A Measured Collapse of The Modal μ-Calculus Alternation Hierarchy

Doron Bustan[1], Orna Kupferman[2], and Moshe Y. Vardi[1]

[1] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email:{doron_b,vardi}@cs.rice.edu***
[2] Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il†

Abstract. The μ-calculus model-checking problem has been of great interest in the context of concurrent programs. Beyond the need to use symbolic methods in order to cope with the state-explosion problem, which is acute in concurrent settings, several concurrency related problems are naturally solved by evaluation of μ-calculus formulas. The complexity of a naive algorithm for model checking a μ-calculus formula ψ is exponential in the alternation depth d of ψ. Recent studies of the μ-calculus and the related area of parity games have led to algorithms exponential only in $\frac{d}{2}$. No symbolic version, however, is known for the improved algorithms, sacrificing the main practical attraction of the μ-calculus.
The μ-calculus can be viewed as a fragment of first-order fixpoint logic. One of the most fundamental theorems in the theory of fixpoint logic is the *Collapse Theorem*, which asserts that, unlike the case for the μ-calculus, the fixpoint alternation hierarchy over finite structures collapses at its first level. In this paper we show that the Collapse Theorem of fixpoint logic holds for a measured variant of the μ-calculus, which we call $\mu^{\#}$-calculus. While μ-calculus formulas represent characteristic functions, i.e., functions from the state space to $\{0,1\}$, formulas of the $\mu^{\#}$-calculus represent measure functions, which are functions from the state space to some measure domain. We prove a *Measured-Collapse Theorem*: every formula in the μ-calculus is equivalent to a least-fixpoint formula in the $\mu^{\#}$-calculus. We show that the Measured-Collapse Theorem provides a logical recasting of the improved algorithm for μ-calculus model-checking, and describe how it can be implemented symbolically using Algebraic Decision Diagrams. Thus, we describe, for the first time, a symbolic μ-calculus model-checking algorithm whose complexity matches the one of the best known enumerative algorithm.

1 Introduction

The *modal μ-calculus*, often referred to as the "μ-calculus", is a propositional modal logic augmented with least and greatest fixpoint operators. It was introduced in [22], following earlier studies of fixpoint calculi in the theory of program correctness [11, 31, 32]. Over the past 20 years, the μ-calculus has been established as essentially the

*** Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, and IIS-9978135, by BSF grant 9800096, and by a grant from the Intel Corporation.
† Supported in part by BSF grant 9800096, and by a grant from Minerva.

"ultimate" program logic, as it expressively subsumes all propositional program logics, including dynamic logics such as PDL, process logics such as YAPL, and temporal logics such as CTL* [13]. The μ-calculus has gained further prominence with the discovery that its formulas can be evaluated symbolically in a natural way [6], leading to industrial acceptance of computer-aided verification.

A central issue for any logic is the *model-checking* problem: is a given structure a model of a given formula. For modal logics we ask whether a given formula holds in a given state of a given Kripke structure. The μ-calculus model-checking problem has been of great interest in the context of concurrent programs. A significant feature of expressing model checking in terms of the μ-calculus is that it naturally leads to *symbolic* algorithms, which operates on sets of states, and can scale up to handle exceedingly large state spaces [28]. Beyond the need to use symbolic methods in order to cope with the state-explosion problem [6], which is acute in concurrent settings, several concurrency-related problems are naturally solved by evaluation of μ-calculus formulas. This includes checks for fair simulation between two components of a concurrent systems [14, 16] and reasoning about the interaction between a component and its environment, which is naturally expressed by means of parity games [8] (solving parity games is known to be equivalent to μ-calculus model checking [12]). Indeed, the model-checking problem for the μ-calculus has been the subject of extensive research (see [10] for an overview and [18–20, 23, 25, 27, 33] for more recent work). The precise complexity of this problem has been open for a long time; it was known to be in UP$\cap$co-UP [19] and PTIME-hard [25].

From a practical perspective, the interesting algorithms are those that have time bounds of the form $n^{O(d)}$, where n is the product of the size of the structure and the length of the formula, and d is the *alternation depth* of the formula, which measures the depth of alternation between least fixpoint and greatest fixpoint operators. A naive algorithm would have d as the exponent, since alternating fixpoints of depth d yield nested loops of depth d, each of which involves n iterations. This naive algorithm uses space $O(dn)$ [13]. The alternation depth is interesting as a measure of syntactic complexity, since, on one hand, many logics can be expressed in low-alternation-depth fragments of the μ-calculus [13, 12], and, on the other hand, the μ-calculus alternation hierarchy is strict [4]. As noted, the naive algorithm can be naturally implemented in a symbolic fashion, operating on sets of states.

The first improvement to the naive approach was presented in [27] (and slightly improved in [33]), who got the exponent down to $d/2$ at the cost of exponential worst-case space complexity. It was then shown By Jurdzinski [20] how to obtained the improved exponent together with the $O(dn)$ space bound. Common to these algorithms is the elimination of alternating fixpoints; they use monotone fixpoint computation that simulates the effects of alternating fixpoints by means of so-called *progress measures*. Progress measures are functions that measure the progress of a computation; see [21, 30, 24] for other applications. While the improved algorithms have better time complexity, they sacrifice the main practical attraction of μ-calculus – these algorithms are enumerative and no symbolic version of them is known.

It is well known that modal logic can be viewed as a fragment of first-order logic [2]. Thus, the μ-calculus can be viewed as a fragment of *first-order fixpoint logic*, often

referred to as "fixpoint logic", which is the extension of first-order logic with least and greatest fixpoint operators. Fixpoint logic has been the subject of extensive research in the context of database theory [1] and finite-model theory [9]. One of the most fundamental theorems in the theory of fixpoint logic is the *Collapse Theorem*, which asserts that, unlike the case for the μ-calculus, the fixpoint alternation hierarchy over finite structures collapses at its first level; that is, every formula in fixpoint logic can be expressed as a least-fixpoint formula [15, 17, 26]. The key to this collapse is the simulation of the effect of alternating fixpoints by means of so-called *stage functions*, which measure the progress of fixpoint computations.

Our main result in this paper is the unification of these two disparate lines of research. We show that the Collapse Theorem of fixpoint logic can be adapted to the μ-calculus. Both progress measure and stage functions measure the progress of fixpoint computations. The key difference between fixpoint logic and the μ-calculus is that while in fixpoint logic progress measures can be constructed *within* the logic (by means of the Stage-Comparison Theorem [29]), this cannot be done in the μ-calculus [4], since it allows fixpoint operators only on unary predicates. In order to simulate the construction of progress measures *within* the μ-calculus, we define the $\mu^{\#}$-calculus. While in the μ-calculus variables represent characteristic functions, i.e., functions from the state space to $\{0, 1\}$, in the $\mu^{\#}$-calculus variables represent *measure functions*, which are functions from the state space to some measure domain. We then prove a *Measured-Collapse Theorem*: every formula in the μ-calculus is equivalent to a least-fixpoint formula in the $\mu^{\#}$-calculus.

We then show that the Measured-Collapse Theorem provides a logical recasting of the improved algorithm in [20]. By starting with a μ-calculus formula of alternation depth d, collapsing it to a least-fixpoint $\mu^{\#}$-calculus formula with measure domain $\{0, \ldots, n^{d/2}\}$, and then computing the least fixpoint, we get the improved exponent of $d/2$ together with the $O(dn)$ space bound. Furthermore, this logical recasting of the algorithm suggests how it can be implemented symbolically. A symbolic evaluation of μ-calculus formulas uses Binary Decision Diagrams [5] to represent characteristic functions [6]. For the $\mu^{\#}$-calculus, we suggest representing measure functions by Algebraic Decision Diagrams, which extend Binary Decision Diagrams by allowing arbitrary numerical domains [7]. Thus, we describe, for the first time, a symbolic μ-calculus model-check algorithm whose complexity matches the one of the best known enumerative algorithm. In fact as detailed in Section 4, working with $\mu^{\#}$-calculus enables us to decrease the bound of the number of iterations needed for the simultaneous calculations, leading to a slightly better complexity.

2 Preliminaries

The *μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [22]. We consider a μ-calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operators $\Diamond$ ("exists next") and $\Box$ ("for all next"), as well as least (μ) and greatest (ν) fixpoint operators. We assume that μ-calculus formulas are written in positive normal form (negation only applied to atomic propositions).

Formally, let AP be a set of atomic propositions and let X be a set of variables. The set of μ-calculus formulas over AP and X is defined by induction as follows. (1) If $p \in AP$, then p and $\neg p$ are μ-calculus formulas. (2) If $x \in X$, then x is a μ-calculus formula (in which x is free). (3) If φ, ψ, are μ-calculus formulas, then $\varphi \vee \psi$, $\varphi \wedge \psi$,$\Diamond\varphi$, and $\Box\varphi$ are μ-calculus formulas, (4) If $x \in X$, then $\mu x.\varphi$ and $\nu x.\varphi$ are μ-calculus formulas (in which x is bound). The semantic of μ-calculus is defined with respect to a Kripke structure $M = \langle S, R, L\rangle$, and an assignment $f : X \rightarrow 2^S$ to the variables. Let $\mathcal{F}$ denote the set of all assignments. For an assignment $f \in \mathcal{F}$, a variable $x \in X$, and a set $S' \subseteq S$, we use $f|_{x=S'}$ to denote the assignment in which x is assigned S' and all other variables assigned as in f. A formula ψ is interpreted as a function $\psi^M : \mathcal{F} \rightarrow 2^S$. Thus, given an assignment $f \in \mathcal{F}$, the formula ψ defines a subset of states that satisfy ψ with respect to this assignment. For a definition of the function ψ^M see the full version or [22]. When M is clear from the context, we omit it). A formula with no free variables is called a *sentence*. Note that the assignment f is required only for the valuation of the free variables in ψ. In particular, no assignment is required for sentences. For a sentence ψ, we say that $M, s \models \psi$ if $s \in \psi^M(f)$, for (the arbitrarily chosen) f with $f(x) = \emptyset$ for all $x \in X$.

Let λ denote μ or ν. We assume that every variable $x \in X$ is bound at most once. We refer to the fixpoint subformula in which x is bound as $\lambda(x)$. If $\lambda = \mu$, we say that x is a μ-variable, and if $\lambda = \nu$, we say that it is a ν-variable. Consider a μ-calculus formula of the form $\lambda x.\varphi$. Given an assignment $f \in \mathcal{F}$, we define a sequence of functions $\varphi^j(f) : 2^S \rightarrow 2^S$ inductively as follows. $\varphi^0(f)(S') = S'$ and $\varphi^{j+1}(f)(S') = \varphi(f|_{x=\varphi^j(f)(S')})$. For a μ-calculus formula ψ and a subformula $\varphi = \lambda x.\lambda(x)$ of ψ, we define the *alternation level* of φ in ψ, denoted $al_\psi(\varphi)$, as follows [3]. If φ is a sentence, then $al_\psi(\varphi) = 1$. Otherwise, let $\xi = \lambda' y.\xi'$ be the innermost μ or ν subformula of ψ that has φ as a subformula, and y is free in φ. Then if $\lambda' \neq \lambda$, we have $al_\psi(\varphi) = al_\psi(\xi) + 1$. Otherwise, $al_\psi(\varphi) = al_\psi(\xi)$.

Intuitively, the alternation level of φ in ψ is the number of alternating fixpoint operators we have to "wrap φ with" in order to reach a sub-sentence of ψ. For a variable x, the alternation level of x, denoted $al(x)$ is the alternation level of $\lambda(x)$. Note that it may be that $\lambda(x)$ is a subformula of $\lambda(x')$ and $al(x) = al(x')$. The definition of $al(x)$ partitions X into equivalence classes according to the variable's alternation level. Note that an equivalent class may contain variables that are independent. In order to refine the class further, we define the order $\prec$ to be the minimal relation that satisfies the following. (1) If x' is free in $\lambda(x)$ then $x \prec x'$. (2) If $x \prec y$ and $y \prec x'$ then $x \prec x'$. We define the $\approx$ equivalence relation to be the minimal equivalence relation that contains all pairs (x, x') such that $x \prec x'$ and $al(x) = al(x')$. The relation $\approx$ refines the partition induce by $al(x)$ so that each class contains variables at the same alternation level that do depend on each other and are all are either μ variables or ν variables. We define the width $width(i)$ of an alternation level i as the maximal size of an equivalence class that is contained in the i'th alternation level. Another property of the $\approx$ relation is that for every equivalence class X^e there exists a unique variable $x_m = max(X^e)$ in X^e such that for every other variable $x \in X^e$ we have $x \prec x_m$. We can simultaneously calculate the fixpoint values of all the variables that are in the same equivalence class.

The reason that we use simultaneous fixpoint is that the evaluation of the variables of a μ-calculus formula as defined above is hierarchical, in the sense that in order to update the value of a variable x, we first evaluate all the variables that appear in subformulas of $\lambda(x)$. Since the value of x might be updated up to $|S|$ times, this makes the complexity of the evaluation exponential in the nesting depth of the fixpoint operators. It turns out that this hierarchal computation is needed only when there is alternation of μ and ν variables. Thus, if $\lambda(x)$ is a subformula of $\lambda(y)$ but $x \approx y$, we can compute their value simultaneously. This could reduce the complexity substantially.

Next, we define a simultaneous fixpoint operation over equivalence classes organized in tuples. Let X^e be an equivalence class of variables with respect to $\approx$. Let X' be the set of variables $\{x' | \exists x \in X^e . x \prec x'\}$, and let X'' be the set $\{x'' | \exists x \in X^e . x'' \prec x\}$. Let $x_m = max(X^e)$, then the subformula $\lambda(x_m) = \lambda x_m . \varphi_m$ binds all variables of X^e. Given an assignment $f : X' \rightarrow 2^S$ we consider $\varphi_m(f)$ as a function $\varphi_m(f) : (X^e \rightarrow 2^s) \rightarrow (X^e \rightarrow 2^S)$. This function is used to define the simultaneous fixpoint value of X^e. Note, that all the variables in φ_m are either in X'' or in $X' \cup X^e$. Given an assignment $f : X' \rightarrow 2^S$, assume that an extension of f to $f|_{X^e = \overline{S'}}$ recursively determines the values of the variables in X'' or more precisely the values of the subformulas $\lambda(x'')$. Thus subformulas that are not determined in φ_m are of the the form $\lambda(x')$ where $x' \in X' \cup X^e$. We determine these values using $f|_{X^e = \overline{S'}}$, then for every variable $x \in X^e$ we can calculate the value of φ_x and determine it's new value. We define the simultaneous fixpoint value of X^e as, $\bigcap\{\overline{S'} : \varphi_m(f)(\overline{S'}) \subseteq \overline{S'}\}$ for μ-class and $\bigcup\{\overline{S'} : \overline{S'} \subseteq \varphi_m(f)(\overline{S'})\}$ for ν-class.

Theorem 1. *For every variable x, the μ-calculus and the simultaneous fixpoint assign the same value to x,*

Theorem 2. (Extended Knaster-Tarski)

- $\bigcap\{\overline{S'} | \varphi_m(f)(\overline{S'}) \subseteq \overline{S'}\} = \bigcap\{\overline{S'} | \overline{S'} = \varphi_m(f)(\overline{S'})\} = \bigcup_{i \geq 0} {\varphi_m}^i(f)(\langle \emptyset, \emptyset, \ldots, \emptyset \rangle) = {\varphi_m}^{|S| \cdot |X^e|}(f)(\langle \emptyset, \emptyset, \ldots, \emptyset \rangle)$.
- $\bigcup\{\overline{S'} | \overline{S'} \subseteq \varphi_m(f)(\overline{S'})\} = \bigcup\{\overline{S'} | \overline{S'} = \varphi_m(f)(\overline{S'})\} = \bigcap_{i \geq 0} {\varphi_m}^i(f)(\langle S, S, \ldots S \rangle) = {\varphi_m}^{|S| \cdot |X^e|}(f)(\langle S, S, \ldots S \rangle)$.

3 The Logic $\mu^\#$-calculus

While a formula of the μ-calculus defines a subset of S, namely a mapping from S to $\{0, 1\}$, a formula of the $\mu^\#$-calculus defines a mapping from S to a domain D where D is parameterized by a natural number k and a sequence of natural numbers $n_0, n_1, \ldots n_k$ such that $D = \bigcup_{l=0}^{k} (\{1, 2, \ldots, n_0\} \times \{1, 2, \ldots, n_1\} \times \ldots \times \{1, 2, \ldots, n_l\}) \cup \{\infty, -\infty\}$. We start with the syntax of the $\mu^\#$-calculus. As in the μ-calculus, formulas are defined with respect to a set AP of atomic and a set X of variables. In the $\mu^\#$-calculus, however, each variable is associated with an arity. We write $x^{(c)}$ to indicate that variable x has arity c. Given AP and X, the set of the μ-calculus formulas (in positive normal form) over AP and X is defined by induction as follows.

- If $p \in AP$, then p and $\neg p$ are $\mu^\#$-calculus formulas.
- If $x^{(c)} \in X$, then $x^{(c)}$ is a $\mu^\#$-calculus formula (in which x is free).

– If φ and ψ are μ-calculus formulas then
  - $\varphi \vee \psi$ and $\varphi \wedge \psi$ are $\mu^{\#}$-calculus formulas,
  - $\Diamond\varphi$ and $\Box\varphi$ are $\mu^{\#}$-calculus formulas,
  - For $x^{(c)} \in X$, we have that $\texttt{set}x^{(c)}.\varphi$ and $\texttt{inc}x^{(c)}.\varphi$ are $\mu^{\#}$-calculus formula (in which x is bound).

We define an alternation level, a preorder $\prec$, and an equivalence relation $\approx$ over X in the same way we define it for the μ-calculus. We say that a $\mu^{\#}$-calculus formula is well formed if

– The arity c of a `set`-variable $x^{(c)}$ is equal to the minimal arity of `inc`-variables with alternation level smaller than $al(x)$.
– The arity c of a `inc`-variable $x^{(c)}$ is equal to the minimal arity of `set`-variables with alternation level smaller than $al(x)$, minus one.

We use $sub(\psi)$ to denote all the subformulas of ψ. Before defining the semantics of the $\mu^{\#}$-calculus, we define a parameterized order over the tuples in D. Intuitively, the order is lexicographic, and the parameter enables us to restrict attention to prefixes of the tuples. Formally, we have the following.

Definition 1. *For $d, d' \in D$ and $l \geq 0$, we say that $d <_l d'$ if either $d' = \infty$ and $d \neq \infty$, or $d' \neq -\infty$ and $d = -\infty$ or $d = (d_0, \ldots, d_i)$ and $d' = (d'_0, \ldots, d'_j)$, and either:*

– *For some $k \leq \min(i, j, l)$ we have $d_k < d'_k$ and for every $0 \leq m < k$, $d_m = d'_m$.*
– *$i < \min(l, j)$ and for every $k \leq i$ we have $d_k = d'_k$.*

Definition 2. *For $d, d' \in D$ and $l \geq 0$, we say that $d =_l d'$ if either $d = d'$ or $d = (d_0, \ldots, d_i)$ and $d' = (d'_0, \ldots, d'_j)$, and $l \leq \min(i, j)$ and for every $k \leq l$ we have $d_k = d'_k$.*

Note that $<_l$ is a total order over the tuples with arity $\leq l$. We sometimes use the order without the parameter, with the usual lexicographic interpretation. Thus, $d < d'$ if $d <_l d'$ for $l = \max\{|d|, |d'|\}$, and the minimum and maximum tuple of a set of tuples are defined similarly.

For $d = (d_0, \ldots, d_i)$ and $l \geq 0$, let $\texttt{set}_l(d)$ be greatest l-tuple d' such that $d' \leq_l d$. If $d = \infty$ or $d = -\infty$, then $set_l(d) = d$. Also, let $\texttt{inc}_l(d)$ to be the smallest l-tuple d' in D such that $d <_l d'$. Since $<_l$ is total, such a unique tuple exists. If $d = (n_0, n_1, \ldots, n_l)$, then $\texttt{inc}_l(d) = \infty$, if d is ∞ then $\texttt{inc}_l(d) = d$, and if d is $-\infty$ then $\texttt{inc}_l(d)$ is the l-tuple $(1, 1, \ldots, 1)$.

Consider a Kripke structure $M = \langle S, R, L\rangle$. A *measure function* for M is a function $g : S \to D$. For $c \geq 1$, we say that g is a measure function of arity c if for all $s \in S$, we have $g(s)$ is either a c-tuple in D or an element of $\{\infty, -\infty\}$. The semantics of $\mu^{\#}$-calculus is defined with respect to a Kripke structure $M = \langle S, R, L\rangle$ and an assignment $f : X \to D^S$ to the variables. An assignment f is legal if for all $x^{(c)} \in X$, the measure function $f(x)$ is of arity c. Let $\mathcal{F}^{\#}$ denote the set of all legal assignments. A formula ψ is interpreted as a function $\psi^M : \mathcal{F}^{\#} \to D^S$. Thus, given a legal assignment $f \in \mathcal{F}^{\#}$, the formula ψ defines a measure function for M with respect to f. The function ψ^M is defined, for all $s \in S$, inductively as follows (when M is clear from the context, we omit it).

- $p(f)(s) = \infty$ if $p \in L(s)$ and $p(f)(s) = -\infty$ if $p \notin L(s)$.
- $\neg p(f)(s) = \infty$ if $p \notin L(s)$ and $p(f)(s) = -\infty$ if $p \in L(s)$.
- For a free variable $x^{(c)}$, we have $x^{(c)}(f)(s) = f(x^{(c)})(s)$.
- $(\varphi \vee \psi)(f)(s) = \max\{\varphi(f)(s), \psi(f)(s)\}$.
- $(\varphi \wedge \psi)(f)(s) = \min\{\varphi(f)(s), \psi(f)(s)\}$.
- $(\Diamond\varphi)(f)(s) = \max\{\varphi(f)(s') \mid R(s,s')\}$.
- $(\Box\varphi)(f) = \min\{\varphi(f)(s') \mid R(s,s')\}$.
- $\mathtt{set}x^{(c)}.\varphi(f)(s) = \mathtt{set}_c(\varphi(f)(s))$.
- $\mathtt{inc}x^{(c)}.\varphi(f)(s) = \mathtt{inc}_c(\varphi(f)(s))$.

Let λ denote `set` or `inc`. As in the μ-calculus, we assume that every variable $x^{(c)} \in X$ is bound at most once in a $\mu^{\#}$-calculus formula, and refer to the subformula that bounds $x^{(c)}$ as $\lambda(x)$. We can view a formula as a function $\psi : \mathcal{F}^{\#} \to \mathcal{F}^{\#}$. Indeed, given $f \in \mathcal{F}^{\#}$, all the subformulas of ψ, and in particular $\lambda(x)$, for all $x^{(c)} \in X$, are mapped into measure functions. Formulas of $\mu^{\#}$-calculus are monotone, in the sense that $\varphi(f) \geq f$. Hence, we can talk about the least fixpoint of a $\mu^{\#}$-calculus formula.

Let $g_\psi : X \to D^S$ be the result of applying ψ on the assignment g_0 until a fixpoint is reached, where g_0 assigns to every variable $x^{(c)}$, the assignment $S \to -\infty$. Every variable $x^{(c)}$ can be updated at most $|S| \cdot n_0 \cdot n_1 \cdot \ldots \cdot n_k$ times thus the time complexity is $O(|X| \cdot |S| \cdot n_0 \cdot n_1 \cdot \ldots \cdot n_k)$ and the space complexity is $O(|X| \cdot |S| \cdot (log(n_0) + log(n_1) + \ldots + log(n_k)))$.

Given a μ-calculus formula ψ, we associate with ψ a $\mu^{\#}$-calculus formula $\psi^{\#}$ that characterizes the same set of states. We define $\psi^{\#}$ to be ψ where the arity of a variable x is $w(x) = \lceil \frac{al(x)}{2} \rceil$, every μ operator is replaced by a `set` operator, and every ν operator is replaced by an `inc` operator. In order to check whether a Kripke structure M satisfies $\psi^{\#}$, we define the domain D where $k = \lceil \frac{max_{x \in X}(al(x))}{2} \rceil$ and for every $0 \leq i \leq k$ we have $n_i = width(2 \cdot i + 1) \cdot |S|$.

Theorem 3. (Measured Collapse) *Let ψ be a μ-calculus formula, and let M be a Kripke structure. Then, $M, s \models \psi$ iff $g_{\psi^{\#}}(\psi^{\#})(s) = \infty$.*

The proof of Theorem 3 is described in the full version. Theorem 3 implies a simple model-checking algorithm for the μ-calculus. Given a μ-calculus formula ψ and a Kripke structure M, translate ψ into $\psi^{\#}$ and check whether $M \models \psi^{\#}$. The time complexity of this algorithm is $O(|X| \cdot width(1) \cdot width(3) \cdot \ldots \cdot width(2 \cdot k + 1) \cdot |S|^{k+1})$ where k is the maximum alternation level of ψ. The space complexity is $O(|X| \cdot |S| \cdot (log(width(1)) + log(width(2)) + \ldots + log(width(k))))$. Note that in the model-checking algorithm that uses a reduction to parity games, the time complexity is $O(|X| \cdot |al(1)| \cdot |al(3)| \cdot \ldots \cdot |al(2 \cdot k + 1)| \cdot |S|^{k+1})$.

Recall that for all i, we have that $width(i) \leq al(i)$. Thus, our complexity is better. The improved complexity follows from the fact that the reduction of μ-calculus model checking to parity games does not take into account the fact that some variables with the same alternation level may be independent of each other. On the other hand, the translation to $\mu^{\#}$-calculus refines the partition induced by the alternating level to the relation $\approx$.

4 Symbolic $\mu^{\#}$-calculus Model Checking and Parity Games

As discussed in Section 1, the improved algorithms for μ-calculus model checking are not symbolic. In this section we describe a symbolic algorithm for $\mu^{\#}$-calculus model checking. The Measured Collapse Theorem then implies a symbolic algorithm for μ-calculus model checking, and our complexity matches the improved complexity of [20]. In addition, we show how the algorithm in [20], for the equivalent problem of solving parity games, can be viewed as a computation of a least fixed-point over a measured domain, and describe a symbolic implementation for it that follows from this view. A symbolic evaluation of μ-calculus formulas uses Binary Decision Diagrams (BDDs) [5] to represent characteristic functions [6]. For the $\mu^{\#}$-calculus, we use Algebraic Decision Diagrams (ADDs), which extend BDDs by allowing arbitrary numerical domains [7].

Symbolic evaluation of $\mu^{\#}$-calculus formulas Consider a $\mu^{\#}$-calculus formula ψ and a Kripke structure $M = \langle S, R, L\rangle$. We define the product of ψ and M as the graph $G_{\psi,M} = \langle V, E\rangle$, where

- $V = sub(\psi) \times S$.
- $E((\varphi, s), (\varphi', s'))$ iff one of the following holds.
  - $s = s'$ and there is φ'' such that φ is $\varphi' \vee \varphi''$, $\varphi'' \vee \varphi'$, $\varphi' \wedge \varphi''$, or $\varphi'' \wedge \varphi'$.
  - $R(s, s')$ and φ is $\Diamond\varphi'$ or $\Box\varphi'$.
  - $s = s'$ and φ is $\mathtt{set}x^{(c)}.\varphi'$ or $\mathtt{inc}x^{(c)}.\varphi'$.
  - $s = s'$, $\varphi = x^{(c)}$, and $\varphi' = \lambda x^{(c)}.\varphi''$.

We refer to vertices of the form $(\varphi' \vee \varphi'', s)$ or $(\Diamond\varphi', s)$ as *max vertices*, and to vertices of the form $(\varphi' \wedge \varphi'', s)$ or $(\Box\varphi', s)$ as *min vertices*.

Let $g_\psi : sub(\psi) \to D^S$ be the least fixpoint of ψ. We describe the calculation of g_ψ by means of a function $f_{\psi,M} : V \to D$ such that $f_{\psi,M}(\varphi, s) = g_\psi(\varphi)(s)$. Note that for all $\varphi \in sub(\psi)$, we have that $s \models \varphi$ iff $f_{\psi,M}(s, \varphi) = \infty$. In order to calculate $f_{\psi,M}$, we describe a sequence of functions $f_0, f_1, \ldots$ such that $f_{\psi,M} = f_i$ where i is the least such that $f_i = f_{i+1}$. The functions $f_i : V \to D$ are defined inductively as follows. We start with f_0.

- If $v = (p, s)$ then $f_0(v) = \infty$ if $s \models p$ and $f_0(v) = -\infty$ if $s \not\models p$.
- If $v = (\neg p, s)$ then $f_0(v) = \infty$ if $s \not\models q$ and $f_0(v) = -\infty$ if $s \models q$.
- For all other vertices $f_0(v) = -\infty$.

Given f_i, we define f_{i+1} as follows.

- If v is of the form (p, s) or $(\neg p, s)$ then $f_{i+1}(v) = f_i(v)$.
- If v is a max vertex, then $f_{i+1}(v) = \max\{f_i(v') \mid (v, v') \in E\}$.
- If v is a min vertex, then $f_{i+1}(v) = \min\{f_i(v') \mid (v, v') \in E\}$.
- If v is of the form $(x^{(c)}, s)$ then v has a single successor v' and $f_{i+1}(v) = f_i(v')$.
- If v is of the form $(\mathtt{set}\ x^{(c)}.\varphi, s)$, then v has a single successor v' and $f_{i+1}(v) = \mathtt{set}_c(f_i(v'))$.
- If v is of the form $(\mathtt{inc}\ x^{(c)}.\varphi, s)$, then v has a single successor v' and $f_{i+1}(v) = \mathtt{inc}_c(f_i(v'))$.

Proposition 1. *Consider a Kripke structure M and $\mu^{\#}$-calculus formula ψ. For all $\varphi \in sub(\psi)$ and $s \in S$, we have $g_\psi(\varphi)(s) = f_{\psi,M}(\varphi, s)$.*

We now describe how to compute $f_{\psi,M}$ symbolically. We use BDDs to represent sets and relations, and use ADDs to represent measure functions. Consider a Kripke structure $M = \langle S, R, L\rangle$ and a formula ψ. Let $G_{\psi,M} = \langle V, E\rangle$ be their product as defined above. We assume that M is given symbolically by one BDD h_R for R, and $|AP|$ BDDs – one BDD h_p for each $p \in AP$, representing the set of states that satisfy p (when the state space is given by truth assignments to AP, there is no need for these BDDs) . Given these BDDs, constructing BDDs that represent V and E is straightforward. In particular, we assume that E is represented by the BDD h_E, and we also have the following BDDs for subsets of V: a BDD h_{AP} for vertices of the form (p, s) or $(\neg p, s)$, BDDs $h_{\max}$ and $h_{\min}$ for the max and min vertices, respectively, a BDD h_X for vertices of the form $(x^{(c)}, s)$ for some c, BDDs $h_{\mathtt{set},j}$ for vertices of the form $(\mathtt{set}x^{(j)}.\varphi, s)$, and BDDs $h_{\mathtt{inc},j}$ for vertices of the form $(\mathtt{inc}x^{(j)}.\varphi, s)$. Finally, the procedure also gets an integer c_{max}, which is the maximal arity of a variable in X.

The algorithm for computing $f_{\psi,M}$ is described in Figure 1. Apart from the Boolean BDD operators OR, AND, and NOT, we use the operator $\rightarrow (h, d)$, which gets a BDD $h \subseteq V$ and some $d \in D$, and creates an ADD that maps all the elements of h to d, and the following procedures.

- `MAX`, which given an ADD $f : V \rightarrow D$ and the BDD h_E, returns an ADD that assigns to every vertex $v \in h_{max}$ the value $\max\{f(v')|E(v, v')\}$.
- `MIN`, which given an ADD $f : V \rightarrow D$ and the BDD h_E, returns an ADD that assigns to every vertex $v \in h_{min}$ the value $\min\{f(v')|E(v, v')\}$.
- `ASSIGN`, which given an ADD $f : V \rightarrow D$ and the BDD h_E, returns an ADD that assigns to every vertex $v \in h_X$ the value $f(v')$ for the single v' with $E(v, v')$.
- `SET(f,j)`, which given an ADD $f : V \rightarrow D$, the BDD h_E, and $1 \leq j \leq c_{max}$, returns an ADD that assigns to every vertex $v \in h_{set,j}$ the value $\mathtt{set}_j(f(v'))$ for the single v' with $E(v, v')$.
- `INC(f,j)`, which given an ADD $f : V \rightarrow D$, the BDD h_E, and $1 \leq j \leq c_{max}$, returns an ADD that assigns to every vertex $v \in h_{inc,j}$ the value $\mathtt{inc}_j(f(v'))$ for the single v' with $E(v, v')$.
- `OR` between ADDs, which gets ADDs that map disjoint subsets of V to D and returns their union (all the ADDs are defined for all the vertices in V, but some vertices are mapped to some special value, which enables us to represent by ADDs also partial functions).

Since all procedures assign values to the vertices according their successors, it is useful to generate, given an ADD f and the BDD h_E, the ADD $f_{suc} : V \times V \rightarrow D$ such that $f_{suc}(v, v') = d$ if $E(v, v')$ and $f(v') = d$. If $\neg E(v, v')$, then $f_{suc}(v, v') = \infty$. The ADD f_{suc} is simply the result of an AND operation on h_E and a ADD of f with renamed variables. Using f_{suc}, the implementation of `ASSIGN` is straightforward as $\exists v'.(f_{suc}$ AND $h_x)$. The implementation of `INC` and `SET` is similar except that we replace every leaf d in the ADD of $(f_{suc}$ AND $h_{\mathtt{inc},j})$ or $(f_{suc}$ AND $h_{\mathtt{set},j})$ with $\mathtt{inc}_j(d)$ or $\mathtt{set}_j(d)$ respectively. The procedures `MAX` and `MIN` are more complicated and are described in the full version.

```
MODEL_CHECK
 h_AP^T = (OR_{p∈AP}({p} AND h_p))OR (OR_{p∈AP}({¬p} AND NOT (h_p));
 f_q^T =→ (h_q^T, ∞) ;
 f = f_q^T OR → ((h_V AND NOT h_q^t), −∞);
 repeat
     f_old = f;   f_max = MAX(f_old) ;
     f_min = MIN(f_old) ;   f_x = ASSIGN(f_old) ;
     f_set = false;   f_inc = false;
     for j = 1 to c_max do
         f_set = f_set OR SET(f_old, j);   f_inc = f_inc OR INC(f_old, j)
     f = f_q OR f_max OR f_min OR f_set OR f_inc;
 until f = f_old
```

Fig. 1. The symbolic algorithm for $\mu^{\#}$-calculus model checking.

Let us now analyze the complexity of the procedure. The number of iterations required for the procedure to reach a fixed point is bounded by $|D| \cdot |V|$ which is $|S|^{\lceil \frac{al(\psi)}{2} \rceil} \cdot |S| \cdot |\psi|$. Each iteration involves an applications of the MIN/MAX procedures (that are the most costly). In the full version, we show that these procedures apply at most $|V|^2 \cdot log(|V|) = (|S| \cdot |\psi|)^2 \cdot log((|S| \cdot |\psi|)$ ADD operations. Thus, the overall complexity is $O(|S|^{\lceil \frac{al(\psi)}{2} \rceil + 3} \cdot |\psi|^3) \cdot log((|S| \cdot |\psi|)$ ADD operations.

Parity games A parity game is played on a graph $\langle V, E \rangle$, where V is partitioned into two sets: V_0 of even vertices and V_1 of odd vertices. Every vertex v has a priority $p(v) \in \{0, 1, \ldots k-1\}$. A parity game over $\langle V_0, V_1, E, p \rangle$ is played by two players, referred to as the odd and the even player. A play over the game starts by putting a pebble at some initial vertex v and proceeds infinitely many rounds. In each round, one of the players moves the pebble on an edge from the current vertex to one of its successors. If the source vertex is in V_0, the even player moves the pebble; otherwise the odd player moves the pebble. The play generates an infinite sequence of vertices ρ. Let $inf(\rho)$ be the set of vertices that appear infinitely often in ρ. The odd player wins the game if the vertex with minimal priority in $\inf(\rho)$ has an odd priority. Otherwise, the even player wins. The problem is to determine, given a game graph $\langle V_0, V_1, E, p \rangle$, the set of vertices from which the odd player has a winning strategy.

In [20], an algorithm for solving parity games is suggested. Below, we describe the algorithm in terms of measure function. Let $D = \bigcup_{j=1}^{\frac{k}{2}} \{0, 1, \ldots, |V|\}^j \cup \{\infty, -\infty\}$, let F be the set of all measure functions $f : V \to D$ and let f_0 be the initial function that assigns $-\infty$ to all vertices. A game graph G induces a function from F to F, where for a measure function $f \in F$, the measure function $G(f)$ is defined, for all $v \in V$, as follows:

$$G(f)(v) = \begin{cases} \max_{(v,v') \in E} f(v') & \text{if } v \in V_1 \text{ and } p(v) \text{ is even.} \\ \max_{(v,v') \in E} \mathtt{inc}_{\lceil \frac{p(v)}{2} \rceil}(f(v')) & \text{if } v \in V_1 \text{ and } p(v) \text{ is odd.} \\ \min_{(v,v') \in E} f(v') & \text{if } v \in V_0 \text{ and } p(v) \text{ is even.} \\ \min_{(v,v') \in E} \mathtt{inc}_{\lceil \frac{p(v)}{2} \rceil}(f(v')) & \text{if } v \in V_0 \text{ and } p(v) \text{ is odd.} \end{cases}$$

If we denote by f_G to the least fixpoint of G, then the set of winning vertices for the odd player is $\{v|f_G(v) = \infty\}$, and the set of winning vertices for the even player is $\{v|f_G(v) < \infty\}$. The measure function f_G can be used for generating a winning strategy $\pi : V_0 \to V$ for the even player where for every $v \in V_0$ we have $\pi(v) = v'$ such that $f_G(v') = min\{f_G(v'')|(v, v'') \in E\}$. Thus, the even player moves to a successor of v with minimal measure. A symbolic procedure that generate a strategy is given in the full version.

A symbolic implementation of the algorithm similar to the symbolic evaluation of $\mu^{\#}$-calculus formulas is described in Figure 2. The procedure calls the following procedures

- MAXe, which given an ADD $f : V \to D$, the BDD h_E, and an even $1 \leq j \leq \frac{k}{2}$, returns an ADD that assigns to every vertex $v \in V_1$ with $p(v) = j$, the value $\max\{f(v')|E(v, v')\}$.
- MAXo, which given an ADD $f : V \to D$, the BDD h_E, and an odd $1 \leq j \leq \frac{k}{2}$, returns an ADD that assigns to every vertex $v \in V_1$ with $p(v) = j$, the value $\max\{\mathtt{inc}_{\lceil \frac{j}{2} \rceil}(f(v')) : E(v, v')\}$.
- MINe and MINo, defined similarly for vertices in V_0.

The symbolic implementation of these procedures is similar to the implementation of the MAX and MIN procedures of the former section, and is described in the full version.

```
PARITY(G)
f =→ (V, −∞);
repeat
   f_old = f;   f = false;
   for j = 1 to k/2 do
      if j is even then f = f OR MAXe(f_old, j) OR MINe(f_old, j);
      if j is odd then f = f OR MAXo(f_old, j) OR MINo(f_old, j);
  end for
until f = f_old;
```

Fig. 2. A symbolic algorithm for solving parity games.

Complexity: Similarly to the previous section, we can bound the number of iterations by $|V|^{\lceil \frac{k}{2} \rceil} \cdot |V|$. Thus, the overall complexity is $O(|V|^{\lceil \frac{k}{2} \rceil + 3} \cdot log(|V|))$ ADD operations.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.
2. J. F. A. K. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1985.
3. G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proc. TACAS*, LNCS 1055, 1996.
4. J.C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *TCS*, 195(2):133–153, 1998.

5. R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.
6. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *I&C*, 98(2):142–170, 1992.
7. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi", Algebraic decision diagrams and their applications. *FMSD*, 10 (2/3): 171–206, 1997
8. A. Chakrabarti, L. de Alfaro, T.A. Henzinger, M. Jurdzinski, and F.Y.C. Mang. Interface compatibility checking for software modules. In *14th CAV*, LNCS 2404, pp. 428–441, 2002.
9. H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, 1995.
10. E.A. Emerson. Modal Checking and the μ-Calculus, *Descriptive Complexity and Finite Models*, American Mathematical Society, pp. 185–214, 1997.
11. E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th ICALP*, pp. 169–181, 1980.
12. E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th CAV*, LNCS 697, pp. 385–396, 1993.
13. E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st LICS*, pp. 267–278, 1986.
14. K. Etessami, Th. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In *Proc. 28th ICALP*, LNCS 2076, pp. 694–707, 2001.
15. Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.
16. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. *I&C*, 173(1):64–81, 2002.
17. N. Immerman. Relational queries computable in polynomial time. *I&C*, 68:86–104, 1986.
18. M. Jurdzinski J. Voge. A discrete strategy improvement algorithm for solving parity games. In E. A. Emerson and A. P. Sistla, editors, *Proc 12th CAV*, LNCS 1855, pp. 202–215, 2000.
19. M. Jurdzinski. Deciding the winner in parity games is in UP $\cap$ co-UP. *IPL*, 68(3):119–124, 1998.
20. M. Jurdzinski. Small progress measures for solving parity games. In *Proc. 17th TACAS*, LNCS 1770, pp. 290–301, 2000.
21. N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. 32nd FOCS*, pp. 358–367, 1991.
22. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.
23. O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th STOC*, pp. 224–233, 1998.
24. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM ToCL*, 2001(2):408–429, 2001.
25. O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.
26. D. Leivant. Inductive definitions over finite structures. *I&C*, 89:95–108, 1990.
27. D. Long, A. Brown, E. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In *Proc. 6th CAV*, LNCS 818, pp. 338–350, 1994.
28. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
29. Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974.
30. Klarlund N and F.B. Schneider. Proving nondeterministically specified safety properties using progress measures. *I&C*, 107(1):151–170, 1993.
31. D. Park. Finiteness is μ-ineffable. *TCS*, 3:173–181, 1976.
32. V.R. Pratt. A decidable μ-calculus: preliminary report. In *22nd FOCS*, pp. 421–427, 1981.
33. H. Seidl. Fast and simple nested fixpoints. *IPL*, 59(6):303–308, 1996.