Model Checking Quantified Computation Tree Logic

Arend Rensink

Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE, The Netherlands
rensink@cs.utwente.nl

Abstract. Propositional temporal logic is not suitable for expressing properties on the evolution of dynamically allocated entities over time. In particular, it is not possible to trace such entities through computation steps, since this requires the ability to freely mix quantification and temporal operators.

In this paper we study *Quantified Computation Tree Logic* (QCTL), which extends the well-known propositional computation tree logic, PCTL, with first and (monadic) second order quantification. The semantics of QCTL is expressed on *algebra automata*, which are automata enriched with abstract algebras at each state, and with reallocations at each transition that express an injective renaming of the algebra elements from one state to the next. The reallocations enable minimization of the automata modulo bisimilarity, essentially through symmetry reduction. Our main result is to show that each combination of a QCTL formula and a finite algebra automaton can be transformed to an equivalent PCTL formula over an ordinary Kripke structure, while maintaining the symmetry reduction. The transformation is structure-preserving on the formulae. This gives rise to a method to lift any model checking technique for PCTL to QCTL.

1 Introduction

Ever since its conception in the 80's, model checking has been based on modal extensions of propositional logic. That is to say, the properties that can be formulated and checked have as their smallest building blocks a finite set of atomic propositions, each of which is satisfied by a subset of the states of the model (Kripke structure, transition system, automaton) being checked. This means that, for the purpose of model checking, the information in each of the states is abstracted to the subset of propositions satisfied there.

Since the propositions themselves can be defined in any manner whatsoever (as long as only finitely many of them are considered at the same time) this setup can be used also in settings where the states have rich associated domains — for instance, the state snapshots of a software system. A good example of this principle arises in software models with a fixed set of variables over a finite set of values, such as can be written in, for instance, Spin's input language Promela [15]: there the states are essentially valuations of those variables, and typical propositions are (in)equations over the variables. As a more sophisticated example, one can define propositions that are actually closed first-order formulae interpreted over the states; this allows the expression of existential and universal properties even in a setting where the size of the state domains (such as the number of variables or entities) is not fixed. As an example, one may think of a property like “the buffer can always eventually become empty” interpreted in a model where values are added to and removed from cells of a buffer of variable (bounded) size; here the proposition “the buffer is empty” actually corresponds to the first-order property “no cell in the buffer contains a value”.

(Note that a property such as this one is independent of the size of the buffer; this is why quantification is essential to be able to formulate it. When the model is fixed, so that the maximum size of the state domains is known, any closed first-order state formula can be *expanded* to an equivalent quantifier-free one by flattening the quantifiers to a finite disjunction or conjunction ranging over all existing values.)

This setup can also be explained in terms of a two-layered logic: at the bottom we have a language to express those properties of individual states that are considered interesting for verification purposes; on top of that we define a modal logic, in which the properties of the lower level are treated as propositions. There are, however, system-level properties that are relevant to the correctness of a system and yet cannot be expressed in this two-layered setup. Typically, these are properties where the *behaviour of individual entities over time* is at issue. An example that will be used throughout this paper is “values are removed from the buffer in the same order they are inserted” (or “the buffer has FIFO-behaviour”). Here it is important not only that a buffer cell contains *some* object, but also that *the same* object was (or was not) contained by some buffer cell in the next or previous state. In order to express this, we need to track the identity of the object over multiple states, which can only be done through quantification *outside* the modal operators; hence, the two-layer hierarchy no longer suffices.

From this observation, it follows that there is interest in logics in which quantification and modalities can be freely mixed — a point we have argued before in [11, 10, 19], and has been made independently by Yahav et al. in [21]. In contrast to the latter, we pursue a model checking approach. In our work cited above this was limited to multisets resp. singly-linked lists, which however were unbounded in size; in the current paper we study arbitrary algebraic structures (like [21]), albeit (in our case) for finite state, or in other words, bounded models only. As modal logic we take Quantified Computation Tree Logic (QCTL), which adds first and (monadic) second order quantification to Propositional Computation Tree Logic (PCTL, see [6]). The contribution of this paper is to show that:

1. Using second order quantification, QCTL formulae can not only be used to track entities over time, but also to express (de)allocation schedules, such as the fact that entities are deleted in their order of creation.
2. Any combination of a property in QCTL together with a finite model to be checked (in which the size of the state domains is variable) can be transformed to a combination of an expanded, quantifier-free formula and an expanded model, such that the model checking question has the same answer in both combinations.
3. This can be made to work on models that are minimized up to bisimilarity (using reallocations between states) without losing the reduction due to that minimization.

Regarding the latter point: for our models (which we call *algebra automata* since the state domains are algebras of some fixed signature) we use an idea from history-dependent automata (Montanari and Pistore [16, 17]): each transition carries a *reallocation* function from the entities in its source to thoses in its target state. This allows states with symmetrical domains to be merged, and thus can help to keep models small: [17] shows that history-dependent automata can be minimized with respect to bisimilarity. Depending on the amount of symmetry in the system, minimization can result in a logarithmically smaller model in terms of the number of states and transitions, while keeping the algebra sizes constant — at the price of the reallocations themselves.

In the terminology of quantified modal logic (see, e.g., Fitting [12, 13], Basin et al. [2]), our models have variable domains and non-rigid designators, and our transformation has a strong analogy to *Skolemization* — the introduction of a new constant (non-rigidly designating) for every quantified logical variable. The idea is essentially that of *case splitting* for existential quantifiers, modified to take (de)allocation and reallocations into account and to retain the state space reduction due to minimization.

The transformation theorem of this paper implies that existing tools and techniques for PCTL model checking (see, e.g., [7]) can be used directly for QCTL, once the property and model are both given. The complexity of the transformation of the automaton depends on the maximal nesting depth of quantifiers in the formula, d, and the maximum size of the algebras in the individual states, a: the transformation results in a worst-case blow-up exponential in d and a, or just linear in a if the formula contains first-order quantification only. On the formula the transformation results in blow-up linear in the number of temporal operators and quantifiers. Note that this complexity is no better than was to be expected by a simple combinatorial argument based on the boundedness of the model, but is still interesting in the light of the aforementioned potential for symmetry reduction.

Sect. 2 defines and discusses the logic, Sect. 3 defines its semantics and Sect. 4 defines the transformation and proves the main result. Sect. 5 discusses some improvements, including the addition of (Büchi) fairness. We draw conclusions and discuss related work in Sect. 6. For space reasons, most proofs had to be omitted; however, see http://www.cs.utwente.nl/~rensink/papers/concur2006-full.pdf.

2 The Logic

The structures that we will model and reason about in this paper are built on a set of *names* Name. The same names are used for functions and predicates in the model and for logical variables. Names will be interpreted by strict partial functions $\mathsf{Ent}_\perp^\alpha \rightharpoonup \mathsf{Ent}_\perp^\tau$ for some set of entities Ent (where $\perp \notin \mathsf{Ent}$ stands for undefinedness and $\mathsf{Ent}_\perp = \mathsf{Ent} \cup \{\perp\}$), with $\alpha \in \mathsf{Nat}$ (the *arity*) and $\tau \in \{0, 1\}$ (the *type*). In fact for the sake of conciseness we assume that every name $n \in \mathsf{Name}$ has a fixed arity αn and type τn, which are respected by the interpretation. We use $\mathsf{Name}^{i,j}$ with $i \in \mathsf{Nat}$ and $j = 0, 1$ for the subset of names with arity i and type j. The entities, which from the point of the formalism are uninterpreted, can in practice be made to stand for arbitrary data and reference values; for instance, in software models they can stand for stack frames or heap objects.

The intuition is that if $\tau n = 0$ then n denotes a partial function to a singleton set (since $\mathsf{Ent}^0 = \{\varepsilon\}$ consists of the empty sequence only), which in turn corresponds to the characteristic function for a predicate with arity αn (or a monadic second-order variable if used in the logic); the predicate is taken to hold in a given state if and only its value is defined. If, moreover, $\alpha n = 0$ then n corresponds to a proposition. On the other hand, if $\tau n = 1$ then n is a (partially defined) operator with arity αn; If, moreover, $\alpha n = 0$ then n corresponds to an ordinary constant (in the model) or first-order variable (in the logic).

As meta-variables over Name, we typically use p to range over predicates (including propositions), f for functions, and c for constants; for the general case we use x, y.

We now introduce the logic studied in this paper, QCTL. The syntax is given by the following grammar, which defines *terms* (meta-variables t, u) and *formulae* (meta-variables ϕ, ψ):

$$\begin{aligned} t &::= f(\boldsymbol{u}) \\ \phi &::= t \mid t \equiv u \mid \exists x\, \phi \mid \phi \wedge \psi \mid \neg\phi \mid \mathsf{EX}\, \phi \mid \mathsf{E}(\phi \;\mathsf{U}\; \psi) \mid \mathsf{A}(\phi \;\mathsf{U}\; \psi) \end{aligned}$$

In our examples, we assume that negation binds strongest, and quantification as well as EX bind weakest.

– A term $t = f(\boldsymbol{u})$ stands for the application of $f \in \mathsf{Name}$ to a vector of sub-terms $\boldsymbol{u}$, with $|\boldsymbol{u}| = \alpha f$. If $\alpha f = 0$ then $\boldsymbol{u} = \varepsilon$, in which case we usually omit the brackets and write only f.
  The notion of type can be extended from names to terms in a natural way, by specifying $\tau t = \tau f$ if $t = f(\boldsymbol{u})$; t is *well-typed* if for all $u_i \in \boldsymbol{u}$, $\tau u_i = 1$ and u_i is again well-typed. Since functions are generally partial, terms may evaluate to $\perp$.
– A formula $\phi = t$ expresses that (the interpretation of) t is defined; i.e., t does *not* evaluate to $\perp$. (Note that, due to the fact that our interpretations can be partial, we are not in classical logic.) In particular, if $\tau t = 0$ (meaning that $t = p(\boldsymbol{u})$ for some predicate p) then this is the usual interpretation of predicates. On the other hand, $\neg t$ with $\tau t = 1$ denotes that the entity denoted by t no longer exists — presumably because it has been deallocatied in a transition leading up to the current state.
– $\phi = t {\equiv} u$ expresses equality of the interpretations of t and u, where it is assumed that $\tau t = \tau u = 1$. Equality will be interpreted strictly, meaning that $t = u$ will be false if either t or u (or both) evaluate to $\perp$. Non-strict equality is expressed by $(t \vee u) \Rightarrow t \equiv u$.
– $\phi = \exists x\, \psi$ is existential quantification over x in ψ; it will be deemed valid if an appropriate (defined) value can be found for x such that ψ then holds. We limit this to first-order and monadic second-order quantification ($x \in \mathsf{Name}^{0,1}$ or $x \in \mathsf{Name}^{1,0}$, respectively).
  In the interpretation of the logic, to be defined below, a sub-formula x in the context of a quantified formula $\exists x\, \psi$ (i.e., the usage of a logical first-order variable as a formula) stands for the fact that the entity denoted by x is still "alive", i.e., has not been de-allocated. For second-order variables p, the sub-formula $p(x)$ in the context of $\exists p\, \psi$ denotes that x is among the (surviving) entities in the set p.

The other clauses correspond to the usual connectives from computation tree logic. Briefly:

– $\mathsf{EX}\, \phi$ expresses that ϕ holds in some state directly reachable from the current state;
– $\mathsf{E}(\phi \;\mathsf{U}\; \psi)$ expresses that there is a run of the system starting in the current state, in which ψ holds at some point, and ϕ holds at all earlier points;
– $\mathsf{A}(\phi \;\mathsf{U}\; \psi)$ expresses that for all runs of the system starting in the current state, ψ holds at some point, and ϕ holds in all earlier points.

We will also freely use the derived formulae $\forall x\, \phi$, $\phi \vee \psi$, $\mathsf{AX}\, \phi$ (the dual of $\mathsf{EX}\, \phi$), $\mathsf{AF}\, \phi$ and $\mathsf{EF}\, \phi$ (defined as $\mathsf{A}(\mathbf{tt}\ \mathsf{U}\ \phi)$ and $\mathsf{E}(\mathbf{tt}\ \mathsf{U}\ \phi)$, respectively) and their duals $\mathsf{EG}\, \phi$ and $\mathsf{AF}\, \phi$. We also define the *free names* of a formula ϕ, denoted $\mathit{fn}(\phi)$, as usual; we use $\mathit{fn}^{i,j}(\phi)$ $(= \mathit{fn}(\phi) \cap \mathsf{Name}^{i,j})$ to denote the subset of names with arity i and type j.

An important special class of formulae are the *propositional* ones. These are formulae for which the first- and second-order features of the logic are essentially unused: the only names are propositions (i.e., with $\alpha = \tau = 0$) and no quantification is used.

Definition 1 (propositional formulae). *A formula ϕ is called* propositional *if it is quantifier-free and $\mathit{fn}(\phi) \subseteq \mathsf{Name}^{0,0}$.*

Example 1. Assume $\mathit{List}, \mathit{Cell}, \mathit{Data}, S \in \mathsf{Name}^{1,0}$, $x, y \in \mathsf{Name}^{0,1}$, $\mathit{next}, \mathit{val} \in \mathsf{Name}^{1,1}$ and $\mathit{connect} \in \mathsf{Name}^{2,0}$. The following are example properties of QCTL:

1. $\mathsf{AG}\forall x\, (\mathit{Cell}(x) \Leftrightarrow \mathit{Data}(\mathit{val}(x)))$, expressing a type invariant, viz. that in all reachable states, val is defined only for, and for all, Cell-type entities, and always yields a Data-type entity.
2. $\forall x\, (\mathit{Data}(x) \Rightarrow \mathsf{AF}\, \neg x)$, expressing that all currently existing Data-type entities are eventually de-allocated.
3. $\forall S\, \mathsf{EF}\, \exists x\, (\mathit{Data}(x) \wedge \neg S(x))$, expressing that in all system behaviours, some new Data-type entity is eventually allocated. (Note that S is a second-order variable; $\neg S(x)$ expresses that x is not in the set S, meaning that the entity denoted by x did not exist in the state where S was bound.)
4. $\mathsf{AG}\, \forall x, y(\mathit{List}(x) \wedge \mathit{Cell}(y) \wedge \mathit{connect}(x, y) \Rightarrow (\mathsf{AG}\, y) \vee \mathsf{A}(\mathit{connect}(x, y)\, \mathsf{U}\, \neg y))$, expressing that cells can become disconnected from a List-type entity only when they are de-allocated.
5. $\mathsf{AG}\, \forall S\, (\forall x\, \mathit{Data}(x) \Leftrightarrow S(x)) \Rightarrow \mathsf{AG}\, \forall x,y\, S(x) \wedge \mathit{Data}(y) \Rightarrow S(y) \vee \mathsf{A}(y\ \mathsf{U}\ \neg x)$, expressing that Data-type entities are allocated and de-allocated in first-in-first-out order. To understand this, note that the sub-formula $\forall x\, \mathit{Data}(x) \Leftrightarrow S(x)$ specifies that the logical second-order variable S is equivalent (in the state where S is bound) to the predicate Data. Furthermore, $\mathsf{A}(y\, \mathsf{U}\, \neg x)$ expresses that y lives at least as long as x. Thus, $\forall x,y\, S(x) \wedge \mathit{Data}(y) \Rightarrow S(y) \vee \mathsf{A}(y\, \mathsf{U}\, \neg x)$ expresses that, for all Data-type entities x and y, if x existed in the (past) state where S was bound but y did not — meaning that y was created after x — then y will survive x.

We introduce some further syntactic sugar. In the following let $x \in \mathsf{Name}^{0,1}$, $S \in \mathsf{Name}^{1,0}$ and $T \in \mathsf{Name}^{1,\tau}$ for some τ, and let t be a term with $\tau t = 1$.

– $\exists x{:}T\, \phi$ stands for $\exists x\, T(x) \wedge \phi$;
– $\forall x{:}T\, \phi$ stands for $\forall x\, T(x) \Rightarrow \phi$;
– $\ulcorner x{\equiv}t\, \phi$ stands for $\exists x\, x{\equiv}t \wedge \phi$.
– $\ulcorner S{\equiv}T\, \phi$ stands for $\exists S\, (\forall x\, S(x) \Leftrightarrow T(x)) \wedge \phi$.
– $\ulcorner S\, \phi$ stands for $\exists S\, (\forall x\, S(x)) \wedge \phi$.

Thus, in $\exists x{:}T\, \phi$, the first-order variable x is bound to some entity of "type" T (i.e., on which T is defined), whereas in $\ulcorner x{\equiv}t\, \phi$ it is bound precisely to the current value of the term t (which has to be defined). Likewise, in $\ulcorner S{\equiv}T\, \phi$, the second-order variable S is bound to the set of all values of type T; finally, $\ulcorner S\,\phi$ binds S to the set of *all* currently existing values. The last three properties can be read as "let ... equal ... in ϕ."

Using this syntactic sugar, for instance, the property in Ex. 1.5 above becomes

$$\mathsf{AG}\,\ulcorner S{\equiv}Data\,\mathsf{AG}\,\forall x{:}S, y{:}Data(S(y) \vee \mathsf{A}(y\;\mathsf{U}\;\neg x)) \tag{1}$$

Valuations. To interpret the logic we need to express what the names stand for; in other words, we need the concept of a *valuation*. Valuations are defined in terms of *entities*: if $E \subseteq \mathsf{Ent}$ is some set of entities and $N \subseteq \mathsf{Name}$ a set of names, then a valuation of N over E is a function $V\colon N \to E^* \rightharpoonup E^{0,1}$ such that for all $x \in N$, $V(x)\colon E^{\alpha x} \rightharpoonup E^{\tau x}$; in words, V assigns to every name a partial function of the appropriate arity and type. The set of valuations of N over E is denoted $\mathsf{Val}[N,E]$. Valuations are strictly extended to terms, in the natural way:

$$V(f(\boldsymbol{u})) = \begin{cases} V(f)(V(u_1)\cdots V(u_{\alpha f})) & \text{if } V(u_i) \neq \bot \text{ for all } 1 \leq i \leq \alpha f \\ \bot & \text{otherwise.} \end{cases}$$

We call N the *domain* of V, denoted $dom(V)$. Note that we may actually have $V(x) = \bot$ for $x \in N^{0,1}$; in this case, the variable x is in the domain of V despite the fact that V assigns "undefined" to it.

Another way to understand the concept of a valuation $V \in \mathsf{Val}[N,E]$ is that it defines a partial N-*algebra* over the domain E (N being the signature of the algebra).

If $V \in \mathsf{Val}[N,E]$ and $W \in \mathsf{Val}[M,E]$, then $V\{W\}$ equals W wherever it is defined, and V otherwise. Furthermore, $V|_{-x}$ denotes V minus the value for x. Formally:

$$V\{W\} : y \mapsto \begin{cases} W(y) & \text{if } y \in dom(W) \\ V(y) & \text{otherwise.} \end{cases} \qquad V|_{-x} : y \mapsto V(y) \text{ if } x \neq y.$$

It follows that $dom(V\{W\}) = N \cup M$ and $dom(V|_{-x}) = N \setminus \{x\}$.

3 Algebra Automata

To express the semantics of QCTL we define an automata model that includes a fixed set of *model names*, as well as a separate domain of values at each state, with a corresponding valuation of the model names. In fact, the domain and valuation together constitute an *algebra* for the model names (considered as a signature). Furthermore, we use an idea from *History-Dependent Automata* proposed by Montanari and Pistore [17], namely to allow *reallocations* of values between states.

Definition 2. *Let* Ent *be a set of entities. An algebra automaton* $\mathcal{A}$ *over* Ent *is a tuple* $\langle N, S, D, A, \to, I\rangle$ *where*

- $N \subseteq \mathsf{Name}$ *is a finite set of names;*
- S *is a set of states;*
- $D\colon S \to \mathbf{2}^{\mathsf{Ent}}$ *associates with every* $s \in S$ *a domain* $D(s)$ *of values "existing" in* s*;*
- $A\colon S \to \mathsf{Val}[N,\mathsf{Ent}]$ *associates with every* $s \in S$ *an algebra* $A(s) \in \mathsf{Val}[N, D(s)]$*;*

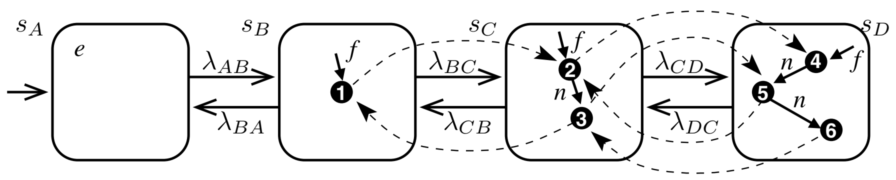

Fig. 1. First-in-first-out list

- $\rightarrow \subseteq S \times (\mathsf{Ent} \rightharpoonup \mathsf{Ent}) \times S$ *is an indexed binary relation between states, where the indices are partial injective functions that map the domain of the source state to the domain of the target state; thus,* $s \rightarrow_\lambda s'$ *implies* $\lambda\colon D(s) \rightharpoonup D(s')$. *Every state has at least one outgoing transition;*
- $I \subseteq S$ *is a set of initial states.*

$\mathcal{A}$ *is called finite if* S *is finite and* $D(s)$ *is finite for all* $s \in S$.

The index λ in a transition $s \rightarrow_\lambda s'$ stands for a *reallocation* (or renaming) of entities. That is, an entity $e \in D(s)$ that does not have an image in λ is *deallocated* (dies) during the transition; otherwise, the entity remains in existence but is known in s' as $\lambda(e)$. Entities $e' \in D(s')$ that are not in the range of λ are *allocated* (created, born). Note that λ is *not* required to preserve the algebraic structure of the state: indeed the structure may change, e.g., references or values may be reassigned, as in the transitions λ_{BC} and λ_{DC} in the following example.

Example 2. Let $e \in \mathsf{Name}^{0,0}, f \in \mathsf{Name}^{0,1}$ and $n \in \mathsf{Name}^{1,1}$ stand for *empty*, the *first* and *next* elements in a list; then **??** shows an algebra automaton with $N = \{e, f, n\}$ which models the behaviour of a list (of maximum length 3) of which the elements are allocated and deallocated in a first-in-first-out manner. The rounded rectangles are states, containing valuations $V \in \mathsf{Val}[N, \mathsf{Nat}]$ (so the numbered nodes represent the entities): proposition e holds in the state where it is inscribed, whereas the constant $V(f)$ and the partial function $V(n)$ are given as arrows. The reallocations λ are shown as dashed arrows, implicitly associated with the transitions in the corresponding direction.

From [17] we recall the important property that history-dependent automata can be minimized with respect to bisimilarity, defined appropriately to abstract from the entities while maintaining the algebraic structure up to isomorphism. That is, a bisimulation between algebra automata $\mathcal{A}_1$ and $\mathcal{A}_2$ is a family of symmetric relations $\{R_f \subseteq S_1 \times S_2\}_{f:\mathsf{Ent}\rightarrow\mathsf{Ent}}$, such that $(s_1, s_2) \in R_f$ implies

- f is an isomorphism from $A_1(s_1)$ to $A_2(s_2)$;
- $s_1 \rightarrow_\lambda s_1'$ implies $s_2 \rightarrow_\mu s_2'$ for some s_2' such that $(s_1', s_2') \in R_g$ with $g \circ \lambda = \mu \circ f$;
- $s_2 \rightarrow_\mu s_2'$ implies $s_1 \rightarrow_\lambda s_1'$ for some s_1' such that $(s_1', s_2') \in R_g$ with $g \circ \lambda = \mu \circ f$.

We call $\mathcal{A}_1$ and $\mathcal{A}_2$ bisimilar, denoted $\mathcal{A}_1 \sim \mathcal{A}_2$, if there exists such a bisimulation $\{R_f\}_f$, such that R_{id} is a total relation between I_1 and I_2.

Essentially, minimization w.r.t. $\sim$ comes down to *symmetry reduction*: all states with isomorphic algebras can be folded together, maintaining the connection with the entities in neighbouring states through the reallocations. In the automaton of Fig. 1, which is already minimized, this can be seen from the fact that the reallocations λ_{CD} and λ_{DC} between are not inverse to one another. Unfolding this example automaton so that all reallocations become (partial) identities results in a quadratic blowup (in the number of states). In the worst case the blowup is exponential in the size of the algebra — or in other words, minimization w.r.t. $\sim$ can result in a logarithmically smaller automaton.

The well-known model of Kripke structures appears as an important special case, where all the names are propositions (i.e., in $\mathsf{Name}^{0,0}$) and there are no entities. We call such automata *propositional*.

Definition 3 (propositional automata). *An algebra automaton is called* propositional *if $N \subseteq \mathsf{Name}^{0,0}$ and $D(s) = \emptyset$ for all $s \in S$.*

A key fact used in this paper (see Th. 2 below) is that for the special case of propositional automata and propositional formulae, efficient solutions to the model checking problem are well known from the literature (cf. [7]).

3.1 Semantics of QCTL

We now express the semantics of QCTL in terms of algebra automata. For this purpose we first need the notion of a *run* of such an automaton. Note that we have applied a common trick by enforcing every state to have at least one outgoing transition; this makes the presentation technically easier.

Definition 4 (paths runs). *Let $\mathcal{A}$ be an algebra automaton. A* path *through $\mathcal{A}$ is a finite or infinite alternating sequence $\sigma = s_0\,\lambda_1\,s_1\,\lambda_2\,s_2\cdots$, ending on a state if the sequence is finite, such that for all λ_i in the sequence, $s_{i-1} \rightarrow_{\lambda_i} s_i$ is a transition in $\mathcal{A}$. The path is called a* run *if it is infinite.*

The set of runs of $\mathcal{A}$ is denoted $runs(\mathcal{A})$. If $\sigma = s_0\,\lambda_1\,s_1\,\lambda_2\,\cdots$ is a run then

- For all $i \geq 0$, $\sigma|_i^s$ denotes the state at position i in the run, i.e., s_i;
- For all $i > 0$, $\sigma|_i^\lambda$ denotes the reallocation at position i in the run, i.e., λ_i;
- For all $i \geq 0$, $\sigma|_{\leq i}^\lambda$ denotes the reallocation *up to* position i in the run, i.e., $\lambda_i \circ \lambda_{i-1} \circ \cdots \circ \lambda_1$. This is interpreted to yield $id_{D(s_0)}$ if $i = 0$.

The semantics of QCTL is expressed by a relation $\mathcal{A}, s, V \models \phi$ where ϕ is a QCTL-formula, $\mathcal{A}$ is an algebra automaton, $s \in S$ is a state of $\mathcal{A}$ and $V \in \mathsf{Val}[M, D(s)]$ is a valuation, with $fn(\phi) \supseteq M \cup N_{\mathcal{A}}$. We write $\mathcal{A}, V \models \phi$ if $\mathcal{A}, s, V \models \phi$ for all $s \in I$. Moreover, we may omit V if $dom(V) = \emptyset$, and $\mathcal{A}$ if it is clear from the context. The modelling relation is defined by induction on the structure of ϕ, as follows:

$$
\begin{array}{lll}
\mathcal{A}, s, V \models t & :\Leftrightarrow & A(s)\{V\}(t) \neq \bot \\
\mathcal{A}, s, V \models t \equiv u & :\Leftrightarrow & A(s)\{V\}(t) = A(s)\{V\}(u)\ \ (\neq \bot) \\
\mathcal{A}, s, V \models \exists x\, \phi & :\Leftrightarrow & \mathcal{A}, s, V\{W\} \models \phi \text{ for some } W \in \mathsf{Val}[\{x\}, D(s)] \\
\mathcal{A}, s, V \models \mathsf{EX}\, \phi & :\Leftrightarrow & \mathcal{A}, s', \lambda \circ V \models \phi \text{ for some } s \rightarrow_\lambda s' \\
\mathcal{A}, s, V \models \mathsf{E}(\phi \,\mathsf{U}\, \psi) & :\Leftrightarrow & \text{there is a } \sigma \in runs(\mathcal{A}) \text{ with } \sigma|_0^s = s \text{ such that} \\
 & & \mathcal{A}, \sigma|_i^s, \sigma|_{\leq i}^\lambda \circ V \models \psi \text{ for some } i \geq 0 \text{ and} \\
 & & \mathcal{A}, \sigma|_j^s, \sigma|_{\leq j}^\lambda \circ V \models \phi \text{ for all } 0 \leq j < i; \\
\mathcal{A}, s, V \models \mathsf{A}(\phi \,\mathsf{U}\, \psi) & :\Leftrightarrow & \text{for all } \sigma \in runs(\mathcal{A}) \text{ with } \sigma|_0^s = s\text{:} \\
 & & \mathcal{A}, \sigma|_i^s, \sigma|_{\leq i}^\lambda \circ V \models \psi \text{ for some } i \geq 0 \text{ and} \\
 & & \mathcal{A}, \sigma|_j^s, \sigma|_{\leq j}^\lambda \circ V \models \phi \text{ for all } 0 \leq j < i.
\end{array}
$$

The following property (the proof of which is straightforward and omitted here) is important in the light of the discussion above regarding minimization up to bisimilarity:

Theorem 1. *If $\mathcal{A} \sim \mathcal{B}$, then $\mathcal{A}, V \models \phi$ iff $\mathcal{B}, V \models \phi$ for all* QCTL*-formulae ϕ.*

We can now formulate the "key fact" about model checking propositional formulae, referred to above:

Theorem 2 (See [7]). *Given a finite algebra automaton $\mathcal{A}$ and a propositional formula ϕ, $\mathcal{A} \models \phi$ can be decided in time linear in the size of ϕ and the size of $\mathcal{A}$.*

Example 3. Without proof, we assert that the automaton of Fig. 1 satisfies the formulae

$$\mathsf{AG}\,\forall x\; n(x) \Rightarrow \Gamma y{\equiv}n(x)\,\mathsf{A}(y{\equiv}n(x)\;\mathsf{U}\;\neg x) \tag{2}$$

$$\mathsf{AG}\,\Gamma S\;\mathsf{AF}\,\exists x\,\neg S(x) \tag{3}$$

$$\mathsf{AG}\,\forall x\;\mathsf{AF}\,\neg x \tag{4}$$

$$\mathsf{AG}\,\Gamma S\;\mathsf{AG}\,\forall x{:}S\,\forall y\; S(y) \vee \mathsf{A}(y\;\mathsf{U}\;\neg x) \tag{5}$$

$$\mathsf{AG}\,\mathsf{EF}\,e\;\;. \tag{6}$$

Property (2) expresses that the n-pointers in the automaton are *immutable* in the sense that whenever the term $n(x)$ is defined for a given entity x, it will go on designating the same value until x itself is deallocated. Property (4) is a simplified form of Ex. 1.2 expressing that every entity is always eventually deallocated. Likewise, (3) is a simplified form of Ex. 1.3 expressing that a fresh entity is always eventually allocated. Property (5) is a simplified version of (1) expressing that entities are created and destroyed in a first-in-first-out schedule.

Finally, (6) expresses that the state where the list is empty always remains reachable. This is in fact a propositional formula and so can be model checked with existing methods (see Th. 2). Note that algebra automata include no fairness criterion, and so it is not true that the empty list is always eventually reached (i.e., the property $\mathsf{AG}\,\mathsf{AF}\,e$ is not satisfied). See, however, Sect. 5 where we discuss the extension of the model with just such a fairness criterion.

The following theorem states an intuitively straightforward property, heavily used in practice, namely that quantifier-free formulae can be treated as if they were propositional, by defining propositions for all basic formulae t and $t \equiv u$ and abstracting the models accordingly.

Theorem 3. *Let $\mathcal{A}$ be an algebra automaton and let ϕ be a formula with $fn(\phi) \subseteq N$. If ϕ is quantifier-free, then there is a propositional formula ϕ' and a propositional automaton $\mathcal{A}'$, with $size(\phi') = size(\phi)$, $S_{\mathcal{A}'} = S_{\mathcal{A}}$ and $\rightarrow_{\mathcal{A}'} = \{(s, \emptyset, s') \mid s \rightarrow_\lambda s'\}$, such that $\mathcal{A} \models \phi$ if and only if $\mathcal{A}' \models \phi'$.*

Proof. We sketch the proof. The idea is to introduce a propositional name $n_\beta \in \mathsf{Name}^{0,0}$ for every *base sub-formula* ψ in ϕ, where a base formula is of the form $\psi = t$ or $\psi = t{\equiv}u$. ϕ' equals ϕ with all base formulae ψ replaced by the corresponding names

n_ψ; $\mathcal{A}'$ is constructed from $\mathcal{A}$ by re-using the states, setting $D'(s) = \emptyset$ for all $s \in S$, re-using the transitions while stripping away the reallocations λ, and defining

$$A'(s)\colon n_\psi \mapsto \begin{cases} \varepsilon & \text{if } \psi = t \text{ and } A(s)(t) \neq \bot \\ \varepsilon & \text{if } \psi = t{\equiv}u \text{ and } A(s)(t) = A(s)(u) \neq \bot \\ \bot & \text{otherwise.} \end{cases}$$

The proof obligation is implied by the following property, which can be proved by induction on the structure of ϕ: For all $s \in S$, $\mathcal{A}, s \models \phi$ if and only if $\mathcal{A}', s \models \phi'$. □

4 Skolemization

The essential idea in model checking a QCTL formula ϕ over a given algebra automation $\mathcal{A}$ is to turn the bound logical variables in ϕ into new (non-rigidly designating) model constants — a principle known as *Skolemization* — and to simulate the binding of a logical variable during the evaluation of the formula by a *random assignment* to the corresponding model constant. We can then equivalently model check a transformed formula ϕ^-, where all quantifications are changed into next-step modalities, over the extended automaton. Since ϕ^- is quantifier-free, due to Th. 3 we can apply existing theory to solve the transformed model checking problem.

In fact it is not enough to add the variables to the model and simulate their assignment: in addition we have to be able to distinguish between the transitions of the original automaton and the new "assignment transitions". This will be done by using *assignment flags*, which are proposition names $\alpha_x \in \mathsf{Name}^{0,0}$ for all variables x to be Skolemized, as well as one distinguished flag $\bar{\alpha}$, which stands for *no assignment* and behaves as the negated disjunction of the α_x. In the remainder we assume that the assignment flags are globally given and distinct from all other names in the automaton and the formula to be checked. Furthermore, for a given set of variables X we use $\alpha_X = \{\alpha_x \mid x \in X\} \cup \{\bar{\alpha}\}$ to denote the set of all assignment flags. We also use β, γ to range over α_X.

Definition 5. *Let $\mathcal{A}$ be an algebra automaton, and $X \subseteq \mathsf{Name}$. The X-Skolemization of $\mathcal{A}$, denoted $\mathcal{A}^{+X}$, is given by $\langle N', S', \rightarrow', D', A', I'\rangle$ with*

$$\begin{aligned}
N' &= N \cup X \cup \alpha_X \\
S' &= \{(s, W, \bar{\alpha}) \mid W \in \mathsf{Val}[X, D(s)]\} \cup \\
&\quad\ \{(s, W, \alpha_x) \mid W \in \mathsf{Val}[X, D(s)], x \in X, W(x) \neq \bot\} \\
\rightarrow' &= \{((s, W, \bar{\alpha}), \lambda, (s', \lambda \circ W, \bar{\alpha})) \mid s \rightarrow_\lambda s'\} \cup \\
&\quad\ \{((s, W, \bar{\alpha}), id_{D(s)}, (s, W', \alpha_x)) \mid (s, W, \bar{\alpha}) \in S', W'|_{-x} = W|_{-x}\} \cup \\
&\quad\ \{((s, W, \alpha_x), id_{D(s)}, (s, W, \bar{\alpha})) \mid (s, W, \bar{\alpha}) \in S'\} \\
D' &= \{((s, W, \beta), D(s)) \mid (s, W, \beta) \in S'\} \\
A' &= \{((s, W, \beta), A(s)\{W\}\{\varepsilon/\beta\}) \mid (s, W, \beta) \in S'\} \\
I' &= \{(s, W, \bar{\alpha}) \mid s \in I, dom(W) = \emptyset\}\ .
\end{aligned}$$

The principle of the construction is to allow, in every state, to "guess" a random value and assign it to one of the new variables in the set X. Each state of the extended automaton is a triple consisting of the corresponding state of the original automaton, a combined assignment W, and an assignment flag β indicating which (if any) of the Skolemized variables has been assigned a new value since the previous state. That is, either $\beta = \bar{\alpha}$ if the valuation was unchanged, or $\beta = \alpha_x$ for some $x \in X$ if a new value for x was guessed in the transitions leading up to the state. There are three types of transitions: those reflected from the original automaton, those reflecting random assignment steps, and those leading back from an assignment state to a "normal" state. In the first type, $\beta = \bar{\alpha}$ in source and target state and the guessed valuation W is kept constant (modulo the reallocation); in the second type, the state is unchanged, $\beta = \alpha_x$ (for some $x \in X$) in the target state and W may change (only) at x; in the third type, $\beta = \alpha_x$ in the source state, and the state and guessed valuation are kept constant.

The corresponding transformation of the formulae is defined as follows:

$$\begin{aligned}
t^- &= t \\
(t \equiv u)^- &= t \equiv u \\
(\neg\phi)^- &= \neg\phi^- \\
(\phi \vee \psi)^- &= \phi^- \vee \psi^- \\
(\exists x\, \phi)^- &= \mathsf{EX}(\alpha_x \wedge \mathsf{EX}\, \phi^-) \\
(\mathsf{EX}\, \phi)^- &= \mathsf{EX}(\bar{\alpha} \wedge \phi^-) \\
\mathsf{E}(\phi\, \mathsf{U}\, \psi)^- &= \mathsf{E}((\bar{\alpha} \wedge \phi^-)\, \mathsf{U}\, (\bar{\alpha} \wedge \psi^-)) \\
\mathsf{A}(\phi\, \mathsf{U}\, \psi)^- &= \mathsf{A}(\phi^-\, \mathsf{U}\, (\bar{\alpha} \Rightarrow \psi^-))
\end{aligned}$$

The intuition is that quantification is operationalised by a transition of the extended automaton, which guesses a value for the quantified variable — followed by another transition that returns to a "regular" state. The quantification operator itself is likewise turned into a pair of next-step operators. In order to distinguish "regular" from "assignment" next-steps, we test for the absence or presence of an assignment flag.

The following is the main theorem of this paper:

Theorem 4. *Let ϕ be an arbitrary formula; let X denote the set of names bound in ϕ. For any algebra automaton $\mathcal{A}$ with $fn(\phi) \subseteq N$, the following equivalence holds:*

$$\mathcal{A} \models \phi \qquad \textit{if and only if} \qquad \mathcal{A}^{+X} \models \phi^- \ .$$

Proof. The theorem follows from the following, stronger property, which holds for all $s \in S$ and $V \in \mathsf{Val}[Y, D(s)]$:

$$\mathcal{A}, s, V \models \phi \quad \text{if and only if} \quad \mathcal{A}^{+X}, (s, V, \bar{\alpha}) \models \phi^- \ .$$

This is proved by induction on the structure of ϕ. □

Example 4. Let ϕ denote property (4), and $\mathcal{A}$ the algebra automaton of Ex. 2, simplified to a list of maximum length 2. Fig. 2 shows $\mathcal{A}^{+\{x\}}$: the dotted arrows are the assignment transitions, and the λ's are indicated by pairs of entities, from the source resp. the target state. The skolemized formula is

$$\phi^- \quad = \quad \mathsf{AG}(\mathsf{AX}(\alpha_x \Rightarrow \mathsf{EX}\,\mathsf{AF}(\bar{\alpha} \Rightarrow \neg x)))$$

Clearly, checking ϕ^- over $\mathcal{A}^{+\{x\}}$ is a case of PCTL model checking. The states where $\bar{\alpha} \Rightarrow \neg x$ holds are shaded in the figure.

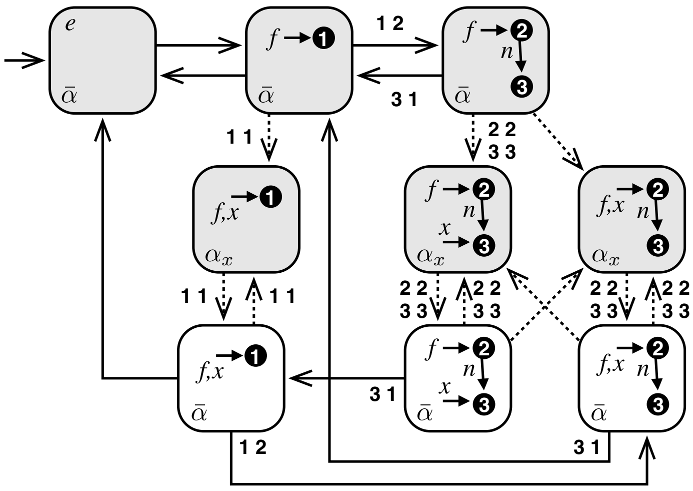

Fig. 2. Skolemization of part of the automaton of Fig. 1 w.r.t. $x \in \mathsf{Name}^{0,1}$

The size of $\mathcal{A}^{+X}$ can be computed as follows: for modelling the possible choices of a first-order variable we need $|D(s)|$ new states for each $s \in S_{\mathcal{A}}$; for a second-order variable this is $2^{|D(s)|}$. The number of "original" transitions between the new states grows with the same factor, and the number of "assignment" transitions is triple the number of new states. Thus Skolemizing a single variable blows up the automaton linearly (for first-order) resp. exponentially (for second-order) in the maximum domain size. This is repeated for every variable bound in ϕ, making the blow-up exponential in the number of variables. (Note that the domais themselves are not affected.)

Theorem 5 (complexity). *Let $\mathcal{A}$ be an algebra automaton with maximum algebra size a; let $X \subseteq \mathsf{Name}^{0,1} \cup \mathsf{Name}^{1,0}$ and let $d_1 = |X^{0,1}|$ and $d_2 = |X^{1,0}|$. If $\mathcal{B} = \mathcal{A}^{+X}$, then $|S_{\mathcal{B}}| \leq s \cdot |S_{\mathcal{A}}|$ and $|\rightarrow_{\mathcal{B}}| \leq s \cdot |\rightarrow_{\mathcal{A}}|$ with $s = O((d_1 + d_2) \cdot a^{d_1} \cdot 2^{a \cdot d_2})$.*

Skolemization of the formula also increases its size, but by a constant factor only.

Theorems 2, 3, 4 and 5 together give rise to the following worst-case time complexity:

Corollary 1. *A* QCTL *formula can be model checked over a finite algebra automaton in time linear in the number of states and transitions, exponential in the size of the formula and exponential in the maximum size of the state domains.*

5 Improvements

As defined above, skolemization only takes a set of (typed) names as input. By taking more information about the formula to be checked into account, the definition can be improved in several ways, resulting in a smaller automaton.

Collectively Bound Names. In Def. 5 the skolemized automaton receives assignment transitions everywhere. Yet they are used only to mimic quantification in the formula, say ϕ, that we want to model check. Through an analysis of ϕ, we can omit many of the assignment transitions or their target states that can clearly never be taken or reached, and thus achieve an appreciable reduction of the skolemized automaton.

A simple observation which can already cause a large reduction is that we only need to assign to sets of variables that occur *together* in some sub-formula. Define the *collectively bound names* $\mathcal{N}(\phi) \subseteq \mathbf{2}^{\mathsf{Name}}$ for arbitrary $\phi \in \mathsf{QCTL}$ as follows:

– $\mathcal{N}(\phi) = \{\emptyset\}$ whenever ϕ is quantifier-free;
– $\mathcal{N}(\neg\phi) = \mathcal{N}(\mathsf{EX}\,\phi) = \mathcal{N}(\phi)$;
– $\mathcal{N}(\phi \vee \psi) = \mathcal{N}(\mathsf{E}(\phi\,\mathsf{U}\,\psi)) = \mathcal{N}(\mathsf{A}(\phi\,\mathsf{U}\,\psi)) = \mathcal{N}(\phi) \cup \mathcal{N}(\psi)$;
– $\mathcal{N}(\exists x\,\phi) = \{Y \cup \{x\} \mid Y \in \mathcal{N}(\phi)\}$.

In model checking ϕ^-, all the states of $\mathcal{A}^{+X}$ that are actually encountered are of the form (s, V, β) with $\{x \in X \mid V(x) \neq \bot\} \subseteq M$ for some $M \in \mathcal{N}(\phi)$. It follows that we may omit all states that are *not* of this form, and still obtain the same answer to the model checking question for ϕ^-. This obviously affects the size of the resulting automaton, since the space over which the valuations V range is now possibly much smaller. In terms of Th. 5, the factor s is now of the order

$$\textstyle\sum_{M\in\mathcal{N}(\phi),d_1=|M^{0,1}|,d_2=|M^{1,0}|}\ (d_1+d_2)\cdot a^{d_1}\cdot 2^{a\cdot d_2}\ .$$

Quantification Order. If we take the above analysis of the formula one step further, it becomes clear that assignment transitions need only ever be taken in the order in which we encounter quantifiers in ϕ, when traversing the syntax tree of ϕ top-down. For instance, in (2) this order is x followed by y, whereas in (5) it is S–x–y. This means that we may cut out transitions that attempt to assign the variables in any different order. Since we may also cut out non-reachable parts of the automaton, this may cause a further reduction.

For instance, in Fig. 2 the transitions leading from the bottom $\bar{\alpha}$-states back to the α_x-states would be removed by this optimization (without, however, a reduction in the number of states).

Assignment. A further optimization, causing a generally unpredictable but potentially large improvement, is to define a special treatment of the binders introduced as syntactic sugar in Sect. 2: $\exists x{:}T\,\phi$ and its dual, but especially $\ulcorner x{\equiv}t\,\phi$, $\ulcorner S{\equiv}T\,\phi$ and $\ulcorner S\,\phi$. Namely, in these cases the possible values assigned to the logical variables are *not* arbitrary values but satisfy some very strict constraints; in fact, in the latter three cases they are bound *precisely* to uniquely defined values.

In terms of Def. 5, if x is bound by such a special syntactic form then the assignment states for x, i.e., the states (s, V, α_x) in $\mathcal{A}^{+X}$, should all satisfy the corresponding constraint on V; i.e., $V(T)(V(x)) \neq \bot$ for $\exists x{:}T\,\phi$, $V(x) = V(t)$ for $\ulcorner x{\equiv}t\,\phi$, etc. Thus, the number of resulting assignment states (for a given s) is no longer $|D(s)|$ or $2^{|D(s)|}$, but much smaller and some cases just 1! Unfortunately, we cannot conclude from this that the whole skolemized automaton will always be that much smaller. This kind of constrained assignment may ruin the symmetry that has originally allowed states to be collapsed (while keeping track of entities through reallocations), and hence may partially or wholly undo the symmetry reduction discussed in Sect. 3.

Non-temporal Quantification In Th. 3 we have recalled how quantifier-free formulae may be reduced to PCTL. As recounted in the introduction, the same principle still works for quantified formulae, as long as all sub-formulae $\exists x\, \psi$ have the property that ψ is without temporal operators. We may take advantage of this by defining yet another optimization, in which all temporal-operator-free sub-formulae are reduced to propositional names, assigned the appropriate value by an extended valuation for the states.

As an example, regard property (3) (Page 118). The sub-formula $\psi = \exists x\, \neg S(x)$ is free of temporal operators and hence a candidate for this optimization. The resulting skolemized property becomes

$$\mathsf{AG}(\bar{\alpha} \Rightarrow \mathsf{EX}(\alpha_S \wedge \mathsf{EX}\,\mathsf{AF}(\bar{\alpha} \Rightarrow n_\psi)))$$

and the skolemized automaton needs to contain assignments to S only.

Fairness. Skolemization can still be applied if the algebra automata are extended with a fairness condition. This involves constructing a corresponding fairness condition for the skolemized automata, where the correspondence should be such that there is the same relation between the fair runs of $\mathcal{A}$ and those of $\mathcal{A}^{+X}$ as there is between the runs as originally defined in Def. 4. We show the necessary construction without proof.

Assume an algebra automaton $\mathcal{A}$ in addition has a component $\mathcal{F} \subseteq \mathbf{2}^S$, and that $runs(\mathcal{A})$ is restricted to those sequences σ such that $\{i \mid \sigma|_i^s \in F\} = \infty$ for all $F \in \mathcal{F}$. Then $\mathcal{A}^{+X}$ should receive a corresponding component $\mathcal{F}'$ defined by

$$\mathcal{F}' = \{\, \{(s, V, \beta) \in S \mid s \in F\} \;\mid F \in \mathcal{F}\} \ .$$

In words, $\mathcal{F}'$ consists of those sets that project onto the fair sets of $\mathcal{F}$. Note that there are, in fact, many fair runs in $\mathcal{A}^{+X}$ that do not project to fair runs of $\mathcal{A}$, because they just cycle around through the assignment states; however, the stripped formulae themselves prevent these "spurious" runs from making a difference to their validity, just as in the case without fairness.

With this adaptation, the proof of Th. 4 goes through just as before. Since, according to [7], model checking PCTL is linear in the size of $\mathcal{F}$, and this size is not affected by the skolemization defined above, Corollary 1 can also be extended with a dependency on the fairness criterion that is linear in the size of $\mathcal{F}$.

6 Conclusions

We have presented an effective technique for checking QCTL, which combines monadic second-order quantification with the temporal operators of CTL, on finite models with arbitrary (bounded) algebraic structure on the states and reallocations on the transitions. The reallocations allow models to be minimized up to bisimilarity (appropriately defined), resulting in a best-case logarithmic reduction in the size of the automata.

It is interesting to note that the technique used in our proof extends to other temporal logics in a limited way only. In order for the encoding of $\exists x\, \phi$ as $\mathsf{EX}(\alpha_x \wedge \mathsf{EX}\, \phi')$ to be valid, it seems crucial that ϕ is a state formula: if it is interpreted in the context of a path then this context is lost in the encoding. This means that the technique is useless for LTL, whereas it can still be used in the fragment of PCTL* where quantification is just allowed on state formulae (of the form $\mathsf{E}\,\phi$ or, dually, $\mathsf{A}\,\phi$).

On the other hand, it should be possible to extend Skolemization to a setting where the temporal modalities are defined through fixpoints, as in the μ-calculus [9]. Here the fact that we can repeatedly assign to the same variable may turn out to be crucial.

The proof theory of quantified modal logic has been studied in depth in the context of philosophical logic. An overview can be found in [14]; some more references were given in the introduction. Results on automated theorem proving (which is a much harder problem than the one studied here, since it is not restricted to finite models) are presented in Castellini and Smaill [4, 5]. Some decidability results on words over infinite alphabets can be found in [18, 3]. Finally, in [1] Baldan et al. present a translation of a quantified temporal logic to a Petri net logic, and so obtain an automatic way to approximate its verification.

As we pointed out in the introduction, a source of more closely related work is Yahav et al. [21]. Their Evolution Temporal Logic, which is a quantified extension of Linear Temporal Logic, is motivated by the same considerations as QCTL, namely to express properties that track entities over time. Through abstraction they can conservatively verify unbounded models, though they do not include reallocations.

As far as we are aware, however, the model checking question was not studied before, at least not for models with arbitrary algebraic structure on the states and quantified temporal logic. For models with unbounded domains (and consequently an infinite number of states, in a suitable finite representation) but very limited structure, some results on model checking were presented in [11, 10]. The first of these shows decidability of model checking for *unstructured* domains, i.e., just sets of entities; the second gives a safe approximation in the case where the domains are singly-linked lists. Finally, model checking for bounded domains and propositional temporal logic has been addressed in many software model checking tools; for instance, Bandera (e.g., [8]).

As future work, we plan to implement the algorithm presented here in the GROOVE tool for graph transformation-based verification [20], thereby realizing one important step of the programme, set out in [19], for model checking graph grammars.

References

[1] P. Baldan, A. Corradini, B. König, and A. LLuch Lafuente. A temporal graph logic for abstractions of graph rewrite systems. Draft, 2005.

[2] D. A. Basin, S. Matthews, and L. Vigano. Labelled modal logics: Quantifiers. *Journal of Logic, Language and Information*, 7(3):237–263, 1998.

[3] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on words with data. Research Report 2005-004, LIAFA — Laboratoire d'Informatique Algorithmique: Fondements et Applications, 2005.

[4] C. Castellini and A. Smaill. A modular, tactic-based approach for first-order temporal theorem proving. In *International Conference on Temporal Logic (ICTL)*, 2000.

[5] C. Castellini and A. Smaill. Proof planning for first-order temporal logic. In Nieuwenhuis, ed., *20th International Conference on Automated Deduction (CADE)*, vol. 3632 of *LNCS*, pp. 235–249. Springer, 2005.

[6] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the IBM Workshop on Logics of Programs*, vol. 131 of *LNCS*, pp. 52–71. Springer, 1982.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *Symposium on Principles of Programming Languages (POPL)*, pp. 117–126. ACM Press, 1983.

[8] J. C. Corbett, M. B. Dwyer, J. Hatcliff, and Robby. Expressing checkable properties of dynamic systems: the bandera specification language. *International Journal on Software Tools for Technology*, 4(1):34–56, 2002.

[9] M. Dam. CTL$\star$ and ECTL$\star$ as fragments of the modal μ-calculus. *Theoretical Comput. Sci.*, 126(1):77–96, 1994.

[10] D. Distefano, J.-P. Katoen, and A. Rensink. Who is pointing when to whom? on the automated verification of linked list structures. In K. Lodaya and M. Mahajan, eds., *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, vol. 3328 of *LNCS*, pp. 250–262. Springer-Verlag, 2004.

[11] D. Distefano, A. Rensink, and J.-P. Katoen. Model checking birth and death. In Baeza-Yates, Montanari, and Santoro, eds., *Foundations of Information Technology in the Era of Network and Mobile Computing*, vol. 223 of *IFIP Conference Proceedings*, pp. 435–447. Kluwer Academic Publishers, 2002.

[12] M. Fitting. Bertrand Russell, Herbrand's theorem, and the assignment statement. In Calmet and Plaza, eds., *Artificial Intelligence and Symbolic Computation (AISC)*, vol. 1476 of *LNAI*, 1998.

[13] M. Fitting. On quantified modal logic. *Fundamenta Informaticae*, 39(1):5–121, 1999.

[14] J. W. Garson. Quantification in modal logic. In Guenthner and Gabbay, eds., *Handbook of Philosophical Logic, Vol. 3*, pp. 267–323. Kluwer, 2 edition, 2001.

[15] G. J. Holtzmann. *The Spin Model Checker: Primer and Reference Manual.* Addison-Wesley, 2003.

[16] U. Montanari and M. Pistore. History-dependent automata. Technical Report TR-11-98, Department of Computer Science, University of Pisa, 1998.

[17] U. Montanari and M. Pistore. History-dependent automata: An introduction. In Bernardo and Bogliolo, eds., *Formal Methods for Mobile Computing, 5th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM)*, vol. 3465 of *LNCS*, pp. 1–28. Springer, 2005.

[18] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In Sgall, Pultr, and Kolman, eds., *Mathematical Foundations of Computer Science (MFCS)*, vol. 2136 of *LNCS*, pp. 560–572. Springer, 2001.

[19] A. Rensink. Towards model checking graph grammars. In Leuschel, Gruner, and Presti, eds., *Workshop on Automated Verification of Critical Systems (AVoCS)*, Technical Report DSSE–TR–2003–2, pp. 150–160. University of Southampton, 2003.

[20] A. Rensink. The GROOVE simulator: A tool for state space generation. In Pfalz, Nagl, and Böhlen, eds., *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, vol. 3062 of *LNCS*, pp. 479–485. Springer-Verlag, 2004.

[21] E. Yahav, T. Reps, M. Sagiv, and R. Wilhelm. Verifying temporal heap properties specified via evolution logic. In Degano, ed., *Programming Languages and Systems: 12th European Symposium on Programming (ESOP)*, vol. 2618 of *LNCS*, pp. 204–222. Springer, 2003.