Verifying Quantitative Properties using Bound Functions

Arindam Chakrabarti[1], Krishnendu Chatterjee[1], Thomas A. Henzinger[1,4], Orna Kupferman[2], and Rupak Majumdar[3]

[1] UC Berkeley, USA
[2] Hebrew University, Israel
[3] UC Los Angeles, USA
[4] EPFL, Switzerland

Abstract. We define and study a quantitative generalization of the traditional boolean framework of model-based specification and verification. In our setting, propositions have integer values at states, and properties have integer values on traces. For example, the value of a quantitative proposition at a state may represent power consumed at the state, and the value of a quantitative property on a trace may represent energy used along the trace. The value of a quantitative property at a state, then, is the maximum (or minimum) value achievable over all possible traces from the state. In this framework, model checking can be used to compute, for example, the minimum battery capacity necessary for achieving a given objective, or the maximal achievable lifetime of a system with a given initial battery capacity. In the case of open systems, these problems require the solution of games with integer values.

Quantitative model checking and game solving is undecidable, except if bounds on the computation can be found. Indeed, many interesting quantitative properties, like minimal necessary battery capacity and maximal achievable lifetime, can be naturally specified by *quantitative-bound automata*, which are finite automata with integer registers whose analysis is constrained by a bound function f that maps each system K to an integer $f(K)$. Along with the linear-time, automaton-based view of quantitative verification, we present a corresponding branching-time view based on a quantitative-bound μ-calculus, and we study the relationship, expressive power, and complexity of both views.

1 Introduction

Traditional algorithmic methods for the verification of finite-state systems, with a set P of *boolean* propositions, translate a system into a transition graph in which each vertex corresponds to a state of the system and is labeled by the propositions that hold in the state. A property of the system is specified by a temporal-logic formula over P or by an automaton over the alphabet 2^P. When the system is closed (i.e., its behavior does not depend on the environment), verification is reduced to *model checking* [7]; for open systems, verification requires *game solving* [1]. While successful for verifying hardware designs [5] and communication protocols [12], this approach cannot adequately handle infinite-state systems that arise, for example, in general software verification. Much research has therefore focused on infinite-state extensions, such as models whose vertices carry a finite, but unbounded amount of information, e.g., a pushdown store, or integer-valued registers [14]. Much of the reasoning about such systems, however, has still focused on boolean specifications (such as "is the buffer size always bounded by 5?") rather than answering quantitative questions (e.g., "what is the maximal buffer size?"). Moreover, the main challenge in most infinite-state formalisms has been to obtain decidability for checking boolean properties, usually by limiting the expressive power of the models or properties.

In contrast, the solution of *quantitative* questions, such as system power requirements and system lifetime, has been considered on a property-by-property basis. Often the solution consists, however, of two basic steps: first, a suitable system of constraints is set up whose solution gives the intended quantitative answer (a "dynamic program"); and second, by considering the characteristics of the system (number of states or maximal initial battery power), a bound is provided on the number of iterations required to solve the dynamic program. We systematize this ad-hoc approach to answering quantitative questions about infinite-state systems in order to make it accessible to design engineers. For this purpose, we extend the traditional boolean verification framework to an *integer-based* framework, which due to its generality permits the modeling of a wide variety of quantitative aspects and properties of systems [6, 4].[1] In particular, we generalize traditional boolean specification formalisms such as automata to the integer-based framework, so that an engineer can express the desired quantitative properties in a natural way. These quantitative automata are then automatically translated into dynamic programs for model checking and game solving. Finally, from parametric bounds given by the engineer, such as bounds on the value of a quantity or on the number of automaton steps necessary for computing a property, we automatically derive iteration bounds on solving the corresponding dynamic program. In all the examples we study, such as maximal lifetime of a system with given initial battery capacity, our generic, systematic approach matches the best known previous, property-specific algorithms.

Specifically, the models we consider, *quantitative structures*, are graphs with finitely many vertices, but every vertex is labeled by a set of *quantitative propositions*, each taking an integer value. For example, the label at each vertex may represent the amount of power consumed when the vertex is visited, or it may represent a buffer size, a time delay, a resource requirement, a reward, a cost, etc. The properties we check are quantitative properties of infinite paths, each representing a run of the system. For instance, we may ask for the peak power

[1] It should be noted that we use the term *quantitative*, as in quantitative verification, quantitative property, or quantitative μ-calculus, simply as referring to "integer-based" rather than "boolean." This is not to be confused with some literature, where the term *quantitative* is used to refer to "probabilistic" systems, and real values are obtained as results of evaluating boolean specifications [2, 13, 16, 10].

consumption along a path, or for the lifetime of a battery along the path given a certain amount of initial battery power (i.e., the number of transitions along the path until the initial battery power is used up). Such properties can be specified by an extension of traditional automata. While a traditional automaton maps infinite paths of a graph with boolean propositions (i.e., infinite words over the alphabet 2^P) to "accept" or "reject", we define *quantitative* automata, which map each infinite path of a graph with quantitative propositions (i.e., infinite words over the alphabet $\mathbb{N}^P$) to an integer. For example, if the proposition $p \in P$ describes the amount of power consumed when the current input letter is read, then an automaton specifying battery lifetime, given initial power $a \in \mathbb{N}$, maps each word $o_1 o_2 o_3 \ldots$ to the maximal $k \geq 0$ for which $\sum_{i=1}^{k} o_i(p)$ is at most a. In model checking, boolean properties of infinite paths can be interpreted either in an existential or universal way, asking whether the property is true on some or all paths from a given state. In quantitative verification, we ask for the *maximal* or *minimal* value of a property over all paths from a state. For the battery lifetime property, this amounts to computing the maximal or minimal achievable lifetime (note that this corresponds to the battery lifetime in the cases that a scheduler resolves all nondeterminism in a friendly vs. an adversarial manner). In a game, where two players (system components) decide which path is taken, boolean properties are interpreted in an $\exists\forall$ fashion ("does player 1 have a strategy so that for all player 2 strategies the property is satisfied?"). Accordingly, we interpret quantitative properties in a *max min* fashion ("what is the maximal value of the property that player 1 can achieve no matter how player 2 plays?").

Since quantitative automata subsume counter machines, model checking and game solving are undecidable. However, unlike much previous work on infinite-state verification, we do not focus on defining decidable subclasses, but we note that in many examples that arise from verification applications, it is often easy and natural to give a *bound function.* This function specifies, for given system parameters (such as number of states, maximal constants, etc.), a threshold when it is safe to conclude that the value of a quantitative property tends to infinity. Accordingly, we specify a quantitative property as a *quantitative-bound automaton*, which is a pair consisting of a quantitative automaton and a bound function. Note that bounds are not constant but depend on the size of the structure over which a specification is interpreted; they are *functions.* We consider *value-bound* functions, which constrain the maximal value of an automaton register, and *iteration-bound* functions, which constrain the maximal number of automaton transitions that need to be analyzed in order to compute the value of the property specified by the automaton. Iteration bounds directly give termination bounds for dynamic programs, and thus better iteration bounds yield faster verification algorithms. In particular, for the battery lifetime property, the generic dynamic-programming algorithms based on iteration bounds are more efficient than the finite-state algorithms derived from value bounds, and they match the best known algorithms that have been devised specifically for the battery lifetime property [6]. Given a value-bound function f, we can always obtain a corresponding iteration-bound function g: for quantitative automata with

$|Q|$ control locations and k registers, and quantitative structures G, the iteration bound $g(G) = O(|Q| \cdot |G| \cdot f(G)^k)$ is sufficient and necessary. Moreover, for certain subclasses of quantitative automata it is possible to derive better iteration bounds. For instance, for *monotonic* quantitative-bound automata (without decreasing register values), we derive iteration-bound functions that are linear with respect to given value-bound functions.

The verification problems for properties specified by quantitative-bound automata are finite-state, and therefore decidable. However, instead of reducing these problems to boolean problems, we provide algorithms that are based on generic and natural, integer-based dynamic programming formulations, where the bound function gives a termination guarantee for the evaluation of the dynamic program. We expect these algorithms to perform well in practice, as they (1) avoid artificial boolean encodings of integers and (2) match, in all the examples we consider, the complexity of the best known property-specific algorithms. The use of bound functions can be viewed as a generalization of bounded model checking [3] from the boolean to the quantitative case. In bounded model checking, the engineer provides a bound on the number of execution steps of a system along with a property. However, the bound is usually a constant independent of the structure, whereas our bound functions capture when search can be terminated without losing information about the structure. Therefore, in bounded model checking, only the structure diameter constitutes a bound function in our sense, because smaller bounds may give counterexamples but not proofs. Of course, as in bounded model checking, our approach could be used to quickly find counterexamples for quantitative verification problems even if the bound function gives values that are smaller than necessary for proof.

Quantitative automata specify dynamic programs. There is a second natural way to specify iterative computation: through the μ-calculus [15]. In a quantitative extension of the μ-calculus, each formula induces a mapping from vertices to integers, and bound functions naturally specify a bound on the number of iterations for evaluating fixpoint expressions. More precisely, for a μ-formula φ, an iteration-bound function g specifies that if, during the iterative calculation of the value of a fixpoint expression in φ on a structure G, a stable value is not reached within $g(G)$ iterations, then the value is infinity. While quantitative extensions of the μ-calculus [13, 16, 10] have been defined before, they were interpreted over probabilistic structures and gave no iteration bounds. Finally, we give a translation from linear-time quantitative-bound automata to the branching-time quantitative-bound μ-calculus. For the purpose of game solving, as in the boolean case, the translation requires that the automaton is deterministic. This gives us symbolic algorithms for the quantitative verification of closed and open systems. Moreover, we show that the relationship [9] between boolean μ-formulas over *transition* graphs and boolean μ-formulas over *game* graphs carries over to the quantitative setting: a quantitative-bound μ-formula computes a particular quantitative property over two-player game graphs iff the formula computes the property over both existential and universal transition graphs (i.e., game graphs where one of the two players has no choices). This shows that the same integer-based symbolic iteration schemes can be used for verifying a quantitative property over both closed and open systems, provided the single-step operation is modified appropriately; this was previously known only for boolean structures, where the dynamic programs are degenerate [9].

2 The Integer-based Quantitative Setting

Quantitative properties. Let P be a nonempty, finite set of *quantitative propositions* (*propositions*, for short). A *quantitative observation* (*observation*, for short) is a function $o\colon P \to \mathbb{N}$ mapping each proposition to a natural number (possibly 0). Let $\mathcal{O}$ be the set of observations. A *quantitative trace* (*trace*, for short) is an infinite sequence $w \in \mathcal{O}^\omega$ of observations. A *quantitative property* (*property*, for short) is a function $\pi\colon \mathcal{O}^\omega \to \mathbb{N} \cup \{\infty\}$ mapping each trace to a natural number or to infinity. Let Π denote the set of properties. These definitions generalize the boolean interpretation [7], where observations are maps from propositions to $\{0,1\}$, and properties are maps from traces to $\{0,1\}$. The following examples describe some quantitative properties.

Example 1 (Response time) Let $P = \{p\}$. Given $a \in \mathbb{N}$, the property $rt_a\colon \mathcal{O}^\omega \to \mathbb{N}$ maps each trace w to $rt_a(w) = \sup\{k \mid \exists w' \in \mathcal{O}^*, w'' \in \mathcal{O}^\omega$ such that $w = w' \cdot (p \mapsto a)^k \cdot w''\}$. Thus, $rt_a(w)$ is the supremal number of consecutive observations mapping the proposition p to the value a in the trace w. This may model the maximal time between a request and a response. The supremum may be infinity. This happens if $w = w' \cdot (p \mapsto a)^\omega$, or if for all $k \geq 0$, the trace w contains a subsequence with at least k successive observations mapping p to a (for example, p may be mapped to $abaabaaabaaaab\ldots$). ∎

Example 2 (Fair maximum) Let $P = \{p, q\}$. The property $fm\colon \mathcal{O}^\omega \to \mathbb{N}$ maps each trace w to the supremal value of the proposition p on w if the proposition q is nonzero infinitely often on w, and to 0 otherwise. The proposition q may model a fairness condition on traces [6]. Formally, $fm(o_0 o_1 o_2 \ldots)$ is $\sup\{o_j(p) \mid j \geq 0\}$ if $\limsup\{o_j(q) \mid j \geq 0\} \neq 0$, and 0 otherwise. The supremum may be infinity. ∎

Example 3 (Lifetime) Let $P = \{p, c\}$. Given $a \in \mathbb{N}$, the property $lt_a\colon \mathcal{O}^\omega \to \mathbb{N}$ maps each trace $w = o_0 o_1 o_2 \ldots$ to $lt_a(w) = \sup\{k \mid \sum_{j=0}^{k} (-1)^{c_j} \cdot o_j(p) \leq a\}$, where $c_j = 0$ if $o_j(c) = 0$, and $c_j = 1$ otherwise. Intuitively, if a zero (resp., nonzero) value $o(c)$ denotes resource consumption (resp., resource gain) in a single step of $o(p)$ units, then $lt_a(w)$ is the supremal number of steps that can be executed without exhausting the resource, given a initial units of the resource. ∎

Example 4 (Peak running total) Let $P = \{p, c\}$ as in the previous example. The property $prt\colon \mathcal{O}^\omega \to \mathbb{N}$ maps each trace $w = o_0 o_1 o_2 \ldots$ to $prt(w) = \sup\{\sum_{j=0}^{k} (-1)^{c_j} \cdot o_j(p) \mid j \geq 0\}$, where again $c_j = 0$ if $o_j(c) = 0$, and $c_j = 1$ otherwise. Intuitively, if a resource is being consumed or gained over the trace w, then $prt(w)$ is the initial amount of the resource necessary so that the resource is never exhausted. ∎

Quantitative structures. A *quantitative system* (*system*, for short) is a tuple $K = (S, \delta, s_0, \langle\cdot\rangle)$, where S is a finite set of states, $\delta \subseteq S \times S$ is a total transition relation, $s_0 \in S$ is an initial state, and $\langle\cdot\rangle\colon S \to \mathcal{O}$ is an observation function that maps each state s to an observation $\langle s\rangle$. A two-player *quantitative game structure* (*game*, for short) is a tuple $G = (S, S_1, S_2, \delta, s_0, \langle\cdot\rangle)$, where S, δ, s_0, and $\langle\cdot\rangle$ are as in systems, and $S_1 \cup S_2 = S$ is a partition of the state space into player-1 states S_1 and player-2 states S_2. At player-1 states, the first player chooses a successor state; at player-2 states, the second player. Note that systems are special cases of games: if $S_i = S$, for $i \in \{1, 2\}$, then the game is called a *player-i system.* We use the term *structure* to refer to both systems and games.

A *trajectory* of the structure G is an infinite sequence $t = r_0 r_1 r_2 \ldots$ of states $r_j \in S$ such that the first state r_0 is the initial state s_0 of G, and $(r_j, r_{j+1}) \in \delta$ for all $j \geq 0$. The trajectory t induces the infinite sequence $\langle t\rangle = \langle r_0\rangle\langle r_1\rangle\langle r_2\rangle \ldots$ of observations. A trace $w \in \mathcal{O}^\omega$ is *generated* by G if there is a trajectory t of G such that $w = \langle t\rangle$. A *player-i strategy,* for $i \in \{1, 2\}$, is a function $\xi_i\colon S^* \times S_i \to S$ that maps every nonempty, finite sequence of states to a successor of the last state in the sequence; that is, $(s, \xi_i(t, s)) \in \delta$ for every state sequence $t \in S^*$ and state $s \in S_i$. Intuitively, $\xi_i(t, s)$ indicates the choice taken by player i according to strategy ξ_i if the current state of the game is s, and the history of the game is t. We write Ξ_i for the set of player-i strategies. For two strategies $\xi_1 \in \Xi_1$ and $\xi_2 \in \Xi_2$, the *outcome* t_{ξ_1,ξ_2} of ξ_1 and ξ_2 is a trajectory of G, namely, $t_{\xi_1,\xi_2} = r_0 r_1 r_2 \ldots$ such that $r_0 = s_0$ and for all $j \geq 0$ and $i \in \{1, 2\}$, if $r_j \in S_i$, then $r_{j+1} = \xi_i(r_0 r_1 \ldots r_{j-1}, r_j)$.

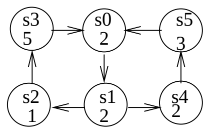

Fig. 1.

Consider the system K shown in Figure 1, with the initial state s_0. Each state s_i of K is labeled with the value $\langle s_i\rangle(p)$ for a proposition p. Consider the property rt_2 from Example 1. For all traces w that correspond to trajectories of K of the form $(s_0 s_1 s_2 s_3)^*$, we have $rt_2(w) = 2$. For all traces w that correspond to trajectories of the form $(s_0 s_1 s_4 s_5)^*$, we have $rt_2(w) = 3$. Moreover, $rt_2(w) \leq 3$ for all traces w generated by K. Now consider a game played on the same structure K, where the state s_1 is a player-2 state. Consider the property lt_{14} from Example 3, supposing that $\langle s\rangle(c) = 0$ for all states s of K. The goal of player 2 is to maximize lifetime given initially 14 units of the resource. Consider the strategy where player 2 chooses s_4 at the first visit to s_1, and chooses s_2 thereafter. This strategy generates a trace w along which p is mapped to $2223\,(2215)^\omega$; hence $lt_{14}(w) = 7$. Note that all memoryless (i.e., history-independent) strategies lead to smaller lifetimes.

3 Quantitative-Bound Automata

3.1 Specifying Quantitative Properties

Syntax. We specify properties using automata. Let $\mathcal{O}$ be a given finite set of observations. Quantitative automata run over input traces in $\mathcal{O}^\omega$. The configuration of a quantitative automaton consists of a control location and an array of registers with values in $\mathbb{N}$. The transitions of quantitative automata are guarded by conditions on the values of the registers and the input observation, and involve, in addition to an update of the control location, also an update of the register values. A k-register *update function* is a recursive function $u \colon \mathbb{N}^k \times \mathcal{O} \rightharpoonup \mathbb{N}^k$ which may be partial. Let U denote the set of update functions. A *quantitative automaton* (*automaton*, for short) is a tuple $A = \langle Q, k, q_0, \gamma \rangle$, where Q is a finite set of control locations, $k \in \mathbb{N}$ is a number of registers, $q_0 \in Q$ is an initial location, and $\gamma \colon Q \to 2^{U \times Q}$ is a transition function that maps each location q to a finite set $\gamma(q)$ of pairs consisting of an update function and a successor location. We require that the transition function γ defines a total relation, namely, for each location $q \in Q$, each observation $o \in \mathcal{O}$, and all register values $\boldsymbol{x} \in \mathbb{N}^k$, there exists $(u, q') \in \gamma(q)$ such that $u(\boldsymbol{x}, o)$ is defined. For technical convenience, we furthermore assume that the automaton has a sink location $q_{halt} \in Q$: if the current location is q_{halt}, then for all observations, the next location is q_{halt} and the values of the registers remain unchanged; that is, $\gamma(q_{halt}) = \{(\lambda \boldsymbol{x}.\, \lambda o.\, \boldsymbol{x}, q_{halt})\}$. We write R for the array of registers, and $R[i] \in \mathbb{N}$ for the value of the i-th register, for $0 \leq i < k$.

Semantics. A *configuration* of the automaton A is a tuple $(q, v_0, v_1, \ldots, v_{k-1}) \in Q \times \mathbb{N}^k$ that specifies the current control location and the values of the registers. The initial configuration of the automaton is $c_{init} = (q_0, 0, 0, \ldots, 0)$, where all k registers are initialized to 0. For an input $o \in \mathcal{O}$, the configuration $c' = (q', v'_0, v'_1, \ldots, v'_{k-1})$ is an *o-successor* of the configuration $c = (q, v_0, v_1, \ldots, v_{k-1})$, denoted by $c \xrightarrow{o} c'$, if there is a transition $(u, q') \in \gamma(q)$ such that $u(v_0, v_1, \ldots, v_{k-1}, o) = (v'_0, v'_1, \ldots, v'_{k-1})$. A *run* of the automaton A over a trace $o_0 o_1 o_2 \ldots \in \mathcal{O}^\omega$ is an infinite sequence $c_0 c_1 c_2 \ldots$ of configurations such that $c_0 = c_{init}$, and $c_j \xrightarrow{o_j} c_{j+1}$ for all $j \geq 0$. The *value* of the run $r = c_0 c_1 c_2 \ldots$ is defined as $val_A(r) = \limsup\{R[0](c_j) \mid j \geq 0\}$, that is, the value of r is the maximal value of the register $R[0]$ which occurs infinitely often along r, if this maximum is bounded; and otherwise the value is infinity. In other words, $val_A(r) = \infty$ iff for all $k \geq 0$, the value of the register $R[0]$ is infinitely often greater than k.

An automaton is *monotonic* if along every run, the value of each register cannot decrease. An automaton is *deterministic* if for every configuration c and input $o \in \mathcal{O}$, there is exactly one o-successor of c. While a deterministic automaton has a single run over every input trace, in general an automaton may have several runs over a given trace, each with a possibly different value. According to the *nondeterministic* (or *existential*) interpretation of automata, the *value* of an automaton A over a trace w, denoted $val_A^{\mathrm{nondet}}(w)$, is the supremal value of all runs of A over w. Formally, $val_A^{\mathrm{nondet}}(w) = \sup\{val_A(r) \mid r \text{ is a run of } A \text{ with } \langle r \rangle = w\}$. An alternative is the *universal* interpretation of automata, where the *value* of A over a trace w, denoted $val_A^{\mathrm{univ}}(w)$, is the infimal value of all runs of A over w; that is, $val_A^{\mathrm{univ}}(w) = \inf\{val_A(w, r) \mid r \text{ is a run of } A \text{ with } \langle r \rangle = w\}$. Note that a deterministic automaton A can be viewed as both a nondeterministic and a universal automaton. The (nondeterministic) automaton A *specifies* (or *com-*

putes) the property $\pi \in \Pi$ if for all traces $w \in \mathcal{O}^\omega$, we have $val_A^{\mathrm{nondet}}(w) = \pi(w)$. This definition captures traditional Büchi automata as a special case: keep one register $R[0]$, which is set to 1 whenever the automaton visits a Büchi accepting control location, and set to 0 otherwise.

Model checking and game solving. Let K be a quantitative system. For a quantitative automaton A, the *max-value* of K with respect to A, denoted $val_A^{\max}(K)$, is the supremal value of all traces generated by K, where we choose the nondeterministic (rather than the universal) interpretation of automata. Formally, $val_A^{\max}(K) = \sup\{val_A^{\mathrm{nondet}}(w) \mid w \text{ is a trace generated by } K\}$. The *min-value* of K with respect to A, denoted $val_A^{\min}(K)$, is the infimal value of all traces generated by K; that is, $val_A^{\min}(K) = \inf\{val_A^{\mathrm{nondet}}(w) \mid w \text{ is a trace generated by } K\}$. Now consider a game G. The value of a strategy pair $\xi_1 \in \Xi_1$ and $\xi_2 \in \Xi_2$ with respect to a *deterministic* automaton A is the value $val_A(\xi_1, \xi_2) = val_A(t_{\xi_1,\xi_2})$ of A over the outcome of the strategies ξ_1 and ξ_2. The *game-value* of G with respect to a deterministic automaton A, denoted $val_A^{\mathrm{maxmin}}(G)$, is defined as $\sup_{\xi_1 \in \Xi_1} \inf_{\xi_2 \in \Xi_2} val_A(\xi_1, \xi_2)$. This is the supremal value of A that player-1 can achieve against all player-2 strategies. The symmetric definition is omitted for brevity.

Given a system K and an automaton A, the *quantitative model-checking problem* (*model checking*, for short) is to determine $val_A^{\max}(K)$ and $val_A^{\min}(K)$. Given a game G and a deterministic automaton A, the *quantitative game-solving problem* (*game solving*, for short) is to determine $val_A^{\mathrm{maxmin}}(G)$. Since registers can contain arbitrary natural numbers, we can encode 2-counter machines as monotonic automata, and hence the model-checking and game-solving problems are undecidable.

3.2 Bound Functions for Automata

Quantitative-bound automata. In order to solve model-checking problems and games, we equip quantitative automata with bound functions. A *quantitative-bound automaton* (QBA) (A, f) consists of a quantitative automaton A and a recursive function $f\colon \mathcal{G} \to \mathbb{N}$, where $\mathcal{G}$ is the set of quantitative structures (systems and games). To compute a property on a structure G, a QBA works with a bound $f(G)$ that depends on G. The motivation is that for many properties, the designer can provide a bound on the maximal value of the automaton registers, or on the number of automaton transitions that need to be executed in order to compute the value of the property if the value is finite. We thus have two interpretations of the bound function f: the *value-bound* interpretation, where $f(G)$ is a bound on the register values, and the *iteration-bound* interpretation, where $f(G)$ is a bound on the automaton transitions.

We define the value of a QBA over a trace generated by a structure for the two possible interpretations. Given a QBA (A, f), a structure G, and a trace w generated by G, let r be a run of the automaton A over w. The value of $r = c_0c_1c_2\ldots$ over w for the value-bound interpretation, denoted $val_{\mathrm{vbound}(A,f)}(r)$, is defined as follows: if there are an index $j \in \mathbb{N}$ and a register

$R[i]$, for $0 \leq i < k$, such that $R[i](c_j) > f(G)$, then $val_{\text{vbound}(A,f)}(r) = \infty$; otherwise $val_{\text{vbound}(A,f)}(r) = val_A(r)$. Intuitively, the value-bound interpretation maps every trace that causes some register to exceed the value bound at some point, to ∞. The value of the run r over w for the iteration-bound interpretation, denoted $val_{\text{ibound}(A,f)}(r)$, is defined as follows: if for all $0 \leq i < k$, we have $\max\{R[i](c_j) \mid f(G) \leq j \leq 2 \cdot f(G)\} = \max\{R[i](c_j) \mid 2 \cdot f(G) \leq j \leq 3 \cdot f(G)\}$, then $val_{\text{ibound}(A,f)}(r) = \max\{R[0](c_j) \mid f(G) \leq j \leq 2 \cdot f(G)\}$; otherwise $val_{\text{ibound}(A,f)}(r) = \infty$. Intuitively, the iteration-bound interpretation checks if the maximal values of all registers stabilize within the iteration bound, and maps a trace to ∞ if some maximal register value does not stabilize.

Given a QBA (A, f), a system K, a game G, a trace generated by K or G, and two interpretations bound $\in$ {vbound, ibound}, we define the values $val^{\text{nondet}}_{\text{bound}(A,f)}(w)$, $val^{\max}_{\text{bound}(A,f)}(K)$, $val^{\min}_{\text{bound}(A,f)}(K)$, and $val^{\text{maxmin}}_{\text{bound}(A,f)}(G)$ analogous to the corresponding definitions in Section 3.1 using $val_{\text{bound}(A,f)}(r)$ instead of $val_A(r)$. The QBA (A, f) *specifies* (or *computes*) the property π on a structure G if for all traces w generated by G, we have $val^{\text{nondet}}_{\text{bound}(A,f)}(w) = \pi(w)$. The following examples illustrate the idea.[2]

Example 5 (Fair maximum) The following QBA (A, f) specifies the property *fm* from Example 2 on all structures G. There are two registers. The register $R[1]$ keeps track of the maximal value of proposition p seen so far. Whenever proposition q has a nonzero value, the value of $R[1]$ is copied to $R[0]$; otherwise $R[0]$ is set to zero. If q has a nonzero value infinitely often, then the maximal value of p occurs infinitely often in $R[0]$; otherwise from some point on, $R[0]$ contains the value 0. The bound function f is defined as follows: if G contains the maximal value Δ for p, then $f(G) = \Delta$ is a suitable value-bound function; if G has N states, then $f(G) = N$ is a suitable iteration-bound function. ∎

Example 6 (Lifetime) The property lt_a from Example 3 can be computed on all structures G by the following QBA (A, f). Let $A = \langle \{q_0, q_{halt}\}, 2, q_0, \gamma \rangle$, where for all inputs $o \in \mathcal{O}$, we have $\gamma(q_0) = \{(o(c) \neq 0 \wedge R'[0] = R[0] + 1 \wedge R'[1] = R[1] - o(p), q_0), (o(c) = 0 \wedge R[1] + o(p) \leq a \wedge R'[0] = R[0] + 1 \wedge R'[1] = R[1] + o(p), q_0), (o(c) = 0 \wedge R[1] + o(p) > a \wedge R'[0] = R[0] \wedge R'[1] = R[1], q_{halt})\}$. In register $R[0]$ the automaton stores the number of transitions already taken, and in $R[1]$ it tracks the amount of the resource used so far; it continues to make transitions as long as it has a sufficient amount of the resource. If G contains N states and the maximal value Δ for p, then $f(G) = a + (N + 1) \cdot \Delta$ is a suitable value-bound function, and $f(G) = N \cdot a + N \cdot (N + 1) \cdot \Delta$ is a suitable iteration-bound function. ∎

3.3 Quantitative-Bound Model Checking and Game Solving

Given a system K and a QBA (A, f), the *quantitative-bound model-checking* problem is to determine $val^{m}_{\text{bound}(A,f)}(K)$, where bound $\in$ {vbound, ibound} and

[2] In the examples, we write update functions as relations $u(\boldsymbol{x}, o, \boldsymbol{x}')$, where unprimed variables denote the values of variables before the update, and primed variables denote the values after the update.

$m \in \{\max, \min\}$. Similarly, given a game G and a deterministic QBA (A, f), the problem of *solving quantitative-bound games* is to determine $val^{\mathrm{maxmin}}_{\mathrm{bound}(A,f)}(G)$, for bound $\in \{\mathrm{vbound}, \mathrm{ibound}\}$. Quantitative-bound model checking and game solving are decidable. In the case of value bounds, the state space is bounded by $O(|G| \cdot |Q| \cdot (f(G) + 2)^k)$, where $|Q|$ is the size of the automaton with k registers, $|G|$ is the size of the structure, and f is the value-bound function. Let G be a structure such that for all propositions $p \in P$ and states $s \in S$, we have $\langle s \rangle(p) \leq \Delta$. Let C_0 be the maximal constant that appears syntactically in the description of the automaton A, and let $C_1 = f(G)$. Call $B = \max\{\Delta, C_0, C_1\}$ the *oblivion bound* for the QBA (A, f) and structure G. Let $g(G) = |G| \cdot |Q| \cdot (B + 2)^k$, where A has k registers. Then $val^m_{\mathrm{vbound}(A,f)}(G) = val^m_{\mathrm{ibound}(A,g)}(G)$, for $m \in \{\max, \min, \mathrm{maxmin}\}$. Thus, we can derive an iteration bound from a value bound.

Formally, the decision problem QBA-VMC (resp., QBA-VGS) takes as input a system K (resp., game G), a QBA (A, f), the oblivion bound B, and a value $a \in \mathbb{N} \cup \{\infty\}$, and returns "Yes" if $val^{\max}_{\mathrm{vbound}(A,f)}(K) \geq a$ (resp., $val^{\mathrm{maxmin}}_{\mathrm{vbound}(A,f)}(G) \geq a$). The decision problems QBA-IMC and QBA-IGS are defined analogously using $val^{\max}_{\mathrm{ibound}(A,f)}(K)$ and $val^{\mathrm{maxmin}}_{\mathrm{ibound}(A,f)}(G)$. We give the oblivion bound as an input to the problems, because the value of $f(G)$ can be unboundedly larger than the descriptions of f and G. We assume that updates take unit time.

Theorem 1 *(1) QBA-VMC is PSPACE-complete and QBA-IMC is EXPTIME-complete. (2) QBA-VGS and QBA-IGS are EXPTIME-complete. (3) Let $|G|$ be the size of the structure and $|Q|$ the automaton size for (A, f) and G. Let $S = |Q| \cdot |G| \cdot (f(G) + 2)^k)$. QBA-VMC and QBA-VGS can be solved in time $O(S)$ and $O(S^2)$ respectively. QBA-IMC and QBA-IGS can be solved in time $O(|Q| \cdot |G| \cdot f(G))$.*

Note that these complexity results reflect the sizes of the state space in which the solution lies. In practice, however, the reachable state space can be much smaller. Hence, on-the-fly state space exploration can be used instead of constructing the entire state space *a priori*. The following examples show that our approach, while being generic and capturing several interesting quantitative verification problems [6] as special cases, still remains amenable to efficient analysis.

Example 7 (Fair maximum) Consider the deterministic QBA (A, f) with value-bound function f from Example 5, which computes the property *fm* from Example 2. This property is exactly the winning condition for the "threshold Büchi games" described in [6]. For a game G, the state space with the value bound has size $O(|G| \cdot |Q| \cdot \Delta)$, where Δ is the maximal value of proposition p in G. This is exponential in $|G|$. However, the iteration bound for this problem is $|G|$, and this gives an $O(|G|^2)$ algorithm, which is the same complexity as the algorithm of [6].[3] ∎

[3] However, computing an iteration-bound function automatically using the optimal value-bound function would lead to a suboptimal iteration-bound function $g(G) = |G| \cdot |Q| \cdot \Delta$.

Example 8 (Peak running total) The property *prt* from Example 4 is exactly the winning condition for the "energy games" of [6]. This property can be computed by a deterministic QBA with two registers and value-bound function $f(G) = |G| \cdot \Delta$, where Δ is the maximal value of p in G. A game-solving algorithm based on value bounds would require time $O(|G|^6 \cdot \Delta^4)$, whereas an algorithm designed specifically to solve this game [6] runs in time $O(|G|^3 \cdot \Delta)$. However, even for this problem, our generic approach, using the optimal iteration-bound function $h(G) = |G|^2 \cdot \Delta$ achieves the best known complexity of $O(|G|^3 \cdot \Delta)$. ∎

In the special case of monotonic automata, efficient iteration bounds can be automatically derived from value bounds. Consider a structure G with N states and a monotonic QBA (A, f) with value-bound function f, location set Q, and k registers. Since the value of each register only increases, within $|Q| \cdot k \cdot N \cdot f(G)$ steps of every run of A over a trace generated by G, either an automaton configuration repeats, or there is a register such that the value of the register has crossed the threshold $f(G)$. Thus $val^{\max}_{\text{vbound}(A,f)}(G)$ is achieved by a run within $|Q| \cdot k \cdot N \cdot f(G)$ steps. Since we only require the monotonicity of the registers in the limit, this observation can be generalized to *reversal-bounded automata* [18], where a bounded number of switches between increasing and decreasing modes of the registers are allowed.

Proposition 1 *Let A be a monotonic automaton with location set Q and k registers, let $f \colon \mathcal{G} \to \mathbb{N}$ be a recursive function, and let $g(G) = |Q| \cdot k \cdot N \cdot f(G)$ for all structures G with N states. Then $val^{m}_{\text{vbound}(A,f)}(G) = val^{m}_{\text{ibound}(A,g)}(G)$ for all structures G and $m \in \{\max, \min, \text{maxmin}\}$.*

As with the other components of a quantitative automaton, the designer has to provide the bound function f. Unfortunately, the task of providing a good value or iteration bound function f, that is, an f that satisfies $val^{m}_{A}(G) = val^{m}_{\text{bound}(A,f)}(G)$ for all structures G, cannot be automated.

Proposition 2 *There is a class of update functions involving only increment operations and equality testing on registers, such that the following two problems are undecidable: (1) given an automaton A, determine if there is a recursive function f such that $val^{\max}_{A}(K) = val^{\max}_{\text{vbound}(A,f)}(K)$ for all systems K; (2) given a QBA (A, f), determine if $val^{\max}_{A}(K) = val^{\max}_{\text{vbound}(A,f)}(K)$ for all systems K.*

4 The Quantitative-Bound μ-Calculus

We now provide an alternative formalism for defining quantitative properties: a fixpoint calculus. Our integer-based μ-calculus generalizes the classical μ-calculus [15], and provides an alternative set of iterative algorithms for model checking and game solving.

Unbounded formulas. Let P be a set of propositions, let $\mathcal{X}$ be a set of variables, and let $\mathcal{F}$ be a set of recursive functions from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. We require that

$\max, \min \in \mathcal{F}$. The formulas of the *quantitative μ-calculus*[4] are defined as

$$\varphi ::= k \mid p \mid X \mid \mathit{upd}(\varphi,\varphi) \mid \mathit{pre}(\varphi) \mid \mu[(X,\varphi),\ldots,(X,\varphi)] \mid \nu[(X,\varphi),\ldots,(X,\varphi)],$$

where k ranges over the constants in $\mathbb{N} \cup \{\infty\}$, p over the propositions in P, X over the variables in $\mathcal{X}$, and *upd* over the functions in $\mathcal{F}$. If *pre* ranges over the set $\{\mathit{Epre}, \mathit{Apre}\}$ of existential and universal next-time operators, we obtain the *system calculus*; if *pre* ranges over the set $\{\mathit{Cpre}_1, \mathit{Cpre}_2\}$ of player-1 and player-2 controllable next-time operators, we obtain the *game calculus*. Each least-fixpoint subformula $\mu[(X_1,\varphi_1),\ldots,(X_m,\varphi_m)]$ and each greatest-fixpoint subformula $\nu[(X_1,\varphi_1),\ldots,(X_m,\varphi_m)]$ binds a set $\{X_1,\ldots,X_m\}$ of variables. A formula φ is *closed* if all occurrences of variables in φ are bound.

The formulas of the quantitative μ-calculus are interpreted over quantitative structures (systems or games). Consider a game $G = (S, S_1, S_2, \delta, s_0, \langle\cdot\rangle)$. A *quantitative valuation* (*valuation*, for short) is a function $\theta\colon S \to \mathbb{N} \cup \{\infty\}$ that maps each state s to a natural number or infinity. We write Θ for the set of valuations. The semantics $[\![\varphi]\!]$ of a closed formula φ over the structure G is a valuation in Θ, which is defined as follows. An *environment* $\mathbb{E}\colon \mathcal{X} \to \Theta$ maps each variable to a valuation. Given an environment $\mathbb{E}$, we write $\mathbb{E}[X := \theta]$ for the environment that maps X to θ, and maps each $Y \in \mathcal{X} \setminus \{X\}$ to $\mathbb{E}(Y)$. Each update function $\mathit{upd} \in \mathcal{F}$ defines a transformer $[\mathit{upd}]\colon \Theta \times \Theta \to \Theta$ that maps a pair of valuations to the valuation obtained by the point-wise application of *upd*. Each next-time operator *pre* defines a transformer $[\mathit{pre}]\colon \Theta \to \Theta$ that maps valuations to valuations. Specifically, $[\mathit{Epre}](\theta)(s) = \max\{\theta(s') \mid (s,s') \in \delta\}$; $[\mathit{Apre}](\theta)(s) = \min\{\theta(s') \mid (s,s') \in \delta\}$; $[\mathit{Cpre}_1](\theta)(s) = [\mathit{Epre}](\theta)(s)$ if $s \in S_1$, and $[\mathit{Cpre}_1](\theta)(s) = [\mathit{Apre}](\theta)(s)$ if $s \in S_2$; $[\mathit{Cpre}_2](\theta)(s) = [\mathit{Apre}](\theta)(s)$ if $s \in S_1$, and $[\mathit{Cpre}_2](\theta)(s) = [\mathit{Epre}](\theta)(s)$ if $s \in S_2$. For an environment $\mathbb{E}$, the semantics $[\![\varphi]\!]_{\mathbb{E}}$ of a (not necessarily closed) formula φ over G is defined inductively:

$$\begin{array}{l}
[\![k]\!]_{\mathbb{E}}(s) \;=\; k; \quad [\![p]\!]_{\mathbb{E}}(s) \;=\; \langle s\rangle(p); \quad [\![X]\!]_{\mathbb{E}}(s) \;=\; \mathbb{E}(X)(s); \\
[\![\mathit{upd}(\varphi_1,\varphi_2)]\!]_{\mathbb{E}}(s) \;=\; [\mathit{upd}]([\![\varphi_1]\!]_{\mathbb{E}}, [\![\varphi_2]\!]_{\mathbb{E}})(s); \\
[\![\mathit{pre}(\varphi)]\!]_{\mathbb{E}}(s) \;=\; [\mathit{pre}]([\![\varphi]\!]_{\mathbb{E}})(s); \\
[\![\mu[(X_1,\varphi_1),\ldots,(X_m,\varphi_m)]]\!]_{\mathbb{E}}(s) \;=\; \limsup\{\mathbb{E}^{\mu}_{j}(X_1)(s) \mid j \geq 0\}; \\
[\![\nu[(X_1,\varphi_1),\ldots,(X_m,\varphi_m)]]\!]_{\mathbb{E}}(s) \;=\; \limsup\{\mathbb{E}^{\nu}_{j}(X_1)(s) \mid j \geq 0\}.
\end{array}$$

The environment $\mathbb{E}^{\mu}_{j}$ is defined inductively by $\mathbb{E}^{\mu}_{0}(X_i) = (\lambda s.\, 0)$ and $\mathbb{E}^{\mu}_{j+1}(X_i) = [\![\varphi_i]\!]_{\mathbb{E}^{\mu}_{j}}$ for all $1 \leq i \leq m$; and $\mathbb{E}^{\mu}_{j}(Y) = \mathbb{E}(Y)$ for all $Y \in \mathcal{X} \setminus \{X_1,\ldots,X_m\}$ and $j \geq 0$. The environment $\mathbb{E}^{\nu}_{j}$ is defined like $\mathbb{E}^{\mu}_{j}$ except that $\mathbb{E}^{\nu}_{0}(X_i) = (\lambda s.\, \infty)$ for all $1 \leq i \leq m$. For monotone boolean formulas, the limsup semantics coincides with the usual fixpoint semantics of the μ-calculus [15]. For a closed formula φ, we define $[\![\varphi]\!]$ as $[\![\varphi]\!]_{\mathbb{E}}$, for an arbitrary environment $\mathbb{E}$. Given a structure G, the closed formula φ *specifies* the valuation $[\![\varphi]\!](G) = [\![\varphi]\!](s_0)$, where s_0 is the initial state of G.

Bound functions. A *quantitative-bound μ-formula* (QBF) (φ, f) consists of a quantitative μ-formula φ and a recursive function $f\colon \mathcal{G} \to \mathbb{N}$ that provides

[4] This is different from the μ-calculi over probabilistic systems defined by [13, 10, 16].

a bound $f(G)$ on the number of iterations necessary for evaluating μ and ν subformulas on any given structure G. The semantics $[\![(\varphi, f)]\!]_{\mathbb{E}}$ of a QBF (φ, f) over a structure G is defined like the semantics of the unbounded formula φ except that each fixpoint subformula is computed by unrolling the fixpoint only $O(f(G))$ times. Formally, a variable X is $f(G)$-*stable* at a state s with respect to a sequence $\{\mathbb{E}_j \mid j \geq 0\}$ of environments if $\max\{\mathbb{E}_j(X)(s) \mid f(G) \leq j \leq 2 \cdot f(G)\} = \max\{\mathbb{E}_j(X)(s) \mid 2 \cdot f(G) \leq j \leq 3 \cdot f(G)\}$. We define $[\![\mu[(X_1, \varphi_1), \ldots, (X_m, \varphi_m)], f]\!](s)$ to be $\max\{\mathbb{E}^{\mu}_j(X_1)(s) \mid f(G) \leq j \leq 2 \cdot f(G)\}$ if all variables X_i, for $1 \leq i \leq m$, are $f(G)$-stable with respect to $\{\mathbb{E}^{\mu}_j \mid j \geq 0\}$; otherwise $[\![\mu[(X_1, \varphi_1), \ldots, (X_m, \varphi_m)], f]\!](s) = \infty$. The semantics $[\![\nu[(X_1, \varphi_1), \ldots, (X_m, \varphi_m)], f]\!]$ of greatest-fixpoint subformulas is defined analogously, using the sequence $\{\mathbb{E}^{\nu}_j \mid j \geq 0\}$ of environments instead. A QBF formula (φ, f) defines an iterative algorithm for computing the valuation $[\![(\varphi, f)]\!](G)$ for any given structure G. Assuming updates take unit time, we can compute $[\![(\varphi, f)]\!](G)$ in $O(f(G)^{\ell})$ time, where ℓ is the alternation depth of φ (i.e., the maximal number of alternations between occurrences of μ and ν operators; for a precise definition see [11]).

We now give examples for which a QBF (φ, f) can be found to specify the same property as the unbounded formula φ over all structures; that is, $[\![(\varphi, f)]\!](G) = [\![\varphi]\!](G)$ for all structures G. We use addition, subtraction, and comparison as update functions in $\mathcal{F}$, and we use the natural numbers 0 and 1 to encode booleans. For instance, we write $\varphi_1 = \varphi_2$ for $\min(\varphi_1 \leq \varphi_2, \varphi_2 \leq \varphi_1)$, and $\neg\varphi_1$ for $1 - \varphi$. The case formula $\text{case}\{(\psi_1, \varphi_1), \ldots, (\psi_n, \varphi_n)\}$ stands for $\max(\min(\psi_1, \varphi_1), \ldots, \min(\psi_n, \varphi_n))$, where the n-ary max operator is obtained by repeated application of the binary max operator. In order to relate the branching-time framework of the quantitative μ-calculus to the linear-time framework of quantitative properties (and quantitative automata), we say that the closed QBF (φ, f) *computes* the property π if for all structures G, $[\![(\varphi, f)]\!](G) = \sup\{\pi(w) \mid w \text{ is a trace generated by } G\}$. In this way, linear and branching time are related *existentially* (through sup rather than inf); hence we use only the *Epre* operator to compute properties. Alternately, we could define a universal semantics where $[\![(\varphi', f)]\!](G) = \inf\{\pi(w) \mid w \text{ is a trace generated by } G\}$, and the *Apre* operator is used.

Example 9 (Fair maximum) Recall the property *fm* from Example 2. The property *fm* is computed over all structures G by the QBF (φ, f) with $\varphi = \mu[(X, \min\{\max\{p, X, \min\{\mathit{Epre}(X), Z\}\}, Z\})]$, where $Z = \nu[(X, \mu[(Y, \mathit{Epre}(\max\{\min\{q, X\}, Y\})])]$, and $f(G) = N$, where N is the number of states of G. Since the longest simple path in G has length at most $N - 1$, every fixpoint is found in N iterations or less. ∎

Example 10 (Lifetime) Over all structures G with N states, the property lt_a from Example 3 is computed by the QBF (φ, f) with $\varphi = \mu[(X, \text{case}\{((c = 0) \wedge (p + \mathit{Epre}(Y) \leq a), X + 1), (c \neq 0, X + 1), (1, X)\}), (Y, \text{case}\{(((c = 0) \wedge (p + \mathit{Epre}(Y) \leq a)), p + \mathit{Epre}(Y)), (c \neq 0, \mathit{Epre}(Y) - a), (1, Y)\})]$ and $f(G) = N \cdot a + N \cdot (N + 1) \cdot \Delta$, where Δ is the maximal value of the proposition p in G.

If a fixpoint is not reached in $N \cdot a + N \cdot (N+1) \cdot \Delta$ iterations, then there is a reachable cycle Γ in G with nonpositive resource consumption, and repeated traversal of Γ ensures an infinite lifetime. ∎

Example 11 (Peak running total) Over all structures G with N states and maximal value Δ for the proposition p, the property *prt* from Example 4 is computed by the QBF (φ, f) with $\varphi = (\mu[(X, \text{case}\{(c = 0, p+\max\{0, \mathit{Epre}(X)\}), (c \neq 0, \max\{0, \mathit{Epre}(X)\} - p)\})], f)$ and $f(G) = N \cdot \Delta$. If a fixpoint is not reached in $N \cdot \Delta$ iterations, then there is no reachable cycle with nonpositive resource consumption, and it is not possible to traverse G forever starting with a finite amount of resources. ∎

From automata bounds to μ-calculus bounds. We establish the connection between properties specified by quantitative automata (a linear-time formalism) and those computed by the quantitative μ-calculus (a branching-time formalism). We show that every deterministic QBA can be converted to a QBF that computes the same property over all systems. This provides an alternative algorithm for quantitative model checking. We then show that the construction is robust [9], and hence, the resulting QBF can also be used for game solving. To formalize this, we define a quantitative μ-calculus over traces, extending the boolean linear-time μ-calculus [17]. The *quantitative-bound trace formulas* (QBTs) are identical to the quantitative-bound μ-formulas, except that they contain the single next-time operator *Pre*. A QBT is interpreted over the traces w generated by a given structure G. To define $[\![(\varphi, f)]\!](w)$ formally, we view the trace $w = o_0 o_1 o_2 \ldots$ as an infinite-state system without branching, analogous to the boolean definition in [9]. However, even though w is infinite-state, the evaluation of every fixpoint subformula in φ is bounded by $f(G)$, which is finite.

Consider a structure K, a game G, and a QBT (φ, f). The system value $\mathit{val}^{\max}_{(\varphi,f)}(K)$ (resp., $\mathit{val}^{\min}_{(\varphi,f)}(K)$) is the supremal (resp., infimal) value of the formula (φ, f) over all traces generated by K. Formally, $\mathit{val}^{\max}_{(\varphi,f)}(K) = \sup\{[\![(\varphi, f)]\!](w) \mid w \text{ is a trace generated by } K\}$, and $\mathit{val}^{\min}_{(\varphi,f)}(K)$ is the inf of the same set. For strategies $\xi_1 \in \Xi_1$ and $\xi_2 \in \Xi_2$, define $\mathit{val}_{(\varphi,f)}(\xi_1, \xi_2) = [\![(\varphi, f)]\!](\langle t_{\xi_1,\xi_2}\rangle)$. The game value $\mathit{val}^{\max\min}_{(\varphi,f)}(G) = \sup_{\xi_1 \in \Xi_1} \inf_{\xi_2 \in \Xi_2} \mathit{val}_{(\varphi,f)}(\xi_1, \xi_2)$ is the supremal value that player 1 can achieve against all player-2 strategies. The following two theorems generalize the results of [9] from boolean to quantitative verification: Theorem 2 establishes the connection between deterministic QBAs and QBTs; Theorem 3 presents a necessary and sufficient criterion, called *robustness*, when a QBT can be used for game solving. Moreover, the QBT constructed in Theorem 2 is robust. Given a QBT (φ, f), let $(\varphi[\mathit{Epre}], f)$ (resp., $(\varphi[\mathit{Apre}], f)$) be the QBF that results by replacing all occurrences of the next-time operator *Pre* with *Epre* (resp., *Apre*).

Theorem 2 *Every deterministic QBA (A, f) can be translated into a QBT (φ, g) such that for all systems K, both $\mathit{val}^{\max}_{(A,f)}(K) = \mathit{val}^{\max}_{(\varphi,g)}(K) = [\![(\varphi[\mathit{Epre}], g)]\!](K)$ and $\mathit{val}^{\min}_{(A,f)}(K) = \mathit{val}^{\min}_{(\varphi,g)}(K) = [\![(\varphi[\mathit{Apre}], g)]\!](K)$.*

Theorem 3 *Given a QBT (φ, f), the following two conditions, called* robustness*, are equivalent. (1) For all systems K, both $val^{\max}_{(\varphi,f)}(K) = [\![(\varphi[Epre], f)]\!](K)$ and $val^{\min}_{(\varphi,f)}(K) = [\![(\varphi[Apre], f)]\!](K)$. (2) For all games G, $val^{\max\min}_{(\varphi,f)}(G) = [\![(\varphi[Cpre_1], f)]\!](G)$.*

Theorem 2 is proved using a standard (boolean) construction of a fixpoint formula from an automaton [8]. Theorem 3 follows from the existence of finite-memory optimal strategies for QBTs.

Acknowledgments. This research was supported in part by the ONR grant N00014-02-1-0671, the AFOSR MURI grant F49620-00-1-0327, and the NSF grants CCR-0225610, CCR-0234690, and CCR-0427202.

References

1. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49:672–713, 2002.
2. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS*, LNCS 1026, pp. 499-513. Springer, 1995.
3. A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, LNCS 1579, pp. 193–207. Springer, 1999.
4. P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information & Computation*, 182:137–162, 2003.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information & Computation*, 98:142–170, 1992.
6. A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT*, LNCS 2855, pp. 117–133. Springer, 2003.
7. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.
8. M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.
9. L. de Alfaro, T.A. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for ω-regular objectives. In *LICS*, pp. 279–290. IEEE, 2001.
10. L. de Alfaro, and R. Majumdar. Quantitative solution of concurrent games. *J. Computer & Systems Sciences*, 68: 374–397, 2004.
11. E.A. Emerson and C. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *LICS*, pp. 267–278. IEEE, 1986.
12. G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, 1991.
13. M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *LICS*, pp. 111–122, IEEE, 1997.
14. O.H. Ibarra, J. Su, Z. Dang, T. Bultan, and R.A. Kemmerer. Counter machines: Decidable properties and applications to verification problems. In *MFCS*, LNCS 1893, pp. 426-435. Springer, 2000.
15. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
16. A. McIver and C. Morgan. Games, probability, and the quantitative μ-calculus. In *LPAR*, LNCS 2514, pp. 292-310. Springer, 2002.
17. M.Y. Vardi. A temporal fixpoint calculus. In *POPL*, pp. 250–259. ACM, 1988.
18. G. Xie, Z. Dang, O.H. Ibarra, and P.S. Pietro. Dense counter machines and verification problems. In *CAV*, LNCS 2725, pp. 93–105. Springer, 2003.