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 [Koz83], following earlier studies of fixpoint calculi in the theory of program correctness [EC80,Par76,Pra81]. Over the past 20 years, the μ-calculus has been established as essentially the "ultimate"

*** 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.

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* [EL86]. The μ-calculus has gained further prominence with the discovery that its formulas can be evaluated symbolically in a natural way [BCM$^+$92], 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 [McM93]. Beyond the need to use symbolic methods in order to cope with the state-explosion problem [BCM$^+$92], which is acute in concurrent settings, several concurrency-related problems are naturally solved by evaluation of μ-calculus formulas. This includes checks for fair simulation between two components of a concurrent systems [EWS01,HKR02] and reasoning about the interaction between a component and its environment, which is naturally expressed by means of parity games [CdAH$^+$02] (solving parity games is known to be equivalent to μ-calculus model checking [EJS93]). Indeed, the model-checking problem for the μ-calculus has been the subject of extensive research (see [Eme97] for an overview and [JV00,Jur98,Jur00,KV98,KVW00,LBC$^+$94,Sei96] for more recent work). The precise complexity of this problem has been open for a long time; it was known to be in UP$\cap$co-UP [Jur98] and PTIME-hard [KVW00].

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 $O(dn)$ [EL86]. 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 [EL86,EJS93], and, on the other hand, the μ-calculus alternation hierarchy is strict [Bra98]. 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 [LBC$^+$94] (and slightly improved in [Sei96]), who got the exponent down to $d/2$ at the cost of exponential worst-case space complexity. It was then shown By Jurdzinski [Jur00] 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 [Kla91,NS93,KV01] 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 [Ben85]. 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 [AHV95] and finite-model theory [EF95]. 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 [GS86,Imm86,Lei90]. 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 [Mos74]), this cannot be done in the μ-calculus [Bra98], 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 [Jur00]. 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 [Bry86] to represent characteristic functions [BCM^{+}92]. For the $\mu^{\#}$-calculus, we suggest representing measure functions by Algebraic Decision Diagrams, which extend Binary Decision Diagrams by allowing arbitrary numerical domains [BFG^{+}97]. 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.

Although the best known algorithm for μ-calculus model-checking is exponential, its similarity to parity games implies that the result of the model checking can be verified in polynomial time. For parity games the results can be represented as a strategy, which is a polynomial size certification ([KV04]) which can be verified in polynomial time. In Section 5 we prove that the result of the $\mu^{\#}$-calculus model-checking can be used as a certification for μ-calculus model-checking. We then present symbolic al-

gorithms for generating certifications for both μ-calculus model-checking and parity games.

2 Preliminaries

The *μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. 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 $f : X \to 2^S$ 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 $\psi^M : \mathcal{F} \to 2^S$. 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 [Koz83]. 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 $\varphi^j(f) : 2^S \to 2^S$ 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 [BC96]. 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 fol-

lowing. (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 $f : X' \to 2^S$ we consider $\varphi_m(f)$ as a function $\varphi_m(f) : (X^e \to 2^s) \to (X^e \to 2^S)$. 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 $f : X' \to 2^S$, 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 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 $\mathtt{set}x^{(c)}.\varphi$ and $\mathtt{inc}x^{(c)}.\varphi$ 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 $\mathtt{set}_l(d)$ be greatest l-tuple d' such that $d' \leq_l d$. If $d = \infty$ or $d = -\infty$, then $set_l(d) = d$. Also, let $\mathtt{inc}_l(d)$ 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 $\neg p(f)(s) = -\infty$ if $p \in L(s)$.
- For a free variable $x^{(c)}$, we have $x^{(c)}(f)(s) = f(x^{(c)})(s)$.
- $(\varphi \vee \psi)(f)(s) = \max\{\varphi(f)(s), \psi(f)(s)\}$.
- $(\varphi \wedge \psi)(f)(s) = \min\{\varphi(f)(s), \psi(f)(s)\}$.
- $(\Diamond\varphi)(f)(s) = \max\{\varphi(f)(s') \,|R(s, s')\}$.
- $(\Box\, \varphi)(f) = \min\{\varphi(f)(s') \,|R(s, s')\}$.
- $\texttt{set}\ x^{(c)}.\varphi(f)(s) = \texttt{set}_c(\varphi(f)(s))$.
- $\texttt{inc}\ x^{(c)}.\varphi(f)(s) = \texttt{inc}_c(\varphi(f)(s))$.

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 Appendix A. 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 [Jur00]. In addition, we show how the algorithm in [Jur00], 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) [Bry86] to represent characteristic functions [BCM$^+$92]. For the $\mu^{\#}$-calculus, we use Algebraic Decision Diagrams (ADDs), which extend BDDs by allowing arbitrary numerical domains [BFG$^+$97].

4.1 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 $g_\psi: sub(\psi)\rightarrow D^S$ be the least fixpoint of ψ. We describe the calculation of g_ψ by means of a function $f_{\psi,M}:V\rightarrow D$ 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 $f_i:V\rightarrow D$ 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 $s \not\models p$ and $f_0(v) = -\infty$ if $s \models p$.
- 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 $f_{suc} : V \times V \to D$ 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 $h_{\texttt{inc},j})$ or $(f_{suc}$ AND $h_{\texttt{set},j})$ with $\texttt{inc}_j(d)$ or $\texttt{set}_j(d)$ respectively. The procedures MAX and MIN are more complicated and are described in Section 4.3.

```
MODEL_CHECK
 h^T_AP = (OR_{p∈AP}({p} AND h_p))OR (OR_{p∈AP}({¬p} AND NOT (h_p));
 f^T_q =→ (h^T_q, ∞) ;
 f = f^T_q OR → ((h_V AND NOT h^T_q), −∞);
 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 $|S|^{\lceil \frac{width(\psi)}{2} \rceil} \cdot |S| \cdot |\psi|$. Each iteration involves an applications of the MIN/MAX procedures (that are the most costly). In Section 4.3 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 $O(|S|^{\lceil \frac{width(\psi)}{2} \rceil + 3} \cdot |\psi|^3) \cdot log((|S| \cdot |\psi|)$ ADD operations.

4.2 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 [Jur00], 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 Section 5.2.

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 next section.

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.

4.3 Computing the minimal/maximal successor

In this section we address the following problem: given a graph $G = \langle V, E\rangle$, where V and E are given in terms of their BDDs, and a measure function $f : V \to D$,

```
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.

represented as an ADD, construct another ADD that represents the measure function $f_{\min} : V \to D$ with $f_{\min}(v) = \min\{f(v')|(v, v') \in E\}$ (or similarly, $f_{\max}(v) = \max\{f(v')|(v, v') \in E\}$).

Recall that it is easy to construct an ADD for the function $f_{suc} : (V \times V) \to D$ such that $f_{suc}(v, v') = d$ if $E(v, v')$ and $f(v') = d$. In Figure 3 we describe a recursive procedure MIN that receives the ADD f_{suc} and construct the ADD for $f_{\min}$. Let n be an ADD node. We refer to the right and left successors of n as $n.r$ and $n.l$, and refer to the variable that n represents as $n.v$. For a variable v, let $o(v)$ be the position of v in the BDD order (the position of the root is 0). We use $var(V)$ and $var'(V)$ to denote the set of variables that encode V and V', respectively. Since E may be a strict subset of $V \times V'$, the function f_{suc} is not defined for all the pairs in $V \times V'$. Thus, one leaf in the ADD for f_{suc} stands for the value of the pairs for which f_{suc} is undefined, and we assume that this value is ∞. Since every vertex has at least one successor, the ADD $f_{\min}$ is defined for all the vertices of V.

The procedure MIN calls the procedure MERGE, described in Figure 4. The procedure MERGE gets pointers to the roots n_1 and n_2 of two ADDs, representing the functions f_1 and f_2, respectively, both from some set U to D. The procedure merges f_1 and f_2 to an ADD in which every $u \in U$ is mapped to $\min(f_1(u), f_2(u))$. Note that since f_1 and f_2 may be partial, we refer to the value of the undefined leaf as ∞, The

```
MIN(ADD n){
    If (n is a terminal node) then return n;
    If (n.v is in var(V)) then return (n.v AND MIN(n.r)) OR ( NOT n.v AND MIN(n.l));
    If (n.v is in var'(V)) then return MERGE(MIN(n.r),MIN(n.l));
}
```

Fig. 3. The procedure MIN

correctness of the MERGE and MIN procedures follows from Lemmas 1 and 2. The proofs of these lemmas are presented in Appendix B.

Lemma 1. *Let $f : U \to D$ be the function represented by MERGE(n_1, n_2). For every $u \in U$, we have $f(u) = \min(f_1(u), f_2(u))$.*

```
MERGE(ADD n1, ADD n2){
   if (n1 and n2 are terminal nodes) then
      return min(n1, n2);
   if (o(n1.v) < o(n2.v)) then
      return (n1.v AND MERGE(n1.r, n2)) OR ( NOT n1.v AND MERGE(n1.l, n2));
   if (o(n1.v) > o(n2.v)) then
      return (n2.v AND MERGE(n2.r, n1)) OR ( NOT n2.v AND MERGE(n2.r, n1));
   return (n1.v AND MERGE(n1.r, n2.r)) OR ( NOT n1.v AND MERGE(n1.l, n2.l));
}
```

Fig. 4. The procedure MERGE

Lemma 2. *Let $f : U \to D$ be the function represented by MIN(n). For every $u \in U$, we have $f(u) = \min\{d|$ there exists $u' \in U'$ such that $f_{suc}(u, u') = d\}$.*

The number of BDD nodes in the ADD f_{suc} is bound by $|V|^2$. The MIN/MAX procedures requires $O(1)$ ADD operations for each ADD node that encodes V vertices, and requires two MERGE procedures for each ADD node that encodes V' vertices. The procedure MERGE gets an ADD f' and apply $O(1)$ ADD operations for each ADD node in f'.

Thus, the MERGE operations are the costly part of the MIN/MAX procedures. Accordingly, we are interesting in the number of ADD operations that all MARGE calls require. The ADD f_{suc} is defined over $2 \cdot log(|V|)$ variables $log(|V|)$ of them are in $vars'(V)$. We enumerate the levels of f_{suc} from top down starting at 0, then, the i'th level consists of at most 2^i ADD nods, each node represents an ADD of size smaller than $2^{(2 \cdot log(|V|)-i)}$. For each level of the $var'(V)$ variables, the MIN/MAX procedures apply at most 2^i MERGE calls each consists of $O(2^{(2 \cdot log(|V|)-i)})$ ADD operations. This implies that for each level the MIN/MAX procedure applies $O(2^{2 \cdot log(|V|)}) = O(|V|^2)$ operations. Thus, the MIN/MAX procedures involves $O(|V|^2 \cdot log(|V|)))$ ADD operations.

5 Certifying the Results

Recall that the model-checking of μ-calculus problem is in UP∩co-UP. This implies that for every model M and formula ψ there exists a polynomial witness for $M \models \psi$ or for $M \not\models \psi$. Furthermore, these witnesses can be verified in polynomial time. While in parity games it is known that strategies can be considered as certifications, for μ-calculus model-checking there is no clear way to certify its results. In this section we show that the assignment $g_{\psi^\#}$, which is the least fixpoint of $\psi^\#$, is a witness that for every s such that $g_{\psi^\#}(\psi)(s) \neq \infty$, we have $M, s \not\models \psi$. For a witness that prove that $M, s \models \psi$, we construct $g_{\neg\psi^\#}$, and show that $g_{\neg\psi^\#}(\neg\psi)(s) \neq \infty$. We also provide a symbolic algorithm that constructs a strategy in parity games.

5.1 A Certification for μ-Calculus

A certification is required to have a polynomial size, and to be able to be verified in polynomial time. The size of $g_{\psi^\#}$ is linear in $|S|$ and logarithmic in $|D|$. Since $|D| <$

$|S|^{|\psi|}$, the size of $g_{\psi^\#}$ is polynomial in M and ψ. It is left to show that it is possible to verify that a given function g is indeed $g_{\psi^\#}$. We claim that given a function g, it is enough to verify that g is a fixpoint of $\psi^\#$ (not necessarily least) in order to prove that $M, s \not\models \psi$. Theorem 3 implies that for the least fixpoint assignment $g_{\psi^\#}$, we have that $g_{\psi^\#}(\psi)(s) \neq \infty$ iff $M, s \not\models \psi$. Suppose that we are given an assignment $g \in \mathbf{F}^\#$ such that $g(\psi)(s) \neq \infty$. Since $g_{\psi^\#} \leq g$, we have that $g_{\psi^\#}(\psi)(s) \neq \infty$, thus, $M, s \not\models \psi$. Checking that g is a fixpoint $g = \psi^\#(g)$ (not necessarily the least), can be done in time linear in the product of M and $\psi^\#$. Thus we have a linear time authentication procedure.

In Figure 5 we present a symbolic procedure `Authenticate` that verify that a given assignment g is a fixpoint. We use the same BDDs and procedures as in the MODEL_CHECK algorithm. Since the MIN/MAX procedures require at most $(|S| \cdot$

```
Authenticate
 h^T_AP = (OR_{p∈AP}({p} AND h_p))OR (OR_{p∈AP}({¬p} AND NOT (h_p));
 h^F_AP = (OR_{p∈AP}({p} AND NOT h_p))OR (OR_{p∈AP}({¬p} AND (h_p));
 f =→ (h^T_AP, ∞)OR → (h^F_AP, −∞) ;
 f_max = MAX(g) ;
 f_min = MIN(g) ;
 f_x = ASSIGN(g) ;
 f_set = false;
 f_inc = false;
 for j = 1 to c_max do
     f_set = f_set OR SET(g, j);
 f_inc = f_inc OR INC(g, j);
 f = f OR f_x OR f_max OR f_min OR f_set OR f_inc;
 If g == f return true else return false
```

Fig. 5. The symbolic procedure that authenticate that g is a fixpoint of $\psi^\#$.

$|\psi|)^2 \cdot \log(|S| \cdot |\psi|)$ ADD operations, this is also the complexity of the procedure.

5.2 Generating a Winning Strategy for Parity Games

As mentioned earlier the measure function that is computed by the PARITY procedure can be used for generating a winning strategy for the even player. We present a symbolic procedure that constructs a strategy for the even player. The result of this procedure is a BDD $\pi \subseteq V_0 \times V$ such that for every even vertex v with measure smaller than ∞ there exists a transition $(v, v') \in \pi$, where v' is a successor of v with minimal measure. We allow v to have more than one outgoing transition, but they all ended in vertices with the same minimal measure.

The procedure STRATEGY call the procedure Min from the previous section that returns an ADD f_{min} that associates with each vertex of V_0 the minimal measure of it's successors. Then, the STRATEGY procedure constructs the ADD f_{suc} that contains all triples $(v, v') \mapsto n$ such that $(v, v') \in E$ and n is the measure of v'. Finally, it

calls the procedure MinSuc that returns a BDD that contains all pairs (v, v') such that $f_{suc}(v, v') = f_{min}(v)$. This relation associate with each vertex in V_0 its successors that have minimal measure. The correctness of the procedure MinSuc follows from Lemma 3.

```
MinSuc(ADD n_min, ADD n_suc){
   if (n_min and n_suc are terminal nodes) then
      return (n_min = n_suc);
   if (o(n_min.v) < o(n_suc.v)) then
      return (n_min.v AND MinSuc(n_min.r, n_suc)) OR ( NOT n_min.v AND MinSuc(n_min.l, n_suc));
   if (o(n_min.v) > o(n_suc.v)) then
      return (n_suc.v AND MinSuc(n_min, n_suc.r)) OR ( NOT n_suc.v AND MinSuc(n_min, n_suc.l));
   return (n_min.v AND MinSuc(n_min.r, n_suc.r)) OR ( NOT n_min.v AND MinSuc(n_min.l, n_suc.l));
}
```

Fig. 6. The procedure MinSuc

Lemma 3. *Let $\pi : U \times U'$ be the relation resulted from MinSuc(n_{min}, n_{suc}), where n_{suc} is an ADD that represents a function from $U \times U'$ to $\mathbb{N}$, and n_{min} is an ADD that represents a function from U to $\mathbb{N}$. For every $(u, u') \in \pi$, we have $n_{suc}(u, u') = n_{min}(u)$.*

The proof of Lemma 3 is given in Appendix B.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

[BC96] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proc. Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[Ben85] J. F. A. K. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1985.

[BFG+97] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Journal of Formal Methods in Systems Design*, 10(2/3):171–206, 1997.

[Bra96] J. Bradfield. The modal mu-calculus alternation hierarchy is strict. In *International Conference on Concurrency Theory (CONCUR)*, volume 1119 of *LNCS*, pages 233–246, 1996.

[Bra98] J.C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, March 1998.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.

[CdAH+02] A. Chakrabarti, L. de Alfaro, T.A. Henzinger, M. Jurdzinski, and F.Y.C. Mang. Interface compatibility checking for software modules. In *Computer Aided Verification, Proc. 14th International Conference*, volume 2404 of *Lecture Notes in Computer Science*, pages 428–441. Springer-Verlag, 2002.

[EC80] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th InternationalColloq. on Automata, Languages and Programming*, pages 169–181, 1980.

[EF95] H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, 1995.

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th International Conference*, volume 697 of *Lecture Notes in Comptuer Science*, pages 385–396, Elounda, Crete, June 1993. Springer-Verlag.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st Symp. on Logic in Computer Science*, pages 267–278, Cambridge, June 1986.

[Eme97] E.A. Emerson. Model checking and the μ-calculus. In N. Immerman and Ph.G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 185–214. American Mathematical Society, 1997.

[EWS01] K. Etessami, Th. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In *Proc. 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 694–707, 2001.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[HKR02] T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. *Information and Computation*, 173(1):64–81, 2002.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[Jur98] M. Jurdzinski. Deciding the winner in parity games is in up $\cap$ co-up. *Information Processing Letters*, 68(3):119–124, 1998.

[Jur00] M. Jurdzinski. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer-Verlag, 2000.

[JV00] M. Jurdzinski J. Voge. A discrete strategy improvement algorithm for solving parity games. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, Chicago, July 2000. Springer.

[Kla91] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 358–367, San Juan, October 1991.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KV98] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 224–233, Dallas, 1998.

[KV01] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2001(2):408–429, July 2001.

[KV04] O. Kupferman and M.Y. Vardi. From complementation to certification. In *10th International Conference on Tools and algorithms for the construction and analysis of systems*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[LBC+94] D. Long, A. Brown, E. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D. L. Dill, editor, *Computer Aided Verification, Proc. 6th International Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 338–350, Stanford, June 1994. Springer-Verlag, Berlin.

[Lei90] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95–108, 1990.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Mos74] Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974.

[NS93] Klarlund N and F.B. Schneider. Proving nondeterministically specified safety properties using progress measures. *Information and Computation*, 107(1):151–170, 1993.

[Par76] D. Park. Finiteness is μ-ineffable. *Theoretical Computer Science*, 3:173–181, 1976.

[Pra81] V.R. Pratt. A decidable μ-calculus: preliminary report. In *Proc. 22nd IEEE Symp. on Foundation of Computer Science*, pages 421–427, 1981.

[Sei96] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996.

A Proving the Measured Collapse Theorem

In this section we prove the Measured Collapse Theorem. Given a μ-calculus formula ψ, we first show that for every Kripke structure M, we can construct another μ-calculus formula ψ' such that $M \models \psi$ iff $M \models \psi'$. The formula ψ' is the result of unfolding all ν-equivalence classes in ψ in a method similar to the one presented in [Sei96]. We then show that $M \models \psi'$ iff $M \models \psi^{\#}$.

For technical convenience, we assume that ψ is of the form $\mu x.\varphi$, meaning, all variables with odd alternation level are ν variables and all variables with even alternation level are μ variables. This implies that only ν-classes are being unfolded. It is easy to extend our result to arbitrary formulas.

A.1 Unfolding μ-calculus Formulas

We start by showing how to unfold a single variable, then we generalized it to a set of variables. Consider a Kripke structure $M = \langle S, R, L\rangle$. Theorem **??** implies that we can evaluate a formula $\nu x.\varphi$ by applying the function φ on S for $|S|$ times. By unfolding φ, we do the same operation syntactically. Let $\psi = \nu x.\varphi$ be a subformula and let X'' be the set $\{x''|x'' \prec x\}$. Unfolding of ψ with respect to x uses new variables of the form $\langle x, k\rangle$, and it proceeds as follows.

- $\psi^0 =$ **true**.
- $\psi^{k+1} = \varphi[x/\psi^k, x''/\langle x'', k\rangle]$, i.e, every occurrence of x is replaced with ψ^k, and every occurrence of $x'' \in X''$ is replaced with the variable $\langle x'', k\rangle$.

Note that $\psi^{|S|}$ has $|X''||S|$ new variables. Intuitively, the variable $\langle x'', k\rangle$ maintains the assignment to x'' after k iterations of applying φ to S. The definition for unfolding μ variables is similar.

Example 1. Let $\psi = \nu x_1.p \wedge \Box\, x_1 \wedge \mu x_2.[\Box\, x_1 \vee \Diamond x_2 \vee q]$, then $x_2 \prec x_1$, and the result of unfolding ψ with respect to x_1 proceeds as follows.

- $\psi^0 =$ **true**.
- - $\psi^1 = p \wedge \Box$ **true** $\wedge\, \mu\langle x_2, 1\rangle.\varphi_{\langle x_2,1\rangle}$.
  - $\varphi_{\langle x_2,1\rangle} = \Box$ **true** $\vee \Diamond\langle x_2, 1\rangle \vee q$.
- - $\psi^2 = p \wedge \Box\, \psi^1 \wedge \mu\langle x_2, 2\rangle.[\Box\, \psi^1 \vee \Diamond\langle x_2, 2\rangle \vee q] = p \wedge \Box\, [p \wedge \mu(\langle x_2, 1\rangle)] \wedge \mu(\langle x_2, 2\rangle)$.
  - $\varphi_{\langle x_2,2\rangle} = \Box\, [p \wedge \mu(\langle x_2, 1\rangle)] \vee \Diamond\langle x_2, 2\rangle \vee q$.
  - $\varphi_{\langle x_2,1\rangle} = \Box$ **true** $\vee \Diamond\langle x_2, 1\rangle \vee q$.

Theorem **??** implies the following.

Lemma 4. *Given a μ-calculus formula ψ of the form $\nu x.\varphi$, and a Kripke structure $M = \langle S, R, L\rangle$, we have that $M \models \psi$ iff $M \models \psi^{|S|}$.*

Note that Lemma 4 does not imply that for some k, the formula ψ^k is equivalent to ψ. It is possible that for some formula ψ, we have that for every k there exists a structure $M = \langle S, R, L\rangle$, with $|S| > k$ such that $M \models \psi^k$ but $M \not\models \psi$. In fact [Bra96] shows

that for every k there exists a μ-calculus formula ψ of alternation depth k such that the alternation depth of every formula φ that is equivalent to ψ, is at least k. (the alternation hierarchy is strict).

Let $\nu x.\varphi$ be a subformula of ψ then unfolding x in ψ is defined as $\psi[\nu x.\varphi/\varphi^{|S|}]$, i.e., replacing the occurrence of $\nu x.\varphi$ by $\varphi^{|S|}$. In [Sei96] the following algorithm is suggested: given a μ-calculus formula ψ and a structure M, remove all ν operators in ψ, then, simultaneously calculate the value of ψ. Note that after removing all ν operators, the alternation level of all the variables that remain is 1, and thus, they can be calculated simultaneously.

Auxiliary Variables Note that in the example above, we have two occurrences of $\mu(\langle x_2, 1\rangle)$. This happens because we use two different copies of ψ^1 in ψ^2. We could solve this problem using more indices, but this would create a redundancy, because obviously both copies of ψ^1 get the same value. Instead, we use auxiliary variables that enable as to have several copies of the same expression in a formula without the need to evaluate this expression more than once. Formally, if we want to refer to a subformula θ of a formula φ, we write $z = \theta$ and replace the occurrences of θ in φ by z.

Here, we use auxiliary variables to represent the formulas ψ^j that are obtained by removing ν-variables. When we remove a ν-variable x, we replace ψ^j with an auxiliary variable $\langle z, j\rangle$. Let ψ be a μ-calculus formula. Let x be a ν-variable, and let X'' the set $\{x''|x'' \prec x\}$. For $k > 0$, we write the k-th unfolding of ψ with respect to x as follows. The k-th subformula is made of k subformulas for each $x'' \in X''$ (using indexed variables), and $k + 1$ auxiliary subformulas for the auxiliary variables $\langle z, l\rangle$. We define the subformulas as follows.

- For $x'' \in X''$ we write k subformulas. We index the variables with indices $1 \leq l \leq k$, where the subformula of $\langle x'', l\rangle$ is, $\varphi_{x''}[x/\langle z, l-1\rangle, y''/\langle y'', l\rangle]$, i.e., every occurrence of x is replaced by $\langle z, l-1\rangle$, and every occurrence of $y'' \in X''$ is replaced by the variable $\langle y'', l\rangle$.
- For x we write $k + 1$ subformulas where $\langle z, 0\rangle =$ **true** and for $1 \leq l \leq k$ we write $\langle z, l\rangle = \varphi_x[x/\langle z, l-1\rangle, y''/\langle y'', l\rangle]$, i.e., every occurrence of x is replaced by $\langle z, l-1\rangle$, every occurrence of $y'' \in X''$ is replaced by the variable $\langle y'', l\rangle$.
- In the subformulas for the variables that are in X', we replace the occurrence of $\lambda(x)$ by $\langle z, k\rangle$.

Finally, $\langle z, k\rangle$ represents φ^k.

Example 2. Let $\psi = \nu x_1.p \wedge x_1 \wedge \mu x_2.[x_1 \vee x_2 \vee x_4 \vee \nu x_3.[q \wedge x_2 \wedge x_3 \wedge x_1]]$, and assume $|S| = 2$. We first unfold x_1:

- $\langle z_1, 0\rangle =$ **true**
- $\varphi_{\langle x_3,1\rangle} = q \wedge \langle x_2, 1\rangle \wedge \langle x_3, 1\rangle \wedge \langle z_1, 0\rangle$
- $\varphi_{\langle x_2,1\rangle} = \langle z_1, 0\rangle \vee \langle x_2, 1\rangle \vee x_4 \vee \nu(\langle x_3, 1\rangle)$.
- $\langle z_1, 1\rangle = p \wedge \langle z_1, 0\rangle \wedge \mu(\langle x_2, 1\rangle)$.
- $\varphi_{\langle x_3,2\rangle} = q \wedge \langle x_2, 2\rangle \wedge \langle x_3, 2\rangle \wedge \langle z_1, 1\rangle$
- $\varphi_{\langle x_2,2\rangle} = \langle z_1, 1\rangle \vee \langle x_2, 2\rangle \vee x_4 \vee \nu(\langle x_3, 2\rangle)$.
- $\langle z_1, 2\rangle = p \wedge \langle z_1, 1\rangle \wedge \mu(\langle x_2, 2\rangle)$.

– $\psi^2 = \langle z_1, 2\rangle$.

Next, we unfold x_3:

– $\langle z_1, 0\rangle =$ **true**
– • $\langle z_3, (1,0)\rangle =$ **true**
  • $\langle z_3, (1,1)\rangle = q \wedge \langle x_2, 1\rangle \wedge \langle z_3, (1,0)\rangle \wedge \langle z_1, 0\rangle$
  • $\langle z_3, (1,2)\rangle = q \wedge \langle x_2, 1\rangle \wedge \langle z_3, (1,1)\rangle \wedge \langle z_1, 0\rangle$
– $\varphi_{\langle x_2,1\rangle} = \langle z_1, 0\rangle \vee \langle x_2, 1\rangle \vee x_4 \vee \langle z_3, (1,2)\rangle$.
– $\varphi_{\langle z_1,1\rangle} = p \wedge \langle z_1, 0\rangle \wedge \mu(\langle x_2, 1\rangle)$.
– • $\langle z_3, (2,0)\rangle =$ **true**
  • $\langle z_3, (2,1)\rangle = q \wedge \langle x_2, 2\rangle \wedge \langle z_3, (2,0)\rangle \wedge \langle z_1, 1\rangle$
  • $\langle z_3, (2,2)\rangle = q \wedge \langle x_2, 2\rangle \wedge \langle z_3, (2,1)\rangle \wedge \langle z_1, 1\rangle$
– $\varphi_{\langle x_2,2\rangle} = \langle z_1, 1\rangle \vee \langle x_2, 2\rangle \vee x_4 \vee \langle z_3, (2,2)\rangle$.
– $\langle z_1, 2\rangle = p \wedge \langle z_1, 1\rangle \wedge \mu(\langle x_2, 2\rangle)$.
– $\psi^2 = \langle z_1, 2\rangle$.

Unfolding equivalence sets Generalizing the unfolding method for unfolding equivalence sets is fairly easy. Let ψ be a μ-calculus formula. Let X^e be an equivalence class of ν-variables, and let X'' the set variables that are smaller than the variables of X^e with respect to $\prec$. For $k > 0$, we write the k-th unfolding of ψ with respect to X^e as follows. The k-th subformula is made of k subformulas for each $x'' \in X''$ (using indexed variables), and $k+1$ auxiliary subformulas for the auxiliary variables $\langle z, l\rangle$ that replace the variables in X^e. We define the subformulas as follows.

– For $x'' \in X''$ we write k subformulas. We index the variables with indices $1 \leq l \leq k$, where the subformula of $\langle x'', l\rangle$ is, $\varphi_{x''}[x/\langle z, l-1\rangle, y''/\langle y'', l\rangle]$, i.e., every occurrence of $x \in X^e$ is replaced by $\langle z, l-1\rangle$, and every occurrence of $y'' \in X''$ is replaced by the variable $\langle y'', l\rangle$.
– For $x \in X^e$ we write $k+1$ subformulas where $\langle z, 0\rangle =$ **true** and for $1 \leq l \leq k$ we write $\langle z, l\rangle = \varphi_x[x/\langle z, l-1\rangle, y''/\langle y'', l\rangle]$, i.e., every occurrence of $x \in X^e$ is replaced by $\langle z, l-1\rangle$, every occurrence of $y'' \in X''$ is replaced by the variable $\langle y'', l\rangle$.
– In the subformulas for the variables that are in X', we replace $\lambda(x)$ where $x \in X^e$ by $\langle z, k\rangle$.

Lemma 5 generalizes Lemma 4.

Lemma 5. *Given a μ-calculus formula ψ of the form $\nu x.\varphi$, and a Kripke structure $M = \langle S, R, L\rangle$. Let X^e be the equivalence class of x, then $M \models \psi$ iff $M \models \psi^{|X^e|\cdot|S|}$.*

Organizing the Indices as Tuples Unfolding all ν-classes results in many indices. In the example above we organized the indices of z_3 in a tuple such that the first index is the result of unfolding $\{x_1\}$ and the second index is the result of unfolding $\{x_3\}$. We would like to formalized this method such that the indices are organized in a tuple, and we can associate an index with the equivalence class that "creates it".

Let ψ be a μ-calculus formula, and let ψ' be the result of removing all the ν-equivalence classes from ψ. Let x be a variable in ψ. Then, every ν-class X^e such that x depend on X^e adds an index to the occurrences of x in ψ' and so does the equivalence class of x if it is a ν-class. Then, the number of indices added to x is $w(x)$. We determine the position of the indices in the tuple according to $\prec$. Thus, the first index refers to the equivalence class X^e of the maximal variable x_m in $X' = \{x' | x \prec x'$ and x' is a ν variable$\}$, the second index refers to the maximal variable in $X' \setminus X^e$ and so on. Since $\prec$ is a total order over X', the position of every index in the tuple is well defined. We denote by $\langle x, d\rangle$ the variable that is the result of x with the indices of d.

Proposition 2. *Let x be a variable. Let $d = (d_0, d_1, \ldots d_{w(x)})$ be the tuple attached to x, and let $x \prec x'$. Then the position of the index created from the unfolding of the equivalence class of x' in the tuple, is $w(x')$.*

Note that we enumerate the positions in the tuple starting from one.

A.2 Optimizing the Unfolding Method

We would like to have a hierarchy over the variables of ψ such that $s \in \psi(f)(\langle x, d\rangle)$ only if for all $d' <_{w(x)} d$, we have $s \in \psi(f)(\langle x, d'\rangle)$. However, looking at the previous example, we can see that it is possible that $s \in \langle z_3, (2,0)\rangle$ but $s \notin \langle z_3, (1,2)\rangle$. Next, we define a new optimized unfolding method, which has this desired property.

Definition 3. *We denote by $d^{(l)}_{max}$ the maximal tuple of length l and by $d^{(l)}_{min}$ the minimal tuple of length l, i.e., $d^{(l)}_{max} = (|S| \cdot width(1), |S| \cdot width(3), \ldots, |S| \cdot width(2 \cdot l + 1))$, and $d^{(l)}_{min} = (1, 1, \ldots, 1, 0)$.*

Note that $d^{(l)}_{min}$ can match only auxiliary variables. The optimized definition is identical to the original except for the assignment to auxiliary variables with least index 0, for which we have the following definition. Let ψ be a μ-calculus formula, and let X^e be the ν-class that is being unfolded. We distinguish between two cases:

1. If there is no ν-equivalence class X' that is greater than X^e with respect to $\prec$, then we assign **true** to all the variables in X^e with index 0.
2. Otherwise, X^e contains auxiliary variables of the form $\langle z, d\rangle$, where d is the index obtained by the unfolding of ν-classes that are greater with respect to $\prec$. Let $l = w(x)$ for all $x \in X^e$. If $d = d^{(w(x))}_{min}$, then $\langle z, d\rangle =$ **true**, else let d' be the largest tuple that is smaller than d but with the same arity. Then, $\langle z, d\rangle = \langle z, d'\rangle$.

A μ-calculus formula ψ with free variables X' and bind variables X'' is considered as a function form $\mathcal{F}$ to 2^S. However, during its evaluation. ψ assign subsets of S to all the variables in X''. Thus, we can consider ψ as a function form assignments $X' \to 2^S$ to assignments $X'' \to 2^S$. This generalization maintain the monotonicity property. Let f_1 and f_2 be assignments in $\mathcal{F}$ such that for every $x' \in X'$ we have $f_1(x') \subseteq f_2(x')$ (denoted $f_1 \subseteq f_2$). Then, for every $x'' \in X''$ we have $\psi(f_1)(x'') \subseteq \psi(f_2)(x'')$.

Another parameter that we can generalize is the initialization of the variables in the computation of the fixpoint values. Theorem 2 implies that given a μ-equivalence class

$\mathbf{X}^e$ with maximal variable $x_m = max(X^e)$, and an assignment $f \in \mathcal{F}$, the least fixpoint value of the variables of X^e is
$(\varphi_m(f))^{|S|\cdot|X^e|}$(**false**, **false**, . . . , **false**). In this case we say that we initialize the variables of X^e to
(**false**, **false**, . . . , **false**). Similarly, for ν-equivalence class we initialize the variables to (**true**, **true**, . . . , **true**). We define the set $\mathcal{I}$ of all assignments $X'' \to 2^S$. We define the original assignment i^o as the assignment that assign **false** to all μ-variables, and assign **true** to all nu-variables. We say that i_1 we say that $i_1 \subseteq i_2$ if for every $x'' \in X''$ we have $i_1(x'') \subseteq i_2(x'')$. We say that an assignment i is μ consistent iff $i \subseteq i^o$ and ν consistent iff $i^o \subseteq i$. Given a μ-calculus formula ψ we consider it as a function from $\mathcal{F} \times \mathcal{I}$ to an assignment $X'' \to 2^S$. We write it as $\psi(f, i)(x'')$.

Proposition 3. *Let ψ be a μ-calculus formula, let f_1 and f_2 be in $\mathcal{F}$ and let i_1 and i_2 be μ-consistent assignment such that $f_1 \subseteq f_2$ and $i_1 \subseteq i_2$. Then, for every $x'' \in X''$ we have $\psi(f, i_1)(x'') \subseteq \psi(f, i_2)(x'')$.*

Lemma 6. *Let ψ be a μ-calculus formula, let f be in $\mathcal{F}$. Let $i \in \mathcal{I}$ be a ν consistent assignment such that for all $x'' \in X''$ we have $\psi(f, i^o)(x'') \subseteq i(x'')$. Then, for all $x'' \in X''$ we have $\psi(f, i^o)(x'') = \psi(f, i)(x'')$.*

Proof. : The first direction $\psi(f, i^o)(x'') \subseteq \psi(f, i)(x'')$ is implied directly from the monotonicity property. We prove that $\psi(f, i)(x'') \subseteq \psi(f, i^o)(x'')$. Note that $\psi(f, i^o)(x'')$ is a fixpoint value for all the variables of X''. This implies that for every $x'' \in X''$ we have $\psi(f, i^o)(x'') = \psi(f, \psi(f, i^o))(x'')$. Since for every $x'' \in X''$ we have $\psi(f, i^o)(x'') \subseteq i(x'')$, monotonicity implies that $\psi(f, i)(x'') \subseteq \psi(f, i^o)(x'')$. □

Lemma 7. *Let ψ be a μ-calculus formula and let $\ddot{\psi}$ be the result of unfolding ψ by the optimized method as described above. Let, $M = \langle S, R, L\rangle$ be a Kripke structure. Then $M \models \psi$ iff $M \models \ddot{\psi}$.*

Proof. : When we unfold ψ in the original method, we initialized every auxiliary variable of the form $\langle z, (d, 0)\rangle$ to **true**. Let S' be the value of $\langle z, (d, |S| \cdot n_{w(z)})\rangle$, then Lemma 6 implies that for every set $S' \subseteq S'' \subseteq$ **true**, we can assign $\langle z, (d, 0)\rangle = S''$ without changing the value of $\langle z, (d, |S| \cdot n_{w(z)})\rangle$. Since for every d and d' such that $d' \leq d$ we have that the value of $\langle z, (d, |S| \cdot n_{w(z)})\rangle$ is contained in $\langle z, (d', |S| \cdot n_{w(z)})\rangle$, we can assign $\langle z, (d, 0)\rangle = \langle z, (d', |S| \cdot n_{w(z)})\rangle$. This implies that the improved unfolding method result in the same value as the original method. □ The proof of Lemma 7 is presented in Appendix **??**. Note that in Example 2, the only subformula that is being changed is $\langle z_3, (2, 0)\rangle = \langle z_3, (1, 2)\rangle$. We can replace every occurrence of $\langle z_3, (2, 0)\rangle$ by $\langle z_3, (1, 2)\rangle$, so the only tuples that contain 0 are of the form $(1, 1, \ldots, 1, 0)$.

A.3 Proving the Collapse Theorem

We show that a μ-calculus formula ψ can be unfolded into a formula ψ' without ν-variables. The ”price” of this translation is the creation of exponentially many copies of subformulas of ψ. We show that the copies of a subformula φ can be organized in a hierarchy such that the position of a copy φ' of subformula φ is determined by a

tuple that is attached to it. Then, we improve the unfolding method such that a state s satisfies a copy of φ with tuple d only if it satisfies all copies of φ with tuple $d' \leq d$. Thus, we conclude that instead of creating multiplies copies of φ we can keep track of the greatest tuple d such that s satisfies φ indexed by d. We show that the $\mu^{\#}$-calculus logic does exactly that. Formally, we show that for the least fixpoint g_ψ of $\psi^{\#}$, we have that $s \models \varphi_{\langle x,d \rangle}$ iff $g_\psi(\varphi_x) \geq_{w(x)} d$.

Lemma 8. *Let ψ be a μ-calculus formula, and let ψ' be the result of unfolding ψ with respect to its ν-variables using the improved method. Let $g_\psi \in \mathcal{F}$ be the least fixpoint of $\psi^{\#}$. Then, for every s and d and for every subformula φ_x, we have that $M, s \models \varphi_{\langle x,d \rangle}$ iff $g_\psi(\varphi^{\#}_x)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. with the exception of auxiliary variables where $M, s \models \langle z, d \rangle$ iff $g_\psi(z)(s) \geq_{w(z)} \mathtt{inc}_{w(z)}(d)$,*

Proof. : The proof is by induction over the computation of the simultaneous least fixpoint of ψ'. We start with a proposition that define the base of the induction.

Proposition 4. *Let x be a variable (z be an auxiliary variable) in a μ-calculus formula ψ without free variables. Let $\varphi_{\langle x,d \rangle}$ ($\varphi_{\langle z,d \rangle}$) be a subformula of result of the optimized unfolding method. Then:*

- *The length of the tuple d is $w(x)$ (or $w(z)$).*
- *all the variables in $\varphi_{\langle m,d \rangle}$ are in*
  $\{\langle x_m, d \rangle\}$ if x is a μ-variable or $\{\langle z, d' \rangle | d = \mathtt{inc}_{w(z_m)}(d')\}$ if it is an auxiliary variable, and
  $\{\langle x', d \rangle\}$ if x is a μ-variable and x' in the same equivalence class or $\{\langle z', d' \rangle | d = \mathtt{inc}_{w(z')}(d')\}$ if z is an auxiliary variable and z' in the same equivalence classand
  $\{\langle x', d' \rangle | x \prec x' \wedge d =_{w(x')} d'\} \cup$
  $\{\langle z', d' \rangle | x \prec z' \wedge d =_{w(z')} \mathtt{inc}_{w(z')} d'\} \cup$
  $\{\langle z'', (d', |S|) \rangle | d' = d \vee d'$ is shorter than d and $d' =_{w(x'')-1} d\} \cup$
  $\{\mu(\langle x'', d' \rangle) | d' = d \vee d'$ is shorter than d and $d' =_{w(x'')} d\}$.

End proposition

In the proof of the lemma we denote the tuple $(1, 1, \ldots 1, 0)$ as $-\infty$. We can do that because all the variables that are attach to a tuple of the form $(1, 1, \ldots, 1, 0)$ assigned to **true**. Note that for every l we have $(1, 1, \ldots, 1) = \mathtt{inc}_l - \infty$. The proof is by induction over the computation of the simultaneous least fixpoint of ψ'.

- Base: Note that ψ' has only μ non-auxiliary variables, thus the initial value of every $\langle x, d \rangle$ is assign to $\emptyset$. Since, $g_0(x) = -\infty$ and there are no variables with these indices in ψ', the lemma holds. As for the auxiliary variables, the basic auxiliary variables are of the form $\langle z, -\infty \rangle$, these variables are set to S. Since for every variable z and state s we have $g_0(z)(s) \geq_{w(z)} \mathtt{inc}_{w(z)}(-\infty)$, the lemma holds.
- Induction step: The induction step is by induction over the structure of ψ'. For each variable x or z, we prove the lemma by induction over the structure of φ_x or φ_z respectively. Let $\mu\langle x, d \rangle.\varphi_{\langle x,d \rangle}$ be a subformula of ψ', assume that the lemma holds for every variable in $\varphi_{\langle x,d \rangle}$. We prove with induction over $\varphi_{\langle x,d \rangle}$ that for every subformula φ' of $\varphi_{\langle x,d \rangle}$, $s \models \varphi'$ iff $g_\psi(\varphi')(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$.

- Base: For $p \in AP$, we have that $p \in L(s)$ iff $g_\psi(p)(s) = \infty$, otherwise, it assigned to $-\infty$ and no subformula of ψ' is attach to $-\infty$. Similarly the lemma holds for $\neg p$. The induction assumption implies that the claim holds for the variables of the subformula $\varphi_{\langle x,d\rangle}$ (with their own tuples). Proposition 4 implies that $\varphi_{\langle x,d\rangle}$ contains seven types of variables:
  - $\langle x, d\rangle$ itself - the induction assumption implies that the lemma holds. The same is true if for $\langle z, d'\rangle$.
  - $\langle x', d\rangle$ itself - the induction assumption implies that the lemma holds. The same is true if for $\langle z', d'\rangle$ for z' in the same equivalence class.
  - $\langle z, d\rangle'$ where $d = \mathtt{inc}_{w(z)}d'$ - the induction assumption implies that the lemma holds.
  - A variable in $\{\langle x', d'\rangle | x \prec x' \wedge d' =_{w(x')} d\}$. In this case the induction hypothesis implies that $s \models \langle x', d'\rangle$ iff $g_\psi(x')(s) \geq_{w(x')} d'$. Since $x \prec x'$, x' has less indices than x, thus $g_\psi(x')(s) \geq_{w(x)} d'$ iff $g_\psi(x')(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$.
  - A variable in $\{\langle z', d'\rangle | x \prec z' \wedge d =_{w(z')} \mathtt{inc}_{w(z')}(d')\}$. In this case the induction hypothesis implies that $s \models \langle z', d'\rangle$ iff $g_\psi(z')(s) \geq_{w(x')} d''$ where $d'' = \mathtt{inc}_{w(z)}(d)$. Thus, the lemma holds as in the previous case
  - A subformula $\mu(\langle x'', d\rangle)$ for either $d' = d$ or d' is shorter than d and $d' =_{w(x'')} d$. Here, the induction hypothesis implies that $s \models \varphi_{\langle x'', d'\rangle}$ iff $g_\psi(\varphi^{\#}_{x''})(s) \geq_{w(x'')} \mathtt{set}_{w(x'')}(d')$. Since we compute the values of the variables simultaneously, $\mu(\langle x'', d'\rangle)$ simply get the value of $\varphi_{\langle x'', d'\rangle}$. Thus, $s \models \varphi_{\langle x'', d'\rangle}$ iff $s \models \mu(\langle x'', d'\rangle)$. Similarly, $\mathtt{set}_{w(x'')}(d') = \mathtt{set}_{w(x)}(d')$. As in the cases above $s \models \mu(\langle x'', d\rangle)$ iff $g_\psi(\mathtt{set}_{w(x'')}x''.\varphi^{\#}_{x''})(s) \geq_{w(x'')} \mathtt{set}_{w(x'')}(d')$ iff
    $g_\psi(\mathtt{set}_{w(x'')}x''.\varphi^{\#}_{x''})(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. Furthermore, we conclude that $s \models \langle x'', d\rangle$ iff $g_\psi(x'') \geq_{w(x'')} d$.
  - An auxiliary variable $\langle z'', (d', |S|)\rangle$ such that $d' = d$ or d' is shorter than d and $d' =_{w(x'')-1} d$ The induction hypothesis implies that $s \models \varphi_{\langle z'', (d', |S|)\rangle}$ iff $g_\psi(\varphi^{\#}_{z''})(s) \geq_{w(z'')} \mathtt{set}_{w(z'')}((d', |S|))$. Since $\langle z'', d'\rangle$ is the result of unfolding a nu-variable, the matching $\mu^{\#}$-calculus subformula is $\mathtt{inc}_{w(z'')}\, z''.\varphi_{z''}$. By the definition of the inc operator, $g_\psi(\varphi^{\#}_{z''})(s) \geq_{w(z'')} \mathtt{set}_{w(z'')}((d', |S|))$ iff $g_\psi(\mathtt{inc}\ z''.\varphi^{\#}_{z''})(s) \geq_{w(z'')} \mathtt{inc}_{w(z'')}((d', |S|))$.
    Note, that
    $g_\psi(\mathtt{inc}\ z''.\varphi^{\#}_{z''})(s) \geq_{w(z'')} \mathtt{inc}_{w(z'')}((d', |S|))$ iff $g_\psi(\mathtt{inc}\ z''.\varphi^{\#}_{z''})(s) \geq_{w(z'')-1} (d')$. The tuple d' is shorter or equal than d, thus the lemma holds as in the cases above.
    Furthermore, we conclude that $s \models \langle z'', d'\rangle$ iff $g_\psi(x'') \geq_{w(x'')} \mathtt{inc}_{w(z'')}(d')$.
- Induction step: Assume that the lemma holds for θ_1 and θ_2. We prove it to φ'.
  - If $\varphi' = \theta_1 \vee \theta_2$, then $s \models \varphi'$ iff $s \models \theta_1$ or $s \models \theta_2$. By the induction hypothesis, this holds iff $g_\psi(\theta^{\#}_1)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$ or $g_\psi(\theta^{\#}_2)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff $max(g_\psi(\theta^{\#}_1)(s), g_\psi(\theta^{\#}_2)(s)) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff $g_\psi(\varphi'^{\#})(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$.

* If $\varphi' = \theta_1 \wedge \theta_2$, then $s \models \varphi'$ iff $s \models \theta_1$ and $s \models \theta_2$. By the induction hypothesis, this holds iff $g_\psi(\theta_1^\#)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$ and $g_\psi(\theta_2^\#)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff $min(g_\psi(\theta_1^\#)(s), g_\psi(\theta_2^\#)(s)) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff $g_\psi(\varphi'^\#)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$.
* If $\varphi' = \Diamond\theta_1$, then $s \models \varphi'$ iff there exists a successor s' of s such that $s' \models \theta_1$. By the induction hypothesis, this holds iff $g_\psi(\theta_1^\#)(s') \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff there exists a successor s' of s such that $g_\psi(\theta_1^\#)(s') \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff $g_\psi(\varphi'^\#)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$.
* If $\varphi = \Box\,\theta_1$, then $s \models \varphi$ iff for every successor s' of s we have $s' \models \theta_1$. By the induction hypothesis, this holds iff for every successor s' of s we have $g_\psi(\theta_1^\#)(s') \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. This holds iff $g_\psi(\varphi'^\#)(s) \geq_{w(x)} \mathtt{set}_{w(x)}(d)$. □

Note that for a μ-calculus formula ψ of the form $\nu x.\varphi$, the Lemmas 7 and 8 together imply Theorem 3.

B Proofs for the Symbolic algorithms

The proof of Lemma 1

Proof. We prove the lemma by induction over the recursive run of the MERGE procedure.

- Base case, if n_1 and n_2 are terminal nodes, then both nodes represents function from a set of one element $\{u\}$ to D. The MERGE procedure assign the minimum value to the single element of the domain.
- We prove the induction step for the case that $o(n_1.v) = o(n_2.v)$. The MERGE procedure simply partition the domain U into two disjoint domains, that differ by the value of $n_1.v$ for each part the MERGE procedure recursively find the minimum function (induction hypothesis), and then the parts are unified using $n_1.v$.
  In case that $o(n_1.v) > o(n_2.v)$, the procedure does the same thing only it refers to n_1 as a node in the level of n_2 with both successors in n_1. The same holds in case that $o(n_1.v) < o(n_2.v)$. □

The proof of Lemma 2

Proof. We prove the lemma by induction over the recursive run of the procedure.

- Base case, trivial.
- Induction closure. We distinguish between two cases:
  1. If $n.v$ is in V, then the procedure simply partitions U into two disjoint parts U_1 and U_2 that differ on the value of $n.v$. This partition defines two function $f_{suc1} : (U_1 \times U') \rightarrow D$ and $f_{suc2} : (U_2 \times U') \rightarrow D$ represented by $n.l$ and $n.r$. Then the procedure recursively computes the minimum functions for both nodes (induction hypothesis) and unifies the functions using $n.v$.

2. If $n.v$ is in V' then MIN($n.r$) and MIN($n.l$) represents two different minimum functions f_1 and f_2 from U to D. Thus we need a function that assign for every u in U the value $min(f_1(u), f_2(u))$. Lemma 2 implies that the MERGE procedure does it. □

Here we present the proof of Lemma 3:

Proof. We prove the lemma by induction over the recursive run of the procedure. The program recursively runs over the ADDs n_{suc} and n_{min}. Note, that in each step, the set of V variables of n_{suc} is equal to the set of variables of n_{min}. In case the ADDs "skip" a node from which both pointers point at the same successor, the procedure "behaves" as if that node exists, thought we ignore this case in the proof.

- Tail case: Both nodes are terminals. In this case the Adds represent functions from a domain with a single element to $\mathbb{N}$. Thus, the function returns **true** iff the nodes are equal.
- Recursion closure. We distinguish between two cases:
  1. If $n_{suc}.v$ is in V, then the procedure simply partitions U into two disjoint parts U_1 and U_2 that differ on the value of $n_{suc}.v$. This partition defines two suc functions $f_{suc1} : (U_1 \times U') \rightarrow D$ and $f_{suc2} : (U_2 \times U') \rightarrow D$ represented by $n_{suc}.r$ and $n_{suc}.l$, and two min functions $f_{min1} : U_1 \rightarrow D$ and $f_{min2} : U_2 \rightarrow D$ represented by $n_{min}.r$ and $n_{min}.l$. Then, the procedure recursively computes the π_1 and π_2 relations for the two parts of U (induction hypothesis) and unifies the relations using $n_{suc}.v$. Since the two domains are disjoint the result is exactly the desired relation.
  2. If $n_{suc}.v$ is in V' then the procedure simply partitions U' into two disjoint parts U'_1 and U'_2 that differ on the value of $n_{suc}.v$. This partition defines two suc functions $f_{suc1} : (U \times U'_1) \rightarrow D$ and $f_{suc2} : (U \times U'_2) \rightarrow D$ represented by $n_{suc}.r$ and $n_{suc}.l$. The Function n_{min} remains the same. Then, the procedure recursively computes the π_1 and π_2 relations for the two parts of U' (induction hypothesis) and unifies the relations using $n_{suc}.v$. using $n_{suc}.v$. □

The complexity of the STRATEGY procedure is linear in the sizes of f_{suc} and f_{min}, thus it does not change the overall complexity.