{I}}$ associated with the SAT instance $\mathcal{I}$ from (1)

$S_{\mathcal{I}}$, has two kind of paths: the *bottom paths*, where a variable and its negation are visited, and the *top paths*, where one literal from each clause is picked.

Nodes in $S_{\mathcal{I}}$ are labeled by symbols (or are blank) but these labels stand for valuations: two nodes carry the same valuation iff they are labeled the same in our picture (as in the proof of Prop. 3.2, the exact values of these valuations are

[4] This elegant proof was indicated by Stéphane Demri.

not relevant). We consider the CTL query $\gamma = \mathsf{EG}\,?$: a solution f to γ in $S_{\mathcal{I}}$ must entail all the valuations appearing along some path, and the minimal solutions have the form $\bigvee_{q \in \pi} v_q$ for π a path. Such a solution, written f_π, is minimal iff f_π is not entailed by some other $f_{\pi'}$.

Clearly, since top paths visit d (dummy), all f_π for π a bottom path are minimal. We claim that there exists other minimal solutions if, and only if, $\mathcal{I}$ is satisfiable. Indeed, if a top path π is such that it is not entailed by any $f_{\pi'}$ for a bottom path π', then π never picks a literal and its negation. Hence π picks a valuation, and that valuation satisfies $\mathcal{I}$. Reciprocally, if $\mathcal{I}$ is satisfiable, the satisfying valuation gives us (a top path that provides) a solution not entailed by the n minimal "bottom paths" solutions.

Finally, γ has at least $n+1$ solutions in $S_{\mathcal{I}}$ iff $\mathcal{I}$ is satisfiable.

Hardness of **Count_Sols** for #P is proved by reducing from #SAT. We illustrate the construction on our earlier example: with (1) we associate the KS depicted in Fig. 4.

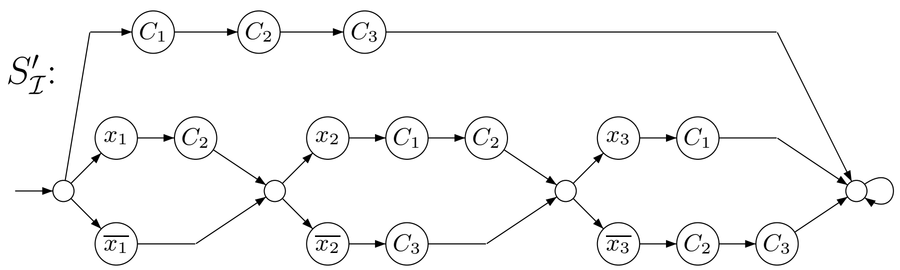

Fig. 4. The KS $S_{\mathcal{I}}$ associated with the SAT instance $\mathcal{I}$ from (1)

In $S'_{\mathcal{I}}$, the top path (denoted by π_{top}) lists all clauses in $\mathcal{I}$ and the other paths provide a way for enumerating all assignments for $\{x_1, \ldots, x_n\}$ by visiting, for every Boolean variable, the positive literal x_i, or the negative $\overline{x_i}$. When a path visits a literal, it immediately visits the clauses that are satisfied by this literal: e.g. C_2 and C_3 follow $\overline{x_3}$ in the bottom right corner of $S'_{\mathcal{I}}$ because $\overline{x_3}$ appears in C_2 and in C_3.

Again, the labels in $S'_{\mathcal{I}}$ stand for valuations and we consider the query $\gamma = \mathsf{EG}\,?$. Clearly, if two paths π and π' correspond to different assignments, then they differ on some literal nodes, so that f_π and $f_{\pi'}$ do not entail one another. However, if π corresponds to an assignment that satisfies $\mathcal{I}$, then f_π is entailed by $f_{\pi_{\text{top}}}$.

Finally, the number of minimal solutions to γ in $S'_{\mathcal{I}}$ is $2^n + 1 - k$ where k is the number of satisfying assignments for $\mathcal{I}$. This provides the required weakly parsimonious reduction [5]. $\square$

Remark 5.2. There is no contradiction between Theorem 5.1, saying it is hard to count the number of minimal solutions, and Theorem 4.4, saying it is easy to tell if there is a unique one. The situation is similar to #SAT which is #P-complete while it is easy to tell whether a CNF over n Boolean variables has 2^n satisfying assignments (i.e. is valid).

6 Conclusions

We studied the problem of computing the set of minimal solutions to arbitrary temporal queries over arbitrary Kripke structures. We showed this problem is intractable in general.

It turns out the problem is easier when one only asks for a few minimal solutions. *A fortiori*, this applies in situations where there is a unique minimal solution (or only a small number of them). Computing a single minimal solution can be done with a linear number of calls to a model checking procedure, i.e. in polynomial-time for CTL queries. Then a second minimal solution can be obtained with a quadratic number of calls, then a third solution with a cubic number, etc. This has some advantages over Chan's approach where only a very restricted set of so-called valid queries can be handled.

We did not implement our algorithms. The next logical step for this study is to implement and evaluate them, using examples drawn from practical case studies. Such an evaluation will tell whether, in practice, temporal queries often have a small number of minimal solutions, whether only computing a partial sample of the full set of minimal solutions can be useful for understanding and validating the model at hand. In principle, and because it has less limitations, our algorithm must be at least as useful as Chan's algorithm, for which he provided convincing application examples [Cha00].

Acknowledgments

We thank the anonymous referees who prompted us to develop the results that are now gathered in section 4.3, and Stéphane Demri who indicated the elegant reduction depicted in Fig. 3.

References

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools.* Springer, 2001.

[5] Obviously, parsimonious reductions cannot be used since telling whether γ has at least one minimal solution can be done in polynomial-time.

[BG01] G. Bruns and P. Godefroid. Temporal logic query checking (extended abstract). In *Proc. 16th IEEE Symp. Logic in Computer Science (LICS'2001), Boston, MA, USA, June 2001*, pages 409–417. IEEE Comp. Soc. Press, 2001.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.

[Cha00] W. Chan. Temporal-logic queries. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 450–463. Springer, 2000.

[CS92] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In *Proc. 3rd Int. Workshop Computer Aided Verification (CAV'91), Aalborg, Denmark, July 1991*, volume 575 of *Lecture Notes in Computer Science*, pages 48–58. Springer, 1992.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, chapter 16, pages 995–1072. Elsevier Science, 1990.

[KG96] O. Kupferman and O. Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, 1996.

[LMS02] F. Laroussinie, N. Markey, and Ph. Schnoebelen. On model checking durational Kripke structures (extended abstract). In *Proc. 5th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2002), Grenoble, France, Apr. 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2002.

[LST00] F. Laroussinie, Ph. Schnoebelen, and M. Turuani. On the expressivity and complexity of quantitative branching-time temporal logics. In *Proc. 4th Latin American Symposium on Theoretical Informatics (LATIN'2000), Punta del Este, Uruguay, Apr. 2000*, volume 1776 of *Lecture Notes in Computer Science*, pages 437–446. Springer, 2000.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[PBD+02] A. C. Patthak, I. Bhattacharya, A. Dasgupta, P. Dasgupta, and P. P. Chakrabarti. Quantified Computation Tree Logic. *Information Processing Letters*, 82(3):123–129, 2002.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA, Oct.-Nov. 1977*, pages 46–57, 1977.