Enriched μ-Calculi Module Checking*

Alessandro Ferrante[1] and Aniello Murano[2]

[1] Università di Salerno, Via Ponte don Melillo, 84084 - Fisciano (SA), Italy
[2] Università di Napoli Federico II, Via Cintia, 80126 - Napoli, Italy

Abstract. The model checking problem for open finite-state systems (called *module checking*) has been intensively studied in the literature with respect to *CTL* and *CTL**. In this paper, we focus on module checking with respect to the *fully enriched μ-calculus* and some of its fragments. Fully enriched μ-calculus is the extension of the propositional μ-calculus with *inverse programs*, *graded modalities*, and *nominals*. The fragments we consider here are obtained by dropping at least one of the additional constructs. For the full calculus, we show that module checking is undecidable by using a reduction from the domino problem. For its fragments, instead, we show that module checking is decidable and ExpTime-complete. This result is obtained by using, for the upper bound, a classical automata-theoretic approach via *Forest Enriched Automata* and, for the lower bound, a reduction from the module checking problem for *CTL*, known to be ExpTime-hard.

1 Introduction

One of the most significant developments in the area of formal design verification has been the discovery of the *model-checking* technique, which is particularly suitable for verifying ongoing behaviors of reactive systems ([CE81, QS81, VW86]). In this verification method, (for a survey, see [CGP99]), the behavior of a system, formally described by a mathematical model, is checked against a behavioral constraint specified by a formula in a suitable temporal logic, which enforces either a linear model of time (formulas are interpreted over linear sequences corresponding to single computations of the system) or a branching model of time (formulas are interpreted over infinite trees, which describe all the possible computations of the system).

In system modeling, we distinguish between *closed* and *open* systems [HP85]. For a closed system, the behavior is completely determined by the state of the system. For an open system, the behavior is affected both by its internal state and by the ongoing interaction with its environment. Thus, while in a closed system all the nondeterministic choices are internal, and resolved by the system, in an open system there are also external nondeterministic choices, which are resolved by the environment [Hoa85]. Model checking algorithms used for the verification of closed systems are not appropriate for the verification of open systems. In the

* Work partially supported by MIUR FIRB Project no. RBAU1P5SS and grant ex-60% 2005, Università di Salerno.

H. Seidl (Ed.): FOSSACS 2007, LNCS 4423, pp. 183–197, 2007.

latter case, we should check the system with respect to arbitrary environments and should take into account uncertainty regarding the environment.

In [KVW01], Kupferman, Vardi, and Wolper extend model checking from closed finite-state systems to open finite-state systems. In such a framework, the open finite-state system is described by a labeled state-transition graph called *module* whose set of states is partitioned into a set of *system states* (where the system makes a transition) and a set of *environment states* (where the environment makes a transition). The problem of model checking a module (called *module checking*) has two inputs: a module M and a temporal formula φ. The idea is that an open system should satisfy a specification φ no matter how the environment behaves. Let us consider the unwinding of M into an infinite tree, say T_M. Checking whether T_M satisfies φ, (formally, $M \models \varphi$) is the usual *model-checking problem* [CE81, QS81]. On the other hand, for an open system, T_M describes the interaction of the system with a maximal environment, i.e., an environment that enables all the external nondeterministic choices. In order to take into account all the possible behaviors of the environment, we have to consider all the trees T obtained from T_M by pruning subtrees whose root is a successor of an environment state (pruning these subtrees correspond to disable possible environment choices). Therefore, a module M satisfies φ (formally, $M \models_r \varphi$, where r stands for "reactively") if all these trees T satisfy φ. The set of all the trees derived from T_M by a legal pruning is denoted by $exec(M)$.

In [KVW01], it has been showed that model checking for open finite-state systems is ExpTime-complete for specification in CTL and 2ExpTime-complete for specification in CTL^*. Moreover, the *program complexity*, i.e., the complexity of the problem assuming the formula to be fixed, is Ptime-complete. Recently, module checking has been also extended to infinite–state systems, by considering open pushdown systems as models [BMP05]. It has been showed that in this framework module checking is 2ExpTime-complete for specification in CTL and 3ExpTime-complete for specification in CTL^*.

The *μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. It is often used as a target formalism for embedding temporal and modal logics with the goal of transferring computational and model theoretic properties such as the ExpTime upper complexity bound (see [BS06] for a survey). *Fully enriched μ-calculus* is the extension of the propositional μ-calculus with *inverse programs*, *graded modalities*, and *nominals*. Intuitively, inverse programs allow to travel backwards along accessibility relations [Var98], nominals are propositional variables interpreted as singleton sets [SV01], and graded modalities enable statements about the number of successors and predecessors of a state [KSV02]. In [BP04], Bonatti and Peron showed that satisfiability is undecidable in the *fully enriched μ-calculus*. On the other hand, the satisfiability problem for interesting fragments of the fully enriched μ-calculus has been showed to be decidable and ExpTime-complete. In particular, it has been showed for the fragments of the fully enriched μ-calculus obtained by dropping at least one of graded modalities (*fully hybrid μ-calculus*)[SV01], nominals (*full graded μ-calculus*) [BLMV06], and inverse programs (*hybrid graded*

	Inverse progr.	Graded mod.	Nominals	Complexity
fully enriched μ-calculus	x	x	x	undecidable
full graded μ-calculus	x	x		ExpTime
full hybrid μ-calculus	x		x	ExpTime
hybrid graded μ-calculus		x	x	ExpTime

Fig. 1. Enriched μ-calculi and known results

μ-calculus)[BLMV06]. These enriched μ-calculi are shown in Fig. 1 together with the complexity of their satisfiability problem.

The above decidability results are based on an automata-theoretic approach via *fully enriched automata (FEAs)*, which run on infinite forests and use a parity acceptance condition. Intuitively, these automata generalize alternating automata on infinite trees in a similar way as the fully enriched μ-calculus extends the standard μ-calculus: FEAs can move up to a node's predecessor (by analogy with inverse programs), move down to at least n or all but n successors (by analogy with graded modalities), and jump directly to the roots of the input forest (which are the analogues of nominals). The decidability results follow from the fact that all the above fragments enjoy the *forest model property* (while some of them do not enjoy neither the tree model property nor the finite model property), and from the fact that the emptiness problem for fully enriched automata is decidable and ExpTime-complete. Observe that decidability of the emptiness problem for FEAs does not contradict the undecidability of the fully enriched μ-calculus: the latter does not enjoy a forest model property [BP04], and hence satisfiability cannot be decided using forest-based FEAs.

In this paper, we extend the module checking problem for finite-state systems to the *fully enriched μ-calculus* and we show that this problem is undecidable. To gain this result, we use a reduction from the domino problem [Ber66], known to be undecidable, by extending an idea due to Bonatti and Peron in [BP04].

Moreover, we consider the problem of module checking for the fragments of the full calculus as listed in Fig. 1. That is, we consider the module checking problem whit respect to formulas of the fully hybrid, full graded, and hybrid graded μ-calculus. We show that in all the above frameworks, the module checking problem is decidable and ExpTime-complete. For the upper bound, we use an automata-theoretic approach via FEA. In more details, given a model M and a formula φ, we first build a Büchi automaton $\mathcal{A}_M$, accepting $exec(M)$. In particular, since M requires to be unwound in a forest rather then a tree (since all the fragments we consider enjoy the forest model property, while those including nominals do not enjoy the tree model property), the set $exec(M)$ is a set of forests, and thus, $\mathcal{A}_M$ is a Büchi automaton running on forests (BFA, for short). Then, accordingly to [SV01] and [BLMV06], we build a FEA $\mathcal{A}_{\neg\varphi}$ accepting all models of $\neg\varphi$, with the intent to check that no models of $\neg\varphi$ are in $exec(M)$. Thus, we check that $M \models_r \varphi$ by checking whether $\mathcal{L}(\mathcal{A}_M) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. The results follow from the fact that BFAs are a particular case of FEAs, which are closed under intersection and have the emptiness problem solvable in ExpTime [BLMV06].

We also show a lower bound matching the obtained upper bound by using a reduction from the module checking for *CTL*, known to be ExpTime-hard.

2 Preliminaries

Labeled Forests. For a finite set X, we denote the set of finite words over X by X^*, the empty word by ε, and with X^+ we denote $X^* \setminus \{\varepsilon\}$. Given a word w in X^* and a symbol x of X, we use $w \cdot x$ to denote the word wx. Let $\mathbb{N}$ be the set of positive integers. For $n \in \mathbb{N}$, let $\mathbb{N}$ be denote the set $\{1, 2, \ldots, n\}$. A *forest* is a set $F \subseteq \mathbb{N}^+$ such that if $x \cdot c \in F$ where $x \in \mathbb{N}^+$ and $c \in \mathbb{N}$, then also $x \in F$. The elements of F are called *nodes*, and the strings consisting of a single natural number are the *roots* of F. For each root $r \in F$, the set $T = \{r \cdot x \mid x \in \mathbb{N}^* \text{ and } r \cdot x \in F\}$ is a *tree* of F (the tree *rooted in* r). For every $x \in F$, the nodes $x \cdot c \in F$ where $c \in \mathbb{N}$ are the *successors* of x, denoted $children(x)$, and x is their *predecessor*. The number of successors of a node x is called the *branching degree* of x, and is denoted by $bd(x)$. The degree of a forest is the maximum of the degrees of a node in the forest and the number of roots.

Let $F \subseteq \mathbb{N}^+$ be a forest and x a node in F. As a convention, we take $x \cdot \varepsilon = x$, $(x \cdot c) \cdot -1 = x$, and $n \cdot -1$ as undefined, for $n \in \mathbb{N}$. We call x a *leaf* if it has no successors. A *path* π in F is a word $\pi = a_1 a_2 \ldots$ of F such that a_1 is a root of F and for every $a_i \in \pi$, either a_i is a leaf (i.e., π ends in a_i) or a_i is a predecessor of a_{i+1}. Given two alphabets Σ_1 and Σ_2, a (Σ_1, Σ_2)-labeled forest is a triple $\langle F, V, E\rangle$, where F is a forest, $V : F \to \Sigma_1$ maps each node of F to a letter in Σ_1, and $E : F \times F \to \Sigma_2$ is a partial function that maps each pair (x, y), with $y \in children(x)$, to a letter in Σ_2. As a particular case, we consider a forest without labels on edges as a Σ_1-labeled forest $\langle F, V\rangle$, and a tree as a forest containing exactly one tree.

A *quasi-forest* is a forest where each node may also have roots as successors. Thus, for each node x of a quasi-forest $F \subseteq \mathbb{N}^+$, we denote with $successor(x)$ the successors of x and $children(x) = successor(x) \setminus \mathbb{N}$. All the other definitions regarding forests easily extend to quasi-forest. Notice that in a quasi-forest, a root can also have several predecessors, while every other node has always a unique one. Clearly, a quasi-forest can be always transformed in a forest by removing root successors.

Enriched Automata. For a given set Y, let $B^+(Y)$ be the set of positive Boolean formulas over Y (i.e., Boolean formulas built from elements in Y using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and $\wedge$ has precedence over $\vee$. For a set $X \subseteq Y$ and a formula $\theta \in B^+(Y)$, we say that X satisfies θ iff assigning **true** to elements in X and assigning false to elements in $Y \setminus X$ makes θ true. For $b > 0$, let $\langle [b]\rangle = \{\langle 0\rangle, \langle 1\rangle, \ldots, \langle b\rangle\}$, $[[b]] = \{[0], [1], \ldots, [b]\}$, and $D_b = \langle [b]\rangle \cup [[b]] \cup \{-1, \varepsilon, \langle root\rangle, [root]\}$.

A fully enriched automaton is an automaton in which the transition function δ maps a state q and a letter σ to a formula in $B^+(D_b \times Q)$. Intuitively, an atom $(\langle n\rangle, q)$ (resp., $([n], q)$) means that the automaton sends copies in state q to $n+1$

(resp., all but n) different successors of the current node, (ε, q) means that the automaton sends a copy (in state q) to the current node, $(-1, q)$ means that the automaton sends a copy to the predecessor of the current node, and $(\langle root\rangle, q)$ and $([root], q)$ mean that the automaton sends a copy to some, respectively all of the roots of the forest. When, for instance, the automaton is in state q, reads a node x, and $\delta(q, V(x)) = (-1, q_1) \wedge ((\langle root\rangle, q_2) \vee ([root], q_3))$, it sends a copy in state q_1 to the predecessor and either sends a copy in state q_2 to one of the roots or a copy in state q_3 to all roots.

Formally, a *fully enriched automaton* (FEA, for short) is a tuple $\mathcal{A} = \langle \Sigma, b, Q, \delta, Q_0, \mathcal{F}\rangle$, where Σ is the input alphabet, $b > 0$ is a counting bound, Q is a finite set of states, $\delta : Q \times \Sigma \to B^+(D_b \times Q)$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $\mathcal{F}$ is the acceptance condition. A *run* of $\mathcal{A}$ on an input Σ-labeled forest $\langle F, V\rangle$ is a tree $\langle T_r, r\rangle$ in which each node is labeled by an element of $F \times Q$. Intuitively, a node in T_r labeled by (x, q) describes a copy of the automaton in state q that reads the node x of F. Runs start in the initial state and satisfy the transition relation. Thus, a run $\langle T_r, r\rangle$ with root z has to satisfy the following: (i) $r(z) = (c, q_0)$ for some root c of F and $q_0 \in Q_0$ (ii) for all $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$, there is a (possibly empty) set $S \subseteq D_b \times Q$, such that S satisfies θ, and for all $(d, s) \in S$, the following hold:

- If $d \in \{-1, \varepsilon\}$, then $x \cdot d$ is defined and there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$;
- If $d = \langle n\rangle$, then there are distinct $i_1, \ldots, i_{n+1} \in \mathbb{N}$ such that for all $1 \leq j \leq n+1$, there is $j' \in \mathbb{N}$ such that $y \cdot j' \in T_r$, $x \cdot i_j \in F$, and $r(y \cdot j') = (x \cdot i_j, s)$;
- If $d = [n]$, then there are distinct $i_1 \ldots, i_{bd(x)-n} \in \mathbb{N}$ such that for all $1 \leq j \leq bd(x)-n$, there is $j' \in \mathbb{N}$ such that $y \cdot j' \in T_r$, $x \cdot i_j \in F$, and $r(y \cdot j') = (x \cdot i_j, s)$;
- If $d = \langle root\rangle$, then for some root $c \in F$ and some $j \in \mathbb{N}$ such that $y \cdot j \in T_r$, it holds that $r(y \cdot j) = (c, s)$;
- If $d = [root]$, then for all roots $c \in F$ there exists $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (c, s)$.

A run $\langle T_r, r\rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. We consider here the *parity acceptance condition*, where $\mathcal{F} = \{F_1, \ldots, F_k\}$ is such that $F_1 \subseteq \ldots \subseteq F_k = Q$. The number k of sets in $\mathcal{F}$ is called the *index* of the automaton. Given a run $\langle T_r, r\rangle$ and an infinite path $\pi \subseteq T_r$, let $Inf(\pi) \subseteq Q$ be such that $q \in Inf(\pi)$ iff there are infinitely many $y \in \pi$ for which $r(y) \in F \times \{q\}$. A path π *satisfies* a parity acceptance condition $\mathcal{F} = \{F_1, \ldots, F_k\}$ iff there is an even i for which $Inf(\pi) \cap F_i \neq \emptyset$ and $Inf(\pi) \cap F_{i-1} = \emptyset$. An automaton *accepts* a forest iff there exists an accepting run of the automaton on the forest. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled forests that $\mathcal{A}$ accepts. The *emptiness problem* for FEAs is to decide, given a FEA $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \emptyset$. In the following theorem we recall the exact complexity of this decision problem.

Theorem 1. *[BLMV06] The nonemptiness problem for a fully enriched automaton $\mathcal{A} = \langle \Sigma, b, Q, \delta, Q_0, \mathcal{F}\rangle$ can be solved in time linear in the size of Σ and b, and exponential in the index of the automaton and number of states.*

The following results on FEAs will be useful in the rest of the paper.

Lemma 1. *[BLMV06] Given two FEAs $\mathcal{A}_1$ and $\mathcal{A}_2$, there exists a FEA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ and whose size is linear in the size of $\mathcal{A}_1$ and $\mathcal{A}_2$.*

As a particular case of FEA, we consider nondeterministic Büchi Automata running on forests (BFA, for short). Formally, a BFA is a tuple $\mathcal{A} = \langle \Sigma, D, Q, \delta, Q_0, \mathcal{F} \rangle$, where Σ, Q, and Q_0 are defined as in FEA, D is a finite set of branching degrees, $\mathcal{F} \subseteq Q$ is a Büchi acceptance condition, and $\delta : Q \times \Sigma \times D^2 \rightarrow 2^{Q^* \times (Q \times Root)^*}$ is the transition relation satisfying $\delta(q, \sigma, d_1, d_2) \in 2^{Q^{d_1} \times (Q \times Root)^{d_2}}$, for every $q \in Q$, $\sigma \in \Sigma$ and $d_1, d_2 \in D$.

A *run* of $\mathcal{A}$ on an input Σ-labeled forest $\langle F, V \rangle$ of branching degree D is a tree $\langle T_r, r \rangle$ in which each node is labeled by an element of $F \times Q$. Formally, $\langle T_r, r \rangle$ is a run if $r(z) = (Q_0, z)$, for some root z of F and $q_0 \in Q_0$, and for all $y \in T_r$ labeled with (q, x), having d successors where d_2 are roots successors and d_1 are the remaining ones, we have that $r(y \cdot i) = \langle q_i, x \cdot i \rangle$ for all $1 \leq i \leq d_1$, $r(y \cdot (d_1 + i)) = \langle q_{d_1+i}, x_i \rangle$ for all $1 \leq i \leq d_2$ and $\langle \langle q_1, \ldots, q_{d_1} \rangle, \langle r(y \cdot (d_1 + 1)), \ldots, r(y \cdot d) \rangle \rangle \in \delta(q, V(x), d_1, d_2)$. A run $\langle F, V \rangle$ of a BFA is accepting if for all paths π of T_r, we have that $Inf(\pi) \cap \mathcal{F} \neq \emptyset$. Notice that $\mathcal{F}$ can be also expressed as the particular parity condition $\{\emptyset, \mathcal{F}\}$.

3 Fully Enriched μ-Calculus

Let AP, Var, $Prog$, and Nom be finite and pairwise disjoint sets of *atomic propositions*, *propositional variables*, *atomic programs*, and *nominals*. A *program* is an atomic program a or its converse a^-. The set of *formulas of the fully enriched μ-calculus* is the smallest set such that (i) **true** and **false** are formulas; (ii) p and $\neg p$, for $p \in AP \cup Nom$, are formulas; (iii) $x \in Var$ is a formula; (iv) if φ_1 and φ_2 are formulas, α is a program, n is a non-negative integer, and y is a propositional variable, then the following are also formulas: $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\langle n, \alpha \rangle \varphi_1$, $[n, \alpha] \varphi_1$, $\mu y.\varphi_1(y)$, and $\nu y.\varphi_1(y)$.

Observe that we use positive normal form, i.e., negation is applied only to atomic propositions. We call μ and ν *fixpoint operators* and use λ to denote a fixpoint operator μ or ν. A propositional variable y occurs *free* in a formula if it is not in the scope of a fixpoint operator, and *bound* otherwise. A *sentence* is a formula that contains no free variables. We refer often to the *graded modalities* $\langle n, \alpha \rangle \varphi_1$ and $[n, \alpha] \varphi_1$ as *atleast formulas* and *allbut formulas* and assume that the integers in these operators are given in binary coding: the contribution of n to the length of the formulas $\langle n, \alpha \rangle \varphi$ and $[n, \alpha] \varphi$ is $\lceil \log n \rceil$ rather than n. We refer to fragments of the fully enriched μ-calculus using the names from Fig. 1. Hence, we say that a formula φ of the fully enriched μ-calculus is also a formula of the *hybrid graded*, *full hybrid*, or *full graded* μ-calculus if φ does not have inverse programs, graded modalities, or nominals, respectively. If at least one of the above holds, we also say that φ is an *enriched μ-calculus* formula. To avoid confusion, we observe that enriched formulas are also formulas of the full calculus, while the converse is not always true. We recall that enriched formulas enjoy the forest model property (as showed in [BLMV06] and [SV01]), while fully enriched formulas does not [BP04].

The semantics of the fully enriched μ-calculus is defined with respect to a *Kripke structure*, i.e., a tuple $K = \langle W, W_0, R, L\rangle$ where W is a non-empty set of *states*, $W_0 \subseteq W$ is the set of initial states, $R : Prog \to 2^{W\times W}$ is a total function (i.e., for each $v \in W$ there is a program $a \in Prog$ and a node w such that $(v, w) \in R(a)$) that assigns to each atomic program a transition relation over W, and $L : AP \cup Nom \to 2^W$ is a labeling function that assigns to each atomic proposition and nominal a set of states such that the sets assigned to nominals are singletons and subsets of W_0. To deal with inverse programs, we extend R as follows: for each $a \in Prog$, set $R(a^-) = \{(v, u) : (u, v) \in R(a)\}$. If $(w, w') \in R(\alpha)$, we say that w' is an *α-successor* of w. Informally, an *atleast* formula $\langle n, \alpha\rangle\varphi$ holds at a state w of a Kripke structure K if φ holds at least in $n + 1$ α-successors of w. Dually, the *allbut* formula $[n, \alpha]\varphi$ holds in a state w of a Kripke structure K if φ holds in all but at most n α-successors of w. Note that $\neg\langle n, \alpha\rangle\varphi$ is equivalent to $[n, \alpha]\neg\varphi$, and that the modalities $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ of the standard μ-calculus can be expressed as $\langle 0, \alpha\rangle\varphi$ and $[0, \alpha]\varphi$, respectively.

To formalize semantics, we introduce valuations. Given a Kripke structure $K = \langle W, W_0, R, L\rangle$ and a set $\{y_1, \ldots, y_n\}$ of variables in Var, a *valuation* $\mathcal{V} : \{y_1, \ldots, y_n\} \to 2^W$ is an assignment of subsets of W to the variables $y_1, \ldots, y_n$. For a valuation $\mathcal{V}$, a variable y, and a set $W' \subseteq W$, we denote by $\mathcal{V}[y \leftarrow W']$ the valuation obtained from $\mathcal{V}$ by assigning W' to y. A formula φ with free variables among $y_1, \ldots, y_n$ is interpreted over the structure K as a mapping φ^K from valuations to 2^W, i.e., $\varphi^K(\mathcal{V})$ denotes the set of points that satisfy φ under valuation $\mathcal{V}$. The mapping φ^K is defined inductively as follows:

- $\mathbf{true}^K(\mathcal{V}) = W$ and $\mathbf{false}^K(\mathcal{V}) = \emptyset$;
- for $p \in AP \cup Nom$, we have $p^K(\mathcal{V}) = L(p)$ and $(\neg p)^K(\mathcal{V}) = W \setminus L(p)$;
- for $y \in Var$, we have $y^K(\mathcal{V}) = \mathcal{V}(y)$;
- $(\varphi_1 \wedge \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cap \varphi_2^K(\mathcal{V})$ and $(\varphi_1 \vee \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cup \varphi_2^K(\mathcal{V})$;
- $(\langle n, \alpha\rangle\varphi)^K(\mathcal{V}) = \{w : |\{w' \in W : (w, w') \in R(\alpha) \text{ and } w' \in \varphi^K(\mathcal{V})\}| \geq n + 1\}$;
- $([n, \alpha]\varphi)^K(\mathcal{V}) = \{w : |\{w' \in W : (w, w') \in R(\alpha) \text{ and } w' \notin \varphi^K(\mathcal{V})\}| \leq n\}$;
- $(\mu y.\varphi(y))^k(\mathcal{V}) = \bigcap\{W' \subseteq W : \varphi^K([y \leftarrow W']) \subseteq W'\}$;
- $(\nu y.\varphi(y))^k(\mathcal{V}) = \bigcup\{W' \subseteq W : W' \subseteq \varphi^K([y \leftarrow W'])\}$.

Notice that α used in the previous graded modalities is a program, i.e., α can be either an atomic program or its converse. Also, notice that no valuation is required for a sentence. Let $K = \langle W, W_0, R, L\rangle$ be a Kripke structure and φ a sentence. For a state $w \in W$, we say that K *satisfies* φ at w, denoted $K, w \models \varphi$, if $w \in \varphi^K$. K is a *model* of φ if there is a $w \in W_0$ such that $K, w \models \varphi$. In what follows, a formula φ *counts* up to b if the maximal integer in *atleast* and *allbut* restrictions used in φ is $b - 1$.

Given a formula φ of the enriched μ-calculus, accordingly to the forest model property, we can define a FEA accepting all forest models of φ. Before giving this result, there is a technical difficulty to be overcome: φ has quasi-forests as models, with labels on both edges and nodes, while FEAs can only accept forests with labels on nodes. This problem can be dealt in the following way. First, we move the label of each edge to the target node of the edge. For this purpose, we introduce a new propositional symbol p_α for each program α. Thus,

for each quasi-forest model $\langle F, V, E\rangle$, we consider the corresponding quasi-forest $\langle F, V'\rangle$ obtained by removing labeling on the edges and using as labeling of nodes the extended labeling function $V'(w) = V(w) \cup \{p_\alpha \mid E(v, w) = \alpha\}$. Then, to solve the problem of edges to the roots in quasi-forests models, we introduce in node labels new propositional symbols $\uparrow_o^\alpha$ (not occurring in the input formula) that represent an α-labeled edge from the current node to the (unique) root node labeled by nominal o. We call the new labeling function V^*, and with $\langle F, V^*\rangle$ we denote the forest encoding of the quasi-forest model $\langle F, V, E\rangle$. A forest $\langle F, V^*\rangle$ can also be considered as a particular Kripke structure by letting the total property holding in $\langle F, V^*\rangle$ if all leaves have a propositional symbol $\uparrow_o^\alpha$ in their labels. Now we can give the following result.

Lemma 2. *[SV01, BLMV06] Given a sentence φ of the enriched μ-calculus that has ℓ* atleast *subsentences, counts up to b, and contains k nominals, we can construct a FEA $\mathcal{A}_\varphi$ such that it accepts exactly the forest encodings of the quasi-forest models of φ having degree at most $\max\{k+1, \ell(b+1)\}$, and such that it has $\mathcal{O}(|\varphi|^2)$ states, index $|\varphi|$, and counting bound b.*

4 Enriched μ-Calculus Module Checking

In this paper we consider open systems, i.e., systems that interact with their environment and whose behavior depends on this interaction. The (global) behavior of such a system is described by a *module* $M = \langle W_s, W_e, W_0, R, L\rangle$, which is a Kripke structure where the set of states $W = W_s \cup W_e$ is partitioned in a set of *system states* W_s and a set of *environment states* W_e.

Given a module M, we assume that its states are ordered and the number of successors of each state w, denoted by $bd(w)$, is finite, and W is considered to be finite. For each state $w \in W$, we denote by $succ(w)$ the ordered tuple (possibly empty) of w's successors. When the module M is in a system state w_s, then all the states in $succ(w_s)$ are possible next states. On the other hand, when M is in an environment state w_e, then the possible next states (that are in $succ(w_e)$) depend on the current environment. Since the behavior of the environment is not predictable, we have to consider all the possible sub-tuples of $succ(w_e)$. The only constraint, since we consider environments that cannot block the system, is that not all the transitions from w_e are disabled.

The set of all (maximal) computations of M starting from the initial states W_0 is described by a $(W, Prog)$-labeled quasi-forest $\langle F_M, V_M, E_M\rangle$, called *computation quasi-forest*, which is obtained by unwinding M in the usual way. The problem of deciding, for a given branching-time formula φ over $AP \cup Nom$, whether $\langle F_M, L \circ V_M, E_M\rangle$ satisfies φ, denoted $M \models \varphi$, is the usual *model-checking problem* [CE81, QS81]. On the other hand, for an open system, $\langle F_M, V_M, E_M\rangle$ corresponds to a very specific environment, i.e., a maximal environment that never restricts the set of its next states. Therefore, when we examine a branching-time specification φ w.r.t. a module M, φ should hold not only in $\langle F_M, V_M, E_M\rangle$, but in all the quasi forests obtained by pruning from $\langle F_M, V_M, E_M\rangle$ subtrees whose root is a child (successor) of a node corresponding to an environment

state, as well as inhibiting some of its jumps to roots, if there are any. The set of these quasi forests is denoted by $exec(M)$, and is formally defined as follows. $\langle F, V, E\rangle \in exec(M)$ iff for each $w_i \in W_0$, we have $V(i) = w_i$, and the following holds:

- For $x \in F$ with $V(x) = w \in W_s$, $succ(w) = \langle w_1, \ldots, w_n, w_{n+1}, \ldots, w_{n+m}\rangle$, and $succ(w) \cap W_0 = \langle w_{n+1}, \ldots, w_{n+m}\rangle$, it holds that
  - $children(x) = \{x \cdot 1, \ldots, x \cdot n\}$ and for $1 \leq i \leq n$, $V(x \cdot i) = w_i$, and $E(x, x \cdot i) = \alpha$ if $(w, w_i) \in R(\alpha)$;
  - for $1 \leq i \leq m$, let $x_i \in \mathbb{N}$ such that $V(x_i) = w_{n+i}$, it holds that $E(x, x_i) = \alpha$ if $(w, w_{n+i}) \in R(\alpha)$;
- For $x \in F$ with $V(x) = w \in W_e$ it holds that there exists a sub-tuple $S = \langle w_{i_1}, \ldots, w_{i_p}, w_{i_{p+1}}, \ldots, w_{i_{p+q}}\rangle$ of $succ(w)$ with $p + q \geq 1$, $S \cap W_0 = \langle w_{i_{p+1}}, \ldots, w_{i_{p+q}}\rangle$ and such that
  - $children(x) = \{x \cdot 1, \ldots, x \cdot p\}$ and for $1 \leq j \leq p$, $V(x \cdot j) = w_{i_j}$, and $E(x, x \cdot j) = \alpha$ if $(w, w_{i_j}) \in R(\alpha)$;
  - for $1 \leq j \leq q$, let $x_j \in \mathbb{N}$ such that $V(x_j) = w_{i_{p+j}}$, it holds that $E(x, x_j) = \alpha$ if $(w, w_{i_{p+j}}) \in R(\alpha)$;

Intuitively, a quasi-forest in $exec(M)$ corresponds to a different behavior of the environment. In the following, we consider quasi-forests in $exec(M)$ as $(2^{AP \cup Nom}, Prog)$-labeled quasi-forests, i.e., taking the label of a node x to be $L(V(x))$.

For a module M and a formula φ of the enriched μ-calculus we say that M satisfies φ, denoted $M \models_r \varphi$, if all the quasi forests in $exec(M)$ satisfy φ. The problem of deciding whether M satisfies φ is called *module checking*, and extends to forests the analogously problem defined in [KVW01] regarding trees. Note that $M \models_r \varphi$ implies $M \models \varphi$, but the converse in general does not hold. Also, note that $M \not\models_r \varphi$ is *not* equivalent to $M \models_r \neg\varphi$, since $M \not\models_r \varphi$ just states that there is some quasi forest in $exec(M)$ satisfying $\neg\varphi$.

5 Deciding Enriched μ-Calculus Module Checking

In this section, we solve the module checking problem for the enriched μ-calculus. In particular, we show that this problem is decidable and ExpTime-complete. For the upper bound, we give an algorithm based on an automata-theoretic approach, by extending to forests and idea of [KVW01]. For the lower bound, we give a reduction from the module checking problem for *CTL*, known to be ExpTime-hard. We start with the upper bound.

Let M be a module and φ an enriched μ-calculus formula. We decide the module-checking problem for M against φ by building a FEA $\mathcal{A}_{M \times \neg\varphi}$ as the intersection of two automata. Essentially, the first automaton, denoted by $\mathcal{A}_M$, is a Büchi automaton that accepts forests encoding of labeled quasi-forests of $exec(M)$, and the second automaton is a FEA that accepts all the forests encoding of labeled quasi-forests that do not satisfy φ. Thus, $M \models_r \varphi$ iff $\mathcal{L}(\mathcal{A}_{M \times \neg\varphi})$ is empty.

The construction of $\mathcal{A}_M$ proposed here extends that given in [KVW01] for solving the module checking problem with respect to CTL and CTL^*. The extension concerns the handling of forest models instead of trees and formulas of the enriched μ-calculus. Before starting, there are few technical difficulties to be overcome. First, we notice that $exec(M)$ contains quasi-forests, with labels on both edges and nodes, while Büchi automata can only accept forests with labels on nodes. This problem can be dealt as we did in Section 3 by moving the label of each edge to the target node of the edge (formally using a new propositional symbol p_α, for each program α) and by substituting edges to roots with new propositional symbols $\uparrow_o^\alpha$ (which represent an α-labeled edge from the current node to the unique root node labeled by nominal o). Let $AP^* = AP \cup \{p_\alpha \mid \alpha \in Prog\} \cup \{\uparrow_o^\alpha \mid \alpha \in Prog \text{ and } o \in Nom\}$, we denote with $\langle F, V^* \rangle$ the $2^{AP^* \cup Nom}$-labeled forest encoding of a quasi-forest $\langle F, V, E \rangle \in exec(M)$, obtained using the above transformations.

Another technical difficulty to handle is relate to the fact that quasi-forests of $exec(M)$ (and thus their encoding) may not share the same structure, since they are obtained by pruning some subtrees from the computation quasi-forest $\langle F_M, V_M, E_M \rangle$ of M. Let $\langle F_M, V_M^* \rangle$ the *computation forest* of M obtained from $\langle F_M, V_M, E_M \rangle$ using the above encoding. By extending an idea of [KVW01], we solve the technical problem by considering each forest $\langle F, V^* \rangle$, encoding of a quasi-forest of $exec(M)$, as a $2^{AP^* \cup Nom} \cup \{\bot\}$-labeled forest $\langle F_M, V^{**} \rangle$ (where $\bot$ is a fresh proposition name not belonging to $AP \cup Nom$) such that for each node $x \in F_M$, if $x \in F$ then $V^{**}(x) = V^*(x)$, otherwise $V^{**}(x) = \{\bot\}$. Thus, we label each node pruned in the $\langle F_M, V_M^* \rangle$ with $\{\bot\}$ and recursively, we label with $\{\bot\}$ its subtrees. In this way, all forests encoding quasi-forests of $exec(M)$ have the same structure of $\langle F_M, V_M^* \rangle$, and they differ only in their labeling. Accordingly we can think of an environment as a strategy for placing $\{\bot\}$ in $\langle F_M, V^{**} \rangle$. Moreover, the environment can also disable jumps to roots. This is performed by removing from enabled environment nodes some of $\uparrow_o^\alpha$ labels. Notice that since we consider environments that do not block the system, each node associated with an environment state has at least one successor not labeled by $\{\bot\}$, unless it has $\uparrow_o^\alpha$ in its label.

Let us denote by $\widehat{exec}(M)$ the set of all $2^{AP^* \cup Nom} \cup \{\bot\}$-labeled $\langle F_M, V^{**} \rangle$ forests obtained from $\langle F, V, E \rangle \in exec(M)$ in the above described manner. The required BFA $\mathcal{A}_M$ must accept all and only the $2^{AP^* \cup Nom} \cup \{\bot\}$-labeled forests in $\widehat{exec}(M)$. Formally, let $M = \langle W_s, W_e, W_0, R, L \rangle$ be a module, $\mathcal{A}_M = \langle \Sigma, D, Q, \delta, Q_0, \mathcal{F} \rangle$ is defined as follows:

- $\Sigma = 2^{AP^* \cup Nom} \cup \{\bot\}$;
- $D = \bigcup_{w \in W} bd(w)$. That is D contains all the branching degrees in M.
- $Q = W \times \{\bot, \top, \vdash\}$. Thus every node w of M induces three states $(w, \bot)$, $(w, \top)$, and $(w, \vdash)$ in $\mathcal{A}_M$. Intuitively, when $\mathcal{A}_M$ is in state $(w, \bot)$, it can read only $\bot$, in state $(w, \top)$, it can read only letters in $2^{AP^* \cup Nom}$, and in state $(w, \vdash)$, then it can read both letters in $2^{AP^* \cup Nom}$ and $\bot$. In this last case, it is left to the environment to decide whether the transition to a state of the form $(w, \vdash)$ is enabled. The three types of states are used to ensure that

the environment enables all transitions from enabled system nodes, enables at least one transition from each enabled environment node, and disables transitions from disabled nodes.

- $Q_0 = \{\langle w_i, \top\rangle \mid w_i \in W_0\}$.
- The transition function $\delta : Q \times \Sigma \times D^2 \to 2^{Q^* \times (Q \times Root)^*}$ is defined as follows. Let $x \in F$ such that $V(x) = w$, $succ(w) = \langle w_1, \ldots, w_n, w'_1, \ldots, w'_m\rangle$, $succ(w) \cap W_0 = \langle w'_1, \ldots, w'_m\rangle$, and there exist $j_1, \ldots, j_m$ such that $V(j_h) = w'_h$, for $1 \leq h \leq m$, then:
  - For $w \in W_e \cup W_s$ and $g \in \{\vdash, \bot\}$ we have

$$\delta((w,g), \bot, n, 0) = \{\langle\langle(w_1, \bot), \ldots, (w_n, \bot)\rangle, \langle\emptyset\rangle\rangle\}$$

    That is, $\delta((w,g), \bot)$ contains exactly one n-tuple of all successors of w without jumps to roots. In this case, all transitions to successors of w are recursively disabled.
  - For $w \in W_s$ and $g \in \{\top, \vdash\}$ we have

$$\delta((w,g), L(w), n, m) = \langle\ \langle(w_1,\top), \ldots, (w_n,\top)\rangle, \langle(\langle w'_1, j_1\rangle,\top), \ldots, (\langle w'_m, j_m\rangle,\top)\rangle\ \rangle\}.$$

    That is, $\delta((w,g), m)$ contains exactly one $(n+m)$-tuple of all successors of w, containing m jumps to roots. In this case all transitions to successors of w are enabled.
  - For $w \in W_e$ and $g \in \{\top, \vdash\}$, let $J = \{\uparrow_o^\alpha \mid \alpha \in Prog \text{ and } o \in Nom\}$ and $X \subseteq L(w)$ such that $(X \setminus J) = (L(w) \setminus J)$, (i.e., X may have less jumps to roots of $L(w)$), we have
    - For $X \cap J = \emptyset$ we have

$$\begin{aligned}\delta((w,g), X, n, 0) = \{\ &\langle(w_1, \top),\ (w_2, \vdash), \ldots, (w_n, \vdash)\rangle,\\ &\langle(w_1, \vdash),\ (w_2, \top), \ldots, (w_n, \vdash)\rangle,\\ &\qquad\vdots\\ &\langle(w_1, \vdash),\ (w_2, \vdash), \ldots, (w_n, \top)\rangle\}.\end{aligned}$$

      That is, $\delta((w,g), X, n, 0)$ contains n different n-tuples of all successors of w, without jumps to roots. When $\mathcal{A}_M$ proceeds according to the ith tuple, the environment can disable all transitions to successors of w, except that to w_i.
    - For $X \cap J = \{\uparrow_{o_1}^{\alpha_1}, \ldots \uparrow_{o_s}^{\alpha_s}\}$ with $s \geq 1$, let $\langle w'_{j_1} \ldots w'_{j_s}\rangle$ a subtuple of $\langle w'_1 \ldots w'_m\rangle$ such that $o_i \in L(w'_{j_i})$, we have

$$\delta((w,g), X, n, s) = \{\langle\ \langle(w_1,\vdash), \ldots, (w_n,\vdash)\rangle, \langle(\langle w'_{j_1}, j_1\rangle, \top), \ldots, (\langle w'_{j_s}, j_s\rangle, \top)\rangle\ \rangle$$

      That is, $\delta((w,g), X, n, s)$ contains one $(n+s)$-tuple of successors of w, where the first n are all not root successors of w and they can be successively disabled.

Notice that δ is not defined when n is different from the number of successors of w that are not jumps to roots, and when the input does not meet the restriction imposed by the $\top$, $\vdash$, and $\bot$ annotations or by the labeling of w.

The automaton $\mathcal{A}_M$ has $3 \cdot |W|$ states, $2^{|AP| \cdot |R|}$ symbols, and the size of the transition relation $|\delta|$ is bounded by $|R|(|W| \cdot 2^{|R|})$.

We recall that a node labeled by $\{\bot\}$ stands for a node that actually does not exist. Thus, we have to take this into account when we interpret formulas of the enriched μ-calculus over forests $\langle F_M, V^* \rangle \in \widehat{exec}(M)$. In order to achieve this, as in [KVW01] we define a function f that transforms the input formula in a formula of the enriched μ-calculus that restricts path quantification to only paths that never visit a state labeled with $\{\bot\}$. The function f we consider extends that given in [KVW01] and is inductively defined as follows:

- $f(\mathbf{true}) = \mathbf{true}$ and $f(\mathbf{false}) = \mathbf{false}$;
- $f(p) = p$ and $f(\neg p) = \neg p$ for all $p \in AP \cup Nom$;
- $f(x) = x$ for all $x \in Var$;
- $f(\varphi_1 \vee \varphi_2) = f(\varphi_1) \vee f(\varphi_2)$ and $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$ for all enriched μ-calculus formulas φ_1 and φ_2;
- $f(\mu x.\varphi(x)) = \mu x.f(\varphi(x))$ and $f(\nu x.\varphi(x)) = \nu x.f(\varphi(x))$ for all $x \in Var$ and enriched μ-calculus formulas φ;
- $f(\langle n, \alpha \rangle \varphi) = \langle n, \alpha \rangle (\neg \bot \wedge f(\varphi))$ for $n \in \mathbb{N}$ and for all programs α and enriched μ-calculus formulas φ;
- $f([n, \alpha]\varphi) = [n, \alpha](\neg \bot \wedge f(\varphi))$ for $n \in \mathbb{N}$ and for all programs α and enriched μ-calculus formulas φ.

Note that the programs α in the previous definition of f can be either an atomic program $a \in Prog$ or its converse a^-. By definition of f, it follows that for each formula φ and $\langle F, V \rangle \in \widehat{exec}(M)$, $\langle F, V \rangle$ satisfies $f(\varphi)$ *iff* the $2^{AP^* \cup Nom}$-labeled tree obtained from $\langle F, V \rangle$ removing all the nodes labeled by $\{\bot\}$ satisfies φ. Therefore, the module checking problem of M against an enriched μ-calculus formula φ is reduced to check the existence of a forest $\langle F, V \rangle \in \widehat{exec}(M) = \mathcal{L}(\mathcal{A}_M)$ satisfying $f(\neg\varphi)$ (note that $|f(\neg\varphi)| = O(|\neg\varphi|)$). We reduce the latter to check the emptiness of a FEA $\mathcal{A}_{M \times \neg\varphi}$ that is defined as the intersection of the BFA $\mathcal{A}_M$ with a FEA $\mathcal{A}_{\neg\varphi}$ accepting exactly the $2^{AP^* \cup Nom} \cup \{\bot\}$ forests encodings of quasi-forest models of $f(\neg\varphi)$. By Lemma 2, if φ is an enriched μ-calculus formula, then $\mathcal{A}_{\neg\varphi}$ has $\mathcal{O}(|\varphi|^2)$ states, index $|\varphi|$, and counting bound b. Therefore, by Lemma 1, $\mathcal{A}_{M \times \neg\varphi}$ has $\mathcal{O}(|W| + |\varphi|^2)$ states, index $|\varphi|$, and counting bound b. We recall that, by Theorem 1, given a FEA, the emptiness is exponential only in its number of states and index, thus we have algorithm to decide the module checking problem for enriched μ-calculus formulas that is exponential both in the size of the module and the size of the formula.

To show a tight lower bound we recall that *CTL* module checking is ExpTime-hard [KVW01] and every *CTL* formula can be linearly transformed in a modal μ-calculus formula [Zap02]. This leads to the module checking problem w.r.t. modal μ-calculus formulas to be ExpTime-hard and thus to the following result

Theorem 2. *The module checking problem with respect to enriched μ-calculus formulas is* ExpTime*-complete.*

6 Fully Enriched μ-Calculus Module Checking

In this section, we deal with the module checking problem for the fully enriched μ-calculus and we show that it is undecidable.

Let us note that, since the fully enriched μ-calculus does not enjoy the forest model property [BP04], we cannot unwind a Kripke structure in a forest. However, it is always possible to unwind it in an equivalent acyclic graph that we call *computation graph*. In order to take into account all the possible behaviors of the environment, we consider all the possible subgraphs of the computation graph obtained disabling some transitions from environment nodes but one. We denote with $graphs(M)$ the set of this graphs. Given a fully enriched μ-calculus formula φ, we have that $M \models_r \varphi$ iff $K \models \varphi$ for all $K \in graphs(M)$.

To show the undecidability of the addressed problem, we need some further definitions. An (infinite) *grid* is a tuple $G = \langle \mathbb{N}^2, h, v \rangle$ such that h and v are defined as $h(\langle x, y \rangle) = \langle x + 1, y \rangle$ and $v(\langle x, y \rangle) = \langle x, y + 1 \rangle$. Given a finite set of *types* T, we will call *tiling* on T a function $\hat{\rho} : \mathbb{N}^2 \to T$ that associates a type from T to each vertex of an infinite grid G, and we call *tiled infinite grid* the tuple $\langle G, T, \hat{\rho} \rangle$. A *grid model* is an infinite Kripke structure $K = \langle W, \{w_0\}, R, L \rangle$, on the set of atomic programs $Prog = \{l^-, v\}$, such that K can be mapped on a grid in such a way that w_0 corresponds to the vertex $\langle 0, 0 \rangle$, $R(v)$ corresponds to v and $R(l^-)$ corresponds to h. We say that a grid model K "corresponds" to a tiled infinite grid $\langle G, T, \hat{\rho} \rangle$ if every state of K is labeled with only one atomic proposition (and zero or more nominals) and there exists a bijective function $\rho : T \to AP$ such that, if $w_{x,y}$ is the state of K corresponding with the node $\langle x, y \rangle$ of G, then $\rho(\hat{\rho}(\langle x, y \rangle)) \in L(w_{x,y})$.

Theorem 3. *The module checking problem for fully enriched μ-calculus is undecidable.*

Proof sketch. To show the result, we use a reduction from the domino problem, known to be undecidable [Ber66]. The domino problem is defined as follows.

Let T be a finite set of types, and $H, V \subseteq T^2$ be two relations describing the types that cannot be vertically and horizontally adjacent in an infinite grid. The domino problem is to decide whether there exists a tiled infinite grid $\langle G, T, \hat{\rho} \rangle$ such that $\hat{\rho}$ preserves the relations H and V. We call such a tiling function a *legal tiling* for G on T.

In [BP04], Bonatti and Peron showed undecidability for the satisfiability problem for fully enriched μ-calculus by also using a reduction from the domino problem. Hence, given a set of types T and relations H and V, they build a (alternation free) fully enriched μ-calculus formula φ such that φ is satisfiable iff the domino problem has a solution in a tiled infinite grid, with a legal tiling ρ on T (with respect to H and V). In particular, the formula they build can be only satisfiable on a grid model K corresponding to a tiled infinite grid with a legal tiling ρ on T. In the reduction we propose here, we use the formula φ used in [BP04]. It remains to define the module.

Let $\{G_1, G_2, \ldots\}$ be the set of all the infinite tiled grids on T (i.e., $G_i = \langle G, T, \hat{\rho}_i \rangle$), we build a module M such that $graphs(M)$ contains, for each $i \geq 1$,

a grid models corresponding to G_i. Therefore, we can decide the domino problem by checking whether $M \models_r \neg\varphi$. Indeed, if $M \models_r \neg\varphi$, then all grid models corresponding to G_i do not satisfy φ and, therefore, there is no solution for the domino problem. On the other side, if $M \not\models_r \neg\varphi$, then there exists a model for φ; since φ can be satisfied only on a grid model corresponding to a tiled infinite grid with a legal tiling on T with respect to H and V, we have that the domino problem has a solution.

Formally, let $T = \{t_1, \ldots, t_m\}$ be the set of types, the module $M = \langle W_s, W_e, W_0, R, L\rangle$ with respect to atomic programs $Prog = \{l^-, v\}$, atomic propositions $AP = T$, and nominals $Nom = \{o_1, \ldots, o_m\}$, is defined as follows:

- $W_s = \emptyset$, $W_e = \{x_1, \ldots, x_m, y_1, \ldots, y_m\}$ and $W_0 = \{x_1, \ldots, x_m\}$;
- for all $i \in \{1, \ldots m\}$, $L(t_i) = \{x_i, y_i\}$ and $L(o_i) = \{x_i\}$;
- $R(v) = \{\langle x_i, x_j\rangle | i, j \in \{1, \ldots, m\}\} \cup \{\langle y_i, y_j\rangle | i, j \in \{1, \ldots, m\}\}$ and $R(l^-) = \{\langle x_i, y_j\rangle | i, j \in \{1, \ldots, m\}\} \cup \{\langle y_i, x_j\rangle | i, j \in \{1, \ldots, m\}\}$

Notice that we duplicate the set of nodes labeled with tiles since we cannot have pairs of nodes in M labeled with more than one atomic program (in our case, with both v and l^-). Moreover the choice of labeling nodes x_i with nominals is arbitrary. Finally, from the fact that the module contains only environment nodes, it immediately follows that, for each i, the grid model corresponding to the infinite tiled grid G_i is contained in $graphs(M)$. □

7 Conclusions

In [KVW01], module checking has been introduced as a useful framework for the verification of open finite-state systems. There, it has been shown that while for *LTL* the complexity of the model checking problem coincides with that of module checking (i.e., it is PSPACE-complete), for the branching time paradigm the problem of module checking is much harder. In fact, *CTL* (resp., *CTL**) module checking is EXPTIME-complete (resp., 2EXPTIME-complete), while model checking is solvable in linear time (resp., exponential time).

In this paper, we have extended the framework of module checking problem for finite-state systems to formulas of the fully enriched μ-calculus and showed that this problem becomes undecidable in this setting. Also, we have investigated this problem with respect to formulas of interesting fragments of the full calculus and, specifically, those belonging to the full hybrid, full graded, and hybrid graded μ-calculus, and showed, in all cases, that module checking is decidable and EXPTIME-complete. In particular, for the upper bound we have proposed an algorithm that is exponential in both the size of the model and the formula. Since module checking for μ-calculus subsumes that for *CTL* and for the latter the program complexity (i.e., the complexity of the problem w.r.t. a fixed formula) is polynomial, it remains as an open problem to decide the exact program complexity of module checking for the considered fragments of the full calculus.

Finally, we recall that model checking for modal μ-calculus is in UP∩CO-UP (see [Zap02] for a survey). Since we have proved that module checking for

modal μ-calculus is ExpTime-hard, we conclude that also for this logic module checking is harder than model checking. Moreover, the model checking algorithm considered in [Zap02] for modal μ-calculus can be easily extended to deal with formulas of the fully enriched μ-calculus, showing that also for this logic the model checking problem is in UP$\cap$co-UP. Using this conjecture, we can extend to the full calculus and its fragments all the previous observations regarding the modal μ-calculus.

References

[Ber66] R. Berger, *The undecidability of the domino problem*, Mem. AMS **66** (1966), 1–72.

[BLMV06] P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi, *The complexity of enriched mu-calculi*, ICALP'06, LNCS 4052, 2006, pp. 540–551.

[BMP05] Laura Bozzelli, Aniello Murano, and Adriano Peron, *Pushdown module checking.*, LPAR, 2005, pp. 504–518.

[BP04] P.A. Bonatti and A. Peron, *On the undecidability of logics with converse, nominals, recursion and counting*, Artificial Intelligence **158** (2004), no. 1, 75–96.

[BS06] J. Bradfield and C. Stirling, *Modal μ-calculi*, Handbook of Modal Logic (Blackburn, Wolter, and van Benthem, eds.), Elsevier, 2006, pp. 722–756.

[CE81] E.M. Clarke and E.A. Emerson, *Design and synthesis of synchronization skeletons using branching time temporal logic*, Proc. of Work. on Logic of Programs, LNCS 131, 1981, pp. 52–71.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model checking*, MIT Press, Cambridge, MA, USA, 1999.

[Hoa85] C.A.R. Hoare, *Communicating sequential processes*, Prentice-Hall International. Upper Saddle River, NJ, USA, 1985.

[HP85] D. Harel and A. Pnueli, *On the development of reactive systems*, Logics and Models of Concurrent Systems, NATO Advanced Summer Institutes, vol. F-13, Springer-Verlag, 1985, pp. 477–498.

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

[KSV02] O. Kupferman, U. Sattler, and M.Y. Vardi, *The complexity of the graded μ-calculus*, CADE'02, LNAI 2392, 2002, pp. 423–437.

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper, *Module checking*, Information and Computation **164** (2001), 322–344.

[QS81] J.P. Queille and J. Sifakis, *Specification and verification of concurrent systems in cesar*, Proc. of 5^{th} Int. Symposium on Programming, LNCS 137, 1981, pp. 337–351.

[SV01] U. Sattler and M.Y. Vardi, *The hybrid mu-calculus*, IJCAR'01, LNAI 2083, 2001, pp. 76–91.

[Var98] M.Y. Vardi, *Reasoning about the past with two-way automata*, ICALP'98, LNCS 1443, 1998, pp. 628–641.

[VW86] M.Y. Vardi and P. Wolper, *An automata-theoretic approach to automatic program verification (preliminary report)*, LICS '86, 1986, pp. 332–344.

[Zap02] J. Zappe, *Modal μ-calculus and alternating tree automata*, Automata, Logics, and Infinite Games (E. Grädel, W. Thomas, and T. Wilke, eds.), LNCS, vol. 2500, Springer, 2002, pp. 171–184.