Enriched μ–Calculus Pushdown Module Checking*

Alessandro Ferrante[1], Aniello Murano[2], and Mimmo Parente[1]

[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 systems (called *module checking*) has been intensively studied in the literature, both for finite–state and infinite–state systems. In this paper, we focus on pushdown module checking with respect to μ–calculus enriched with graded and nominals (*hybrid graded μ-calulus*). We show that this problem is decidable and solvable in double–exponential time in the size of the formula and in exponential time in the size of the system. This result is obtained by exploiting a classical automata–theoretic approach via *pushdown nondeterministic parity tree automata*. In particular, we reduce in exponential time our problem to the emptiness problem for these automata, which is known to be decidable in EXPTIME. As a key step of our algorithm, we show an exponential improvement of the construction of a nondeterministic parity tree automaton accepting all models of a formula of the considered logic. This result, not only allows our algorithm to match the known lower bound, but it is also interesting by itself, since it allows investigating decision problems related to enriched μ-calculus formulas in a greatly simplified manner. We conclude the paper with a discussion on the model checking w.r.t. μ-calculus formulas enriched with backward modalities as well.

1 Introduction

In system design, one of the most challenging problems is to check for system correctness. *Model-checking* is a formal method that allows us to automatically verify, in a suitable way, the ongoing behaviors of *reactive systems* ([CE81, QS81]). In this verification technique (for a survey, see [CGP99]), the behavior of a system, formally described by a mathematical model, is checked against a behavioral constraint, possibly specified by a formula in an appropriate temporal logic.

In system modeling, we distinguish between *closed* and *open* systems [HP85]. While the behavior of a closed system is completely determined by the state of the system, the behavior of an open system depends on the ongoing interaction with its environment [Hoa85]. Model checking algorithms used for the verification of closed systems are not appropriate for open systems. In the latter case, we

* Work partially supported by MIUR FIRB Project no. RBAU1P5SS and the grant *Metodi per la Verifica di Sistemi Software e Real-Time* from University of Salerno.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 438–453, 2007.

should check the system with respect to arbitrary environments and take into account uncertainty regarding the environment. In [KVW01], model checking has been extended 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 *system states* (where the system makes a transition) and *environment states* (where the environment makes a transition). Given a module $\mathcal{M}$, describing the system to be verified, and a temporal logic formula φ, specifying the desired behavior of the system, the problem of model checking a module, called *module checking*, asks whether for all possible environments, $\mathcal{M}$ satisfies φ. Module checking thus involves not only checking that the full computation tree $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ obtained by unwinding $\mathcal{M}$ (which corresponds to the interaction of $\mathcal{M}$ with a maximal environment) satisfies φ (which corresponds to model checking $\mathcal{M}$ with respect to φ), but also that all trees obtained from $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$, by pruning subtrees of environment nodes (these trees correspond to all possible choices of the environment and are collected in $exec(\mathcal{M})$) satisfy φ. To see an example, consider a two-drink dispenser machine that serves, upon request, tea or coffee. The machine is an open system and an environment for the system is an infinite line of thirsty people. Since each person in the line can prefer both tea and coffee, or only tea, or only coffee, each person suggests a different disabling of the external nondeterministic choices. Accordingly, there are many different possible environments to consider. In [KVW01], it has been shown that while for linear–time logics model and module checking coincide, module checking for specifications given in *CTL* and *CTL** is exponentially harder than model checking. Indeed, *CTL* and *CTL** module checking is EXPTIME–complete and 2EXPTIME–complete in the size of the formula, respectively, and both PTIME–complete in the size of the system.

Recently, finite-state module checking has been also investigated with respect to formulas of the *hybrid graded μ–calculus* [FM07], a powerful decidable fragment of the *fully enriched μ-calculus* [BP04, BLMV06]. The *μ–calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. Fully enriched μ–calculus is the extension of the μ–calculus with *inverse programs*, *graded modalities*, and *nominals*. Intuitively, inverse programs allow us 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 of a state [KSV02]. By dropping at least one of the additional constructs, we get a *fragment* of the fully enriched μ-calculus. In particular, by inhibiting backward modalities we get the fragment we call hybrid graded μ-calculus. In [BP04], it has been shown that satisfiability is undecidable in the *fully enriched μ–calculus*. On the other hand, it has been shown in [SV01, BLMV06] that satisfiability for any of its fragments is decidable and EXPTIME-complete. The upper bound result is based on an automata–theoretic approach via *two-way graded alternating parity tree automata (*2GAPT*)*. Intuitively, these automata generalize alternating automata on infinite trees as inverse programs and graded modalities enrich the standard μ–calculus: $2GAPT$ can move up to a node's predecessor and move down to

at least n or *all but n* successors. Using these automata, along with the fact that each fragment of the fully enriched μ-calculus enjoys the *quasi-forest model property*[1], it has been shown in [SV01, BLMV06] that given a formula φ of a fragment logic, it is possible to construct a $2GAPT$ accepting all trees encodings quasi-forests[2] modeling φ. Then, the exponential-upper bound follows from the fact that $2GAPT$ can be exponentially translated in *nondeterministic graded parity tree automata* ($GNPT$), and the emptiness problem for $GNPT$ is solvable in PTIME [KPV02].

Coming back to the finite-state module checking problem for the hybrid graded μ–calculus, this problem has been shown in [FM07] to be EXPTIME–complete. To see an example of its application, consider the previous two-drink dispenser machine such that whenever a customer can choose a drink he can also call a customer service, among $k > 1$ different services. Suppose also that by taking a customer service choice the drink-dispenser machine stops dispensing drinks unless the customer service resets the machine. Suppose now we want to check the property that whenever the customer comes at a choice he can always choose among k different services. This property can be described using a formula of the hybrid graded μ–calculus, whose truth depends on the possibility of jumping to nodes, each labeling the start interaction with a particular service (using nominals), and having exactly k identical of such nodes (using graded modalities). Clearly, such an open system does not satisfy this formula. Indeed, it is not satisfied by the particular behavior that chooses always the same service.

In [BMP05], the module checking technique has been also extended to infinite-state systems by considering *open pushdown systems* (OPD, for short). These are pushdown systems augmented with finite information that allows us to partition the set of configurations (in accordance with the control state and the symbol on the top of the stack) into *system configurations* and *environment configurations*. To see an example of an open pushdown system, consider an extension of the above mentioned two-drink dispenser machine, with the additional constraint that a coffee can be served only if the number of coffees served up to that time is smaller than that of teas served. Such a machine can be clearly modeled as an open pushdown system (the stack is used to guarantee the inequality between served coffees and teas). In [BMP05], it has been shown that pushdown module checking is 2EXPTIME–complete for CTL and 3EXPTIME–complete for CTL^*.

In this paper, we extend the pushdown module checking problem to the hybrid graded μ-calculus and, by exploiting an automata-theoretic approach via pushdown tree automata, we show that this problem is decidable and solvable in 2EXPTIME. In particular, we reduce the addressed problem to the emptiness problem for pushdown tree automata. The algorithm we propose works as follows. Given an OPD $\mathcal{S}$, a module $\mathcal{M}$ induced by the configurations of $\mathcal{S}$, and an hybrid graded μ-calculus formula φ, we first construct in polynomial time a pushdown Büchi tree automaton (PD–NBT) $\mathcal{A}_{\mathcal{M}}$, accepting $exec(\mathcal{M})$. The

[1] A quasi forest is a forest where nodes can have roots as successors.

[2] Encoding is done by using a new root node that connects all roots of the quasi-forest and new atomic propositions which are used to encode programs and jumps to roots.

construction of $\mathcal{A}_{\mathcal{M}}$ we propose here extends that used in [BMP05] by also taking into account that $\mathcal{M}$ must be unwound in a quasi-forest, rather than a tree, with both nodes and edges labeled. Thus, the set $exec(\mathcal{M})$ is a set of quasi-forests, and the automaton $\mathcal{A}_{\mathcal{M}}$ we construct will accept all trees encodings of all quasi-forests of $exec(\mathcal{M})$. From the formula side, accordingly to [BLMV06], we can construct in a polynomial time a $GAPT$ $\mathcal{A}_{\neg\varphi}$ accepting all models of $\neg\varphi$, with the intent of checking that no models of $\neg\varphi$ are in $exec(\mathcal{M})$. Thus, we check that $\mathcal{M}$ models φ for every possible choice of the environment by checking whether $\mathcal{L}(\mathcal{A}_{\mathcal{M}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. To the best of our knowledge, the latter problem can only be solved in triple-exponential time. For example, by using a double-exponential translation of $GAPT$ into nondeterministic parity tree automata (NPT) [BLMV06, KSV02] and the fact that the emptiness problem for the intersection of a PD–NBT and an NPT is solvable in Exptime [KPV02]. Here, by showing a non-trivial exponential reduction of $2GAPT$ into NPT, we show a 2Exptime upper bound for the addressed problem. Since the pushdown module checking problem for CTL is 2Exptime-hard, we get that the addressed problem is then 2Exptime-complete. The exponential improvement on translating $2GAPT$ into NPT does not only allow us to match the known lower bound, but it also turns out to be useful in several automata-theoretic approaches to system verification. In particular, it also allows us to get results concerning decision problems for the hybrid μ–calculus (such as the satisfiability and module checking problems [BLMV06, FM07]) in a simplified way.

The rest of the paper is organized as follows. In the next section, we give preliminaries on labeled forests, hybrid graded μ-calculus, open Kripke structures, and open pushdown systems. In section 3, we recall $2GAPT$ and NPT. In section 4, we show an exponential translation of $2GAPT$ into NPT. In section 5, we give the module checking algorithm and conclude in section 6 with a discussion on model checking w.r.t. fragments of fully enriched μ-calculus also including the backward modality.

2 Preliminaries

Labeled Forests. For a finite set X, we denote the *size* of X by $|X|$, the set of 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 a of X, we use $w \cdot a$ to denote the word wa. Let $\mathbb{N}$ be the set of positive integers. For $n \in \mathbb{N}$, let $\mathbb{N}$ 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 words consisting of a single natural number are *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 at* r). For $x \in F$, the nodes $x \cdot c \in F$ where $c \in \mathbb{N}$ are the *successors* of x, denoted $sc(x)$, and x is their *predecessor*. The number of successors of a node x is called the *degree* of x ($deg(x)$). The degree h of a forest F is the maximum of the degrees of all nodes in F and the number of roots. A forest with degree h is an *h-ary* forest. A full h-ary forest is a forest having h roots and all nodes with degree h.

Let $F \subseteq \mathbb{N}^+$ be a forest, x a node in F, and $n \in \mathbb{N}$. As a convention, we take $x \cdot \varepsilon = \varepsilon \cdot x = x$, $(x \cdot c) \cdot -1 = x$, and $n \cdot -1$ as undefined. We call x a *leaf* if it has no successors. A *path* π in F is a word $\pi = x_1 x_2 \ldots$ of F such that x_1 is a root of F and for every $x_i \in \pi$, either x_i is a leaf (i.e., π ends in x_i) or x_i is a predecessor of x_{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 $y \in sc(x)$, 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. For a node x of a quasi–forest, we set $children(x)$ as $sc(x) \setminus \mathbb{N}$. All the other definitions regarding forests easily extend to quasi–forests. Notice that in a quasi–forest, since each node can have a root as successor, a root can also have several predecessors, while every other node has just one. Clearly, a quasi–forest can always be transformed into a forest by removing root successors.

Hybrid Graded μ–Calculus. Let AP, Var, $Prog$, and Nom be finite and pairwise disjoint sets of *atomic propositions*, *propositional variables*, *(atomic) programs*, and *nominals*. The set of *hybrid graded μ–calculus* formulas 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, $\alpha \in Prog$, n is a non negative integer, and $y \in Var$, then $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\langle n, \alpha \rangle \varphi_1$, $[n, \alpha] \varphi_1$, $\mu y.\varphi_1(y)$, and $\nu y.\varphi_1(y)$ are also formulas. Observe that we use positive normal form, i.e., negation is applied only to atomic propositions.

We call μ and ν *fixpoint operators*. A propositional variable y occurs *free* in a formula if it is not in the scope of a fixpoint operator. 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 respectively *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.

The semantics of the hybrid graded μ–calculus is defined with respect to a *Kripke structure*, i.e., a tuple $\mathcal{K} = \langle W, W_0, R, L \rangle$ 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 function 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. 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 $\mathcal{K}$ if φ holds in at least $n+1$ α–successors of w. Dually, the *allbut* formula $[n, \alpha] \varphi$ holds in a state w of $\mathcal{K}$ 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 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 $\mathcal{K} = \langle W, W_0, R, L \rangle$ 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 $\mathcal{K}$ as a mapping $\varphi^{\mathcal{K}}$ from valuations to 2^W, i.e., $\varphi^{\mathcal{K}}(\mathcal{V})$ denotes the set of points that satisfy φ under valuation $\mathcal{V}$. The mapping $\varphi^{\mathcal{K}}$ is defined inductively as follows:

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

For a state w of a Kripke structure $\mathcal{K}$, we say that $\mathcal{K}$ *satisfies* φ at w if $w \in \varphi^{\mathcal{K}}$. In what follows, a formula φ *counts* up to b if the maximal integer in *atleast* and *allbut* formulas used in φ is $b - 1$.

Open Kripke Structures. 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* $\mathcal{M} = \langle W_s, W_e, W_0, R, L \rangle$, which is a Kripke structure where the set of states $W = W_s \cup W_e$ is partitioned in *system states* W_s and *environment states* W_e.

Given a module $\mathcal{M}$, we assume that its states are ordered and the number of successors of each state w is finite. For each $w \in W$, we denote by $succ(w)$ the ordered tuple (possibly empty) of w's α-successors, for all $\alpha \in Prog$. When $\mathcal{M}$ is in a system state w_s, then all states in $succ(w_s)$ are possible next states. On the other hand, when $\mathcal{M}$ is in an environment state w_e, 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 $\mathcal{M}$ starting from W_0 is described by a $(W, Prog)$–labeled quasi–forest $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$, called *computation quasi–forest*, which is obtained by unwinding $\mathcal{M}$ in the usual way. The problem of deciding, for a given branching–time formula φ over $AP \cup Nom$, whether $\langle F_{\mathcal{M}}, L \circ V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ satisfies φ at a root node, denoted $\mathcal{M} \models \varphi$, is the usual *model–checking problem* [CE81, QS81]. On the other hand, for an open system $\mathcal{M}$, the quasi–forest $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ 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 formula φ w.r.t. $\mathcal{M}$, the formula φ should hold not only in $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$, but in all quasi-forests obtained by pruning from $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ subtrees rooted at children of environment nodes, as well as inhibiting some of their jumps to roots, if there are any. The set of these quasi–forests, which collects all possible behaviors of the environment, is denoted by $exec(\mathcal{M})$ and is formally defined as follows. A quasi–forest $\langle F, V, E \rangle \in exec(\mathcal{M})$ iff for each $w_i \in W_0$, we have $V(i) = w_i$, and for $x \in F$, with $V(x) = w$, $succ(w) =$

$\langle w_1, \ldots, w_n, w_{n+1}, \ldots, w_{n+m} \rangle$, and $succ(w) \cap W_0 = \langle w_{n+1}, \ldots, w_{n+m} \rangle$, there exists $S = \langle w'_1, \ldots, w'_p, w'_{p+1}, \ldots, w'_{p+q} \rangle$ sub-tuple of $succ(w)$ such that $p + q \geq 1$, $S = succ(w)$ if $w \in W_s$ and the following hold:

- $children(x) = \{x \cdot 1, \ldots, x \cdot p\}$ and, for $1 \leq j \leq p$, we have $V(x \cdot j) = w'_j$ and $E(x, x \cdot j) = \alpha$ if $(w, w'_j) \in R(\alpha)$;
- for $1 \leq j \leq q$, let $x_j \in \mathbb{N}$ such that $V(x_j) = w'_{p+j}$, then $E(x, x_j) = \alpha$ if $(w, w'_{p+j}) \in R(\alpha)$.

In the following, we consider quasi–forests in $exec(\mathcal{M})$ as labeled with $(2^{AP \cup Nom}, Prog)$, i.e., the label of a node x is $L(V(x))$. For a module $\mathcal{M}$ and a formula φ of the hybrid graded μ–calculus, we say that $\mathcal{M}$ *reactively* satisfies φ, denoted $\mathcal{M} \models_r \varphi$, if all quasi-forests in $exec(\mathcal{M})$ satisfy φ. The problem of deciding whether $\mathcal{M} \models_r \varphi$ is called *hybrid graded μ–calculus module checking*.

Open Pushdown Systems (*OPD*). An *OPD* over $AP \cup Nom \cup Prog$ is a tuple $\mathcal{S} = \langle Q, \Gamma, \flat, C_0, \Delta, \rho_1, \rho_2, Env \rangle$, where Q is a finite set of (control) *states*, Γ is a finite *stack alphabet*, $\flat \notin \Gamma$ is the *stack bottom symbol*. We set $\Gamma_\flat = \Gamma \cup \{\flat\}$, $Conf = Q \times (\Gamma^* \cdot \flat)$ to be the set of *(pushdown) configurations*, and for each configuration $(q, A \cdot \gamma)$, we set $top((q, A \cdot \gamma)) = (q, A)$ to be a *top configuration*. The function $\Delta : Prog \to 2^{(Q \times \Gamma_\flat) \times (Q \times \Gamma_\flat^*)}$ is a finite set of transition rules such that $\flat$ is always present at the bottom of the stack and nowhere else (thus whenever $\flat$ is read, it is pushed back). Note that we make this assumption also about the various pushdown automata we use later. The set $C_0 \subseteq Conf$ is a finite set of *initial configurations*, $\rho_1 : AP \to 2^{Q \times \Gamma_\flat}$ and $\rho_2 : Nom \to C_0$ are labeling functions associating respectively to each atomic proposition p a set of top configurations in which p holds and to each nominal exactly one initial configuration. Finally, $Env \subseteq Q \times \Gamma_\flat$ specifies the set of *environment configurations*. The size $|\mathcal{S}|$ of $\mathcal{S}$ is $|Q| + |\Delta| + |\Gamma|$.

The *OPD* moves in accordance with the transition relation Δ. Thus, $((q, A), (q', \gamma)) \in \Delta(\alpha)$ implies that if the *OPD* is in state q and the top of the stack is A, it can move along with an *α–transition* to state q', and substitute γ for A. Also note that the possible operations of the system, the labeling functions, and the designation of configurations as environment configurations, are all dependent only on the current control state and the top of the stack.

An *OPD* $\mathcal{S}$ induces a module $\mathcal{M}_\mathcal{S} = \langle W_s, W_e, W_0, R, L \rangle$, where:

- $W_s \cup W_e = Conf$, i.e. the set of pushdown configurations, and $W_0 = C_0$;
- $W_e = \{c \in Conf \mid top(c) \in Env\}$.
- $((q, A \cdot \gamma), (q', \gamma' \cdot \gamma)) \in R(\alpha)$ *iff* there is $((q, A), (q', \gamma')) \in \Delta(\alpha)$;
- $L(p) = \{c \in Conf \mid top(c) \in \rho_1(p)\}$ for $p \in AP$; $L(o) = \rho_2(o)$ for $o \in Nom$.

The *hybrid graded (μ-calculus) pushdown module checking* problem is to decide, for a given *OPD* $\mathcal{S}$ and an enriched μ–calculus formula φ, whether $\mathcal{M}_\mathcal{S} \models_r \varphi$.

3 Tree Automata

Two-way Graded Alternating Parity Tree Automata ($2GAPT$). These automata are an extension of nondeterministic tree automata in such a way that a $2GAPT$ can send several copies of itself to the same successor (*alternating*), send copies of itself to the predecessor (*two-way*), specify a number n of successors to which copies of itself are sent without specifying which successors these exactly are (*graded*), and accept trees along with a *parity condition*, cf. [BLMV06]. To give a formal definition, let us start with some technicalities.

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 $D_b = \langle [b] \rangle \cup [[b]] \cup \{-1, \varepsilon\}$.

Formally, a $2GAPT$ on Σ-labeled trees 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, $q_0 \in Q$ is an initial state, and $\mathcal{F}$ is a parity acceptance condition (see below). Intuitively, an atom $(\langle n \rangle, q)$ (resp. $([n], q)$) means that $\mathcal{A}$ sends copies in state q to $n+1$ (resp. all but n) different successors of the current node, (ε, q) means that $\mathcal{A}$ sends a copy (in state q) to the current node, and $(-1, q)$ means that $\mathcal{A}$ sends a copy to the predecessor of the current node. A *run* of $\mathcal{A}$ on an input Σ-labeled tree $\langle T, V \rangle$ is a tree $\langle T_r, r \rangle$ in which each node is labeled by an element of $T \times Q$. 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 T. 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) $r(z) = (1, q_0)$ for the root 1 of T and (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 $d = \langle n \rangle$ (resp., $d = [n]$), there are at least $t = n+1$ (resp., $t = deg(x) - n$) distinct indexes $i_1, \ldots, i_t$ such that for all $1 \leq j \leq t$, there is $j' \in \mathbb{N}$ such that $y \cdot j' \in T_r$, $x \cdot i_j \in T$, and $r(y \cdot j') = (x \cdot i_j, s)$.

Note that if $\theta =$ **true**, then y does not need to have successors. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta =$ **false**, we cannot have a run that takes a transition with $\theta =$ **false**.

A run $\langle T_r, r \rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. In the parity acceptance condition, $\mathcal{F}$ is a set $\{F_1, \ldots, F_k\}$ such that $F_1 \subseteq \ldots \subseteq F_k = Q$ and k is called the *index* of the automaton. An infinite path π on T_r *satisfies* $\mathcal{F}$ if there is an even i such that π contains infinitely many states from F_i and finitely many states from F_{i-1}. An automaton *accepts* a tree iff there exists an accepting run of the automaton on the tree. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

A $2GAPT$ is a $GAPT$ if $\delta : Q \times \Sigma \to B^+(D_b \setminus \{-1\} \times Q)$ and a $2APT$ if $\delta : Q \times \Sigma \to B^+(\{-1, \varepsilon, 1, \ldots, h\} \times Q)$. Moreover, it is an NPT if $\delta : Q \times \Sigma \to B^+(\{1, \ldots, h\} \times Q)$ and the transition relation δ is in disjunctive normal form, where in each conjunct each direction appears at most once [KVW00]. We now recall a result on $GAPT$ and hybrid graded μ-calculus formulas.

Lemma 1 ([BLMV06]). *Given an hybrid graded μ-calculus sentence φ with ℓ* atleast *subsentences and counting up to b, it is possible to construct a* GAPT *with $\mathcal{O}(|\varphi|^2)$ states, index $|\varphi|$, and counting bound b that accepts exactly each tree that encodes a quasi-forest modeling φ having degree at most* $\max\{|Nom|+1, \ell(b+1)\}$.

Nondeterministic Pushdown Parity Tree Automata (*PD–NPT*). A *PD–NPT* (without ε-transitions), on Σ-labeled full h-ary trees, is a tuple $\mathcal{P} = \langle \Sigma, \Gamma, \flat, Q, q_0, \gamma_0, \rho, \mathcal{F} \rangle$, where Σ is a finite input alphabet, Γ, $\flat$, $\Gamma_\flat$, and Q are as in OPD, (q_0, γ_0) is the initial configuration, $\rho : Q \times \Sigma \times \Gamma_\flat \to 2^{(Q \times \Gamma_\flat^*)^h}$ is a transition function, and $\mathcal{F}$ is a parity condition over Q. Intuitively, when $\mathcal{P}$ is in state q, reading an input node x labeled by $\sigma \in \Sigma$, and the stack contains a word $A \cdot \gamma \in \Gamma^* \cdot \flat$, then $\mathcal{P}$ chooses a tuple $\langle (q_1, \gamma_1), \ldots, (q_h, \gamma_h) \rangle \in \rho(q, \sigma, A)$ and splits in h copies such that for each $1 \leq i \leq h$, a copy in configuration $(q_i, \gamma_i \cdot \gamma)$ is sent to the node $x \cdot i$ in the input tree. A run of $\mathcal{P}$ on a Σ-labeled full h-ary tree $\langle T, V \rangle$ is a $(Q \times \Gamma^* \cdot \flat)$-labeled tree $\langle T, r \rangle$ such that $r(\varepsilon) = (q_0, \gamma_0)$ and for each $x \in T$ with $r(x) = (q, A \cdot \gamma)$, there is $\langle (q_1, \gamma_1), \ldots, (q_h, \gamma_h) \rangle \in \rho(q, V(x), A)$ where, for all $1 \leq i \leq h$, we have $r(x \cdot i) = (q_i, \gamma_i \cdot \gamma)$. The notion of accepting path is defined with respect to the control states that appear infinitely often in the path (thus without taking into account any stack content). Then, the notions given for $2GAPT$ regarding accepting runs, accepted trees, and accepted languages, along with the parity condition, easily extend to *PD–NPT*. We also consider Büchi condition $\mathcal{F} \subseteq Q$, which simply is a special parity condition $\{\emptyset, \mathcal{F}, Q\}$. In the following, we denote with *PD–NBT* a *PD–NPT* with a Büchi condition. The *emptiness* problem for an automaton $\mathcal{P}$ is to decide whether $\mathcal{L}(\mathcal{P}) = \emptyset$. We now recall two useful results on the introduced automata.

Proposition 1 ([KPV02]). *The emptiness problem for a* PD–NPT *on Σ-labeled full h-ary trees, having index m, n states, and transition function ρ, can be solved in time exponential in $n \cdot m \cdot h \cdot |\rho|$.*

Proposition 2 ([BMP05]). *On Σ-labeled full h-ary trees, given a* PD–NBT *$\mathcal{P} = \langle \Sigma, \Gamma, Q, q_0, \gamma_0, \rho, Q \rangle$ and an* NPT *$\mathcal{A} = \langle \Sigma, Q', q_0', \delta, \mathcal{F}' \rangle$, there is a* PD–NPT *$\mathcal{P}'$ such that $\mathcal{L}(\mathcal{P}') = \mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\mathcal{A})$. Moreover, $\mathcal{P}'$ has $|Q| \cdot |Q'|$ states, the same index as $\mathcal{A}$, and the size of the transition relation is bounded by $|\rho| \cdot |\delta| \cdot h$.*

4 From $2GAPT$ to NPT

In this section, we give a nontrivial exponential-time translation from $2GAPT$ to NPT. To the best of our knowledge this exponentially improves the known result from the literature, e.g. using results from [BLMV06, KSV02] one can easily get a

double exponential-time translation. The translation we propose uses the notions of *strategies*, *promises* and *annotations*, which we now recall.

Let $\mathcal{A} = \langle \Sigma, b, Q, \delta, q_0, \mathcal{F} \rangle$ be a $2GAPT$ with $\mathcal{F} = \langle F_1, \ldots, F_k \rangle$ and $\langle T, V \rangle$ be a Σ-labeled tree. Recall that $D_b = \langle [b] \rangle \cup [[b]] \cup \{-1, \varepsilon\}$ and $\delta : (Q \times \Sigma) \to B^+(D_b \times Q)$. For each control state $q \in Q$, let $index(q)$ be the minimal i such that $q \in F_i$. A *strategy tree* for $\mathcal{A}$ on $\langle T, V \rangle$ is a $2^{Q \times D_b \times Q}$-labeled tree $\langle T, \mathsf{str} \rangle$ such that, defined $head(w) = \{q : (q, d, q') \in w\}$ as the set of *sources* of w, it holds that (i) $q_0 \in head(\mathsf{str}(root(T)))$ and (ii) for each node $x \in T$ and state q, the set $\{(q, q') : (q, d, q') \in \mathsf{str}(x)\}$ satisfies $\delta(q, V(x))$.

A *promise tree* for $\mathcal{A}$ on $\langle T, V \rangle$ is a $2^{Q \times Q}$-labeled tree $\langle T, \mathsf{pro} \rangle$. We say that pro *fulfills* str for V if the states promised to be visited by pro satisfy the obligations induced by str as it runs on V. Formally, pro fulfills str for V if for every node $x \in T$, the following hold: "for every $(q, \langle n \rangle, q') \in \mathsf{str}(x)$ (resp. $(q, [n], q') \in \mathsf{str}(x)$), at least $n + 1$ (resp $deg(x) - n$) successors $x \cdot j$ of x have $(q, q') \in \mathsf{pro}(x \cdot j)$".

An *annotation tree* for $\mathcal{A}$ on $\langle T, \mathsf{str} \rangle$ and $\langle T, \mathsf{pro} \rangle$ is a $2^{Q \times \{1,\ldots,k\} \times Q}$-labeled tree $\langle T, \mathsf{ann} \rangle$ such that for each $x \in T$ and $(q, d_1, q_1) \in \mathsf{str}(x)$ the following hold:

- if $d_1 = \varepsilon$, then $(q, index(q_1), q_1) \in \mathsf{ann}(x)$;
- if $d_1 \in \{1, \ldots, k\}$, then for all $d_2 \in \{1, \ldots, k\}$ and $q_2 \in Q$ such that $(q_1, d_2, q_2) \in \mathsf{ann}(x)$, we have $(q, \min(d_1, d_2), q_2) \in \mathsf{ann}(x)$;
- if $d_1 = -1$ and $x = y \cdot i$, then for all $d_2, d_3 \in \{1, \ldots, k\}$ and $q_2, q_3 \in Q$ such that $(q_1, d_2, q_2) \in \mathsf{ann}(y)$, $(q_2, d_3, q_3) \in \mathsf{str}(y)$, and $(q_2, q_3) \in \mathsf{pro}(x)$, it holds that $(t, \min(index(q_1), d_2, index(q_3)), q_3) \in \mathsf{ann}(x)$;
- if $d_1 \in [[b]] \cup \langle [b] \rangle$, $y = x \cdot i$, and $(q, q_1) \in \mathsf{pro}(y)$, then for all $d_2, d_3 \in \{1, \ldots, k\}$ and $q_2, q_3 \in Q$ such that $(q_1, d_2, q_2) \in \mathsf{ann}(y)$ and $(q_2, -1, q_3) \in \mathsf{str}(y)$, it holds that $(t, \min(index(q_1), d_2, index(q_3)), q_3) \in \mathsf{ann}(x)$.

A downward path induced by str, pro, and ann on $\langle T, V \rangle$ is a sequence $\langle x_0, q_0, t_0 \rangle, \langle x_1, q_1, t_1 \rangle, \ldots$ such that $x_0 = root(T)$, q_0 is the initial state of $\mathcal{A}$ and, for each $i \geq 0$, it holds that $x_i \in T$, $q_i \in Q$, and $t_i = \langle q_i, d, q_{i+1} \rangle \in \mathsf{str}(x_i) \cup \mathsf{ann}(x_i)$ is such that either *(i)* $d \in \{1, \ldots, k\}$ and $x_{i+1} = x_i$, or *(ii)* $d \in \langle [b] \rangle \cup [[b]]$ and there exists $c \in \{1, \ldots, deg(x_i)\}$ such that $x_{i+1} = x_i \cdot c$ and $(q_i, q_{i+1}) \in \mathsf{pro}(x_{i+1})$. In the first case we set $index(t_i) = d$ and in the second case we set $index(t_i) = \min\{j \in \{1, \ldots, k\} \mid q_{i+1} \in F_j\}$. Moreover, for a downward path π, we set $index(\pi)$ as the minimum index that appears infinitely often in π. Finally, we say that π is *accepting* if $index(\pi)$ is even.

The following lemma relates languages accepted by $2GAPT$ with strategies, promises, and annotations.

Lemma 2 ([BLMV06]). *Let $\mathcal{A}$ be a* 2GAPT*. A Σ-labeled tree $\langle T, V \rangle$ is accepted by $\mathcal{A}$ iff there exist a strategy tree $\langle T, \mathsf{str} \rangle$, a promise tree $\langle T, \mathsf{pro} \rangle$ for $\mathcal{A}$ on $\langle T, V \rangle$ such that* pro *fulfills* str *for V, and an annotation tree $\langle T, \mathsf{ann} \rangle$ for $\mathcal{A}$ on $\langle T, V \rangle$, $\langle T, \mathsf{str} \rangle$ and $\langle T, \mathsf{pro} \rangle$ such that every downward path induced by* str*,* pro*, and* ann *on $\langle T, V \rangle$ is accepting.*

Given an alphabet Σ for the input tree of a $2GAPT$ with transition function δ, let D_b^δ be the subset containing only the elements of D_b appearing in δ.

Then we denote by Σ' the extended alphabet for the combined trees, i.e., $\Sigma' = \Sigma \times 2^{Q \times D_b^\delta \times Q} \times 2^{Q \times Q} \times 2^{Q \times \{1,\dots k\} \times Q}$.

Lemma 3. *Let $\mathcal{A}$ be a* 2GAPT *running on Σ–labeled trees with n states, index k and counting bound b that accepts h-ary trees. It is possible to construct in exponential-time an* NPT *$\mathcal{A}'$ running on Σ'–labeled h-ary trees that accepts a tree iff $\mathcal{A}$ accepts its projection on Σ.*

Proof. Let $\mathcal{A} = \langle \Sigma, b, Q, q_0, \delta, \mathcal{F} \rangle$ with $\mathcal{F} = \langle F_1, \dots, F_k \rangle$. By Lemma 2, we construct $\mathcal{A}'$ as the intersection of three *NPT* $\mathcal{A}'$, $\mathcal{A}''$, and $\mathcal{A}'''$, each having size exponential in the size of $\mathcal{A}$, such that, given a Σ'-labeled tree $T' = \langle T, (V, \mathsf{str}, \mathsf{pro}, \mathsf{ann}) \rangle$, *(i)* $\mathcal{A}'$ accepts T' iff str is a strategy for $\mathcal{A}$ on $\langle T, V \rangle$ and pro fulfills str for V, *(ii)* $\mathcal{A}''$ accepts T' iff ann is an annotation for $\mathcal{A}$ on $\langle T, V \rangle$, $\langle T, \mathsf{str} \rangle$ and $\langle T, \mathsf{pro} \rangle$, and *(iii)* $\mathcal{A}'''$ accepts T' iff every downward path induced by str, pro, and ann on $\langle T, V \rangle$ is accepting.

The automaton $\mathcal{A}' = \langle \Sigma', Q', q_0', \delta', \mathcal{F}' \rangle$ works as follows: on reading a node x labeled $(\sigma, \eta, \rho, \omega)$, then it locally checks whether η satisfies the definition of strategy for $\mathcal{A}$ on $\langle T, V \rangle$. In particular, when $\mathcal{A}'$ is in its initial state, we check that η contains a transition starting from the initial state of $\mathcal{A}$. Moreover, the automaton $\mathcal{A}'$ sends to each child $x \cdot i$ the pairs of states that have to be contained in $\mathsf{pro}(x \cdot i)$, in order to verify that pro fulfills str. To obtain this, we set $Q' = 2^{Q \times Q} \cup \{q_0'\}$ and $\mathcal{F}' = \{\emptyset, Q'\}$. To define δ', we first give the following definition. For each node $x \in T$ labeled $(\sigma, \eta, \rho, \omega)$, we set

$$
\begin{aligned}
S(\eta) = \{ & \langle S_1, \dots, S_{deg(x)} \rangle \in (2^{Q \times Q})^{deg(x)} \text{ such that} \\
& [\text{for each } (q, \langle m \rangle, p) \in \eta \text{ there is } P \subseteq \{1, \dots deg(x)\} \text{ with } |P| = m + 1 \\
& \text{such that for all } i \in P,\ (q, p) \in S_i] \text{ and} \\
& [\text{for each } (q, [m], p) \in \eta \text{ there is} P \subseteq \{1, \dots deg(x)\} \text{ with } |P| = deg(x) - m \\
& \text{such that for all } i \in P,\ (q, p) \in S_i] \}
\end{aligned}
$$

to be the set of all tuples with size $deg(x)$, each *fulfilling* all graded modalities in $\mathsf{str}(x)$. Notice that $|S(\eta)| \leq 2^{hn^2}$. Then we have

$$
\delta'(q, (\sigma, \eta, \rho, \omega)) = \begin{cases} S(\eta) & \text{if } \forall\ p \in head(\eta),\ \{(d, p') \mid (p, d, p') \in \eta\} \text{ satisfies } \delta(p, \sigma) \\ & \text{and } [(q = q_0^1 \text{ and } q_0 \in head(\eta)) \text{ or } (q \neq q_0^1 \text{ and } q \subseteq \rho)] \\ \mathbf{false} & \text{otherwise.} \end{cases}
$$

Hence, in $\mathcal{A}'$ we have $|Q'| = 2^{n^2}$, $|\delta'| \leq 2^{n^2(k+1)}$, and index 2.

$\mathcal{A}'' = \langle \Sigma', Q'', q_0'', \delta'', \mathcal{F}'' \rangle$ works in a similar way to $\mathcal{A}'$. That is, for each node x, it first locally checks whether the constraints of the annotations are verified; then it sends to the children of x the strategy and annotation associated with x, in order to successively verify whether the promises associated with the children nodes are consistent with the annotation of x. Therefore, in A'' we have $Q'' = 2^{Q \times D_b^\delta \times Q} \times 2^{Q \times \{1,\dots,k\} \times Q}$, $q_0'' = (\emptyset, \emptyset)$, $\mathcal{F}'' = \{\emptyset, Q''\}$, and for a state $(\eta_{prev}, \omega_{prev})$ and a letter $(\sigma, \eta, \rho, \omega)$ we have

$$
\delta''((\eta_{prev}, \omega_{prev}), (\sigma, \eta, \rho, \omega)) = \begin{cases} \langle (\eta, \omega), \dots, (\eta, \omega) \rangle & \text{if the local conditions for the} \\ & \text{annotations are verified} \\ \mathbf{false} & \text{otherwise.} \end{cases}
$$

Hence, in $\mathcal{A}''$ we have $|Q''| \leq 2^{n^2(|\delta|+k)}$, $|\delta''| \leq h \cdot 2^{n^2(|\delta|+k)}$, and index 2.

Finally, to define $\mathcal{A}'''$ we start by constructing a $2APT$ $\mathcal{B}$ whose size is polynomial in the size of $\mathcal{A}$ and accepts $\langle T, (V, \mathsf{str}, \mathsf{pro}, \mathsf{ann})\rangle$ iff there is a non accepting downward path (w.r.t. $\mathcal{A}$) induced by str, pro, and ann on $\langle T, V\rangle$. The automaton $\mathcal{B} = \langle \Sigma', Q^B, q_0^B, \delta^B, \mathcal{F}^B\rangle$ (which in particular does not need direction -1) essentially chooses, in each state, the downward path to walk on, and uses an integer to store the index of the state. We use a special state $\sharp$ not belonging to Q to indicate that $\mathcal{B}$ proceeds in accordance with an annotation instead of a strategy. Therefore, $Q^B = ((Q \cup \{\sharp\}) \times \{1, \ldots, k\} \times Q) \cup \{q_0^B\}$.

To define the transition function on a node x, let us introduce a function f that for each $q \in Q$, strategy $\eta \in 2^{Q \times D_b^\delta \times Q}$, and annotation $\omega \in 2^{Q \times \{1,\ldots,k\} \times Q}$ gives a formula satisfied along downward paths consistent with η and ω, starting from a node reachable in $\mathcal{A}$ with the state q. That is, in each node x, the function f either proceeds according to the annotation ω or the strategy η (note that f does not check that the downward path is consistent with any promise). Formally, f is defined as follows, where $index(p)$ is the minimum i such that $p \in F_i$:

$$f(q,\eta,\omega) = \bigvee_{\substack{(q,d,p)\in\omega \\ d\in\{1,\ldots,k\}}} \langle \varepsilon, (\sharp, d, p)\rangle \quad \vee \quad \bigvee_{\substack{(q,d,p)\in\eta \\ d\in\langle[b]\rangle\cup[[b]]}} \bigvee_{c\in\{1,\ldots,deg(x)\}} \langle c, (q, index(p), p)\rangle$$

Then, we have $\delta^B(q_0^B, (\sigma, \eta, \rho, \omega)) = f(q_0, \eta, \omega)$ and

$$\delta^B((q,d,p),(\sigma,\eta,\rho,\omega)) = \begin{cases} \mathbf{false} & \text{if } q \neq \sharp \text{ and } (q,p) \notin \rho \\ f(p,\eta,\omega) & \text{otherwise.} \end{cases}.$$

A downward path π is non accepting for $\mathcal{A}$ if the minimum index that appears infinitely often in π is odd. Therefore, $\mathcal{F}^B = \langle F_1^B, \ldots, F_{k+1}^B, Q^B\rangle$ where $F_1^B = \emptyset$ and, for all $i \in \{2, \ldots, k+1\}$, we have $F_i^B = \{(q,d,p) \in Q^B \mid d = i-1\}$. Thus, $|Q^B| = kn(n+1)+1$, $|\delta^B| = k \cdot |\delta| \cdot |Q^B|$, and the index is $k+2$. Then, since $\mathcal{B}$ is alternating, we can easily complement it in polynomial-time into a $2APT$ $\overline{\mathcal{B}}$ that accepts a tree iff all downward paths induced by str, pro, and ann on $\langle T, V\rangle$ are accepting. Finally, following [Var98] we construct in exponential-time the desired automaton $\mathcal{A}'''$. □

5 Deciding Hybrid Graded Pushdown Module Checking

In this section, we show that hybrid graded pushdown module checking is decidable and solvable in 2ExpTime. Since *CTL* pushdown module checking is 2ExpTime-hard, we get that the addressed problem is 2ExpTime-complete. For the upper bound, the algorithm works as follows. Given an *OPD* $\mathcal{S}$ and the module $\mathcal{M}_\mathcal{S}$ induced by $\mathcal{S}$, by combining and extending the constructions given in [BMP05] and [FM07], we first build in polynomial-time a *PD–NBT* $\mathcal{A}_\mathcal{S}$ accepting each tree that encodes a quasi-forest belonging to $exec(\mathcal{M}_\mathcal{S})$. Then, given an hybrid graded μ-calculus formula φ, according to [BLMV06], we build in polynomial-time a *GAPT* $\mathcal{A}_{\neg\varphi}$ (Lemma 1) accepting all models of $\neg\varphi$, with the

intent of checking that no models of $\neg\varphi$ are in $exec(\mathcal{M}_\mathcal{S})$ [3]. Then, accordingly to the basic idea of [KVW01], we check that $\mathcal{M} \models_r \varphi$ by checking whether $\mathcal{L}(\mathcal{A}_\mathcal{S}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. Finally, we get the result by using an exponential-time reduction of the latter to the emptiness problem for *PD–NPT*, which from Proposition 1 can be solved in EXPTIME. As a key step of the above reduction, we use the exponential-time translation from *GAPT* into *NPT* showed in Section 4.

Let us start dealing with $\mathcal{A}_\mathcal{S}$. Before building the automaton, there are some technical difficulties to overcome, which are also new with respect to [BMP05]. First note that since $\mathcal{A}_\mathcal{S}$ is a *PD–NBT*, it only deals with trees having labels on nodes, while $exec(\mathcal{M})$ contains quasi-forests with both edges and nodes labeled. To solve this problem, for each quasi-forest in $exec(\mathcal{M})$, the automaton $\mathcal{A}_\mathcal{S}$ accepts a corresponding encoding tree obtained by *(i)* adding a new root connecting all roots of the quasi-forest, *(ii)* moving the label of each edge to the target node of the edge (using a new atomic proposition p_α, for each program α), and *(iii)* substituting "jumps to roots" with new atomic propositions $\uparrow_o^\alpha$ (representing an α-labeled jump to the unique root node labeled by nominal o). Let $AP^* = AP \cup \{p_\alpha \mid \alpha \text{ is a program}\} \cup \{\uparrow_o^\alpha \mid \alpha \text{ is a program and } o \text{ is a nominal}\}$, we denote with $\langle T, V^*\rangle$ the $2^{AP^* \cup Nom}$-labeled tree *encoding* of a quasi-forest $\langle F, V, E\rangle \in exec(\mathcal{M})$, obtained using the above transformations.

Another technical difficulty to handle with is related to the fact that quasi-forests of $exec(\mathcal{M})$ (and thus their encodings) may not be full h-ary, since the nodes of the *OPD* from which $\mathcal{M}$ is induced may have different degrees. Also, quasi-forests of $exec(\mathcal{M})$ may not share the same structure, since they are obtained by pruning subtrees from the computation quasi-forest $\langle F_\mathcal{M}, V_\mathcal{M}, E_\mathcal{M}\rangle$ of $\mathcal{M}$. Let $\langle T_\mathcal{M}, V^*_\mathcal{M}\rangle$ be the h-ary *computation tree* of $\mathcal{M}$ obtained from $\langle F_\mathcal{M}, V_\mathcal{M}, E_\mathcal{M}\rangle$ using the above encoding. By extending an idea of [KVW01, BMP05], we consider each tree $\langle T, V^*\rangle$, encoding of a quasi-forest $\langle F, V, E\rangle$ of $exec(\mathcal{M})$, as a $2^{AP^* \cup Nom \cup \{t\}} \cup \{\bot\}$-labeled full h-ary tree $\langle T_\mathcal{M}, V^{**}\rangle$ (where $\bot$ and t are fresh proposition names not belonging to $AP^* \cup Nom$) in the following way: first we add proposition t to the label of all leaf nodes of the forest; second, for each node $x \in T_\mathcal{M}$ with p children $x \cdot 1, \ldots, x \cdot p$ (note that $0 \leq p \leq h$), we add the children $x \cdot (p+1), \ldots, x \cdot h$ and label these new nodes with $\bot$; finally, for each node x labeled by $\bot$ we add recursively h children labeled by $\bot$. Thus, for each node $x \in T_\mathcal{M} \backslash \{root(T_\mathcal{M})\}$, if $x \in F$ then $V^{**}(x) = V^*(x)$, otherwise $V^{**}(x) = \{\bot\}$ and therefore the proposition $\bot$ is used to denote both "disabled" states and "completion" states. In this way, all trees encoding quasi-forests belonging to $exec(\mathcal{M})$ are full h-ary trees, and they differ only in their labeling. Moreover, the environment can also disable jumps to roots. This is performed by removing from enabled environment nodes some of the $\uparrow_o^\alpha$ labels. Notice that since we consider environments that do not block the system, nodes associated with environment states have at least one successor not labeled by $\{\bot\}$, unless they have $\uparrow_o^\alpha$ in their labels. Putting in practice the construction proposed above, we obtain the following result, where $\widehat{exec}(\mathcal{M})$ is the set of all $2^{AP^* \cup Nom \cup \{t\}} \cup \{\bot\}$-labeled

[3] For better readability, in the rest of the paper we use $\mathcal{M}$ instead of $\mathcal{M}_\mathcal{S}$.

full h-ary trees obtained from $\langle T_{\mathcal{M}}, V^*_{\mathcal{M}}\rangle$ in the above described manner (the detailed construction is reported in the full version of the paper).

Lemma 4. *Given an* OPD $\mathcal{S} = \langle Q, \Gamma, \flat, C_0, \Delta, \rho_1, \rho_2, Env\rangle$ *with branching degree* h*, we can build a* PD–NBT $\mathcal{A}_{\mathcal{S}} = \langle \Sigma, \Gamma, \flat, Q', q'_0, \gamma_0, \delta, Q\rangle$*, which accepts exactly* $\widehat{exec}(\mathcal{M})$*, such that* $\Sigma = 2^{AP^* \cup Nom \cup \{t\}} \cup \{\bot\}$*,* $|Q'| = \mathcal{O}(|Q|^2 \cdot |\Gamma|)$*, and* $|\delta|$ *is polynomially bounded by* $h \cdot |\Delta|$*.*

Let us now go back to the hybrid graded μ-calculus formula φ. Using Lemmas 1 and 3, we get that given an hybrid graded μ-calculus formula φ, we can build in exponential-time an *NPT* $\mathcal{A}_{\neg\varphi}$ accepting all models of $\neg\varphi$. Now, recall that given a module $\mathcal{M}$ induced by an *OPD* S and the automaton $\mathcal{A}_{\mathcal{S}}$ accepting all trees encoding of quasi-forests belonging to $exec(\mathcal{M})$ (see Lemma 4), it is possible to check whether $\mathcal{M} \models_r \varphi$ by checking whether $\mathcal{L}(\mathcal{A}_{\mathcal{S}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. Also, recall by Proposition 2 that $\mathcal{L}(\mathcal{A}_{\mathcal{S}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ can be accepted by a *PD–NPT* $\mathcal{A}$ whose size is exponential in the size of φ and polynomial in the size of $\mathcal{S}$. Finally, by recalling that the emptiness problem for $\mathcal{A}$ can be checked in Exptime (Proposition 1) and that the pushdown module checking problem for *CTL* is 2Exptime-hard with respect to the size of the formula and Exptime-hard in the size of the system [BMP05], we get the following result.

Theorem 1. *The hybrid graded* μ*-calculus pushdown module checking problem is* 2Exptime*–complete with respect to the size of the formula and* Exptime*–complete with respect to the size of the system.*

6 Discussion

As a direct consequence of the algorithm we have proposed, we get that "pushdown model checking" with respect to formulas of the hybrid graded μ-calculus is also solvable in 2Exptime. Indeed, we recall that model checking a closed system is equivalent to check a module in which the maximal environment, which corresponds to the full computation tree, satisfies the specification. Consider now the automaton described in Lemma 4. We can easily simplify the construction to get an automaton that accepts only the full computation tree. Then, by applying our module checking algorithm we easily get the result.

The above idea can be also extended to other fragments of the fully enriched μ-calculus. We recall that this calculus extends the hybrid graded one by also allowing "backward" modalities. Syntactically, it is obtained by allowing a program α in *atleast* and *allbut* formulas to be either an atomic program a or its inverse a^-. To deal with inverse programs, we also extend R as follows: for each $a \in Prog$, we set $R(a^-) = \{(v, u) : (u, v) \in R(a)\}$. The semantics given for hybrid graded μ-calculus extends to fully enriched μ-calculus, accordingly and in a natural way. We recall that fully enriched μ-calculus is undecidable (see [BP04]), while any of its fragments is decidable. We argue that also for those fragments including backwards modalities, the pushdown model checking is solvable in 2Exptime. The main technical difficulty here is that since we are

dealing with a backward modality, the unwinding must take care also of the past configurations. In particular, since each node in the tree can only have one parent, we need to simulate in forward all past configurations, but one. This can be accomplished by inverting the modality (i.e., inverting the program). Using such an unwinding, along with the above idea of constructing an automaton (by simplifying that given in Lemma 4) that accepts only the full computation tree (which now also needs to keep track of the past computations), and the pushdown module checking algorithm idea, we get the result.

Acknowledgments. We thank the anonymous referee for his useful comments regarding backward modalities.

References

[BLMV06] Bonatti, P.A., Lutz, C., Murano, A., Vardi, M.Y.: The complexity of enriched μ-calculi. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 540–551. Springer, Heidelberg (2006)

[BMP05] Bozzelli, L., Murano, A., Peron, A.: Pushdown module checking. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 504–518. Springer, Heidelberg (2005)

[BP04] Bonatti, P.A., Peron, A.: On the undecidability of logics with converse, nominals, recursion and counting. Artif. Intelligence 158(1), 75–96 (2004)

[CE81] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

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

[FM07] Ferrante, A., Murano, A.: Enriched μ-calculus module checking. In: FOSSACS 2007. LNCS, vol. 4423, pp. 183–197 (2007)

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

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

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

[KPV02] Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown specifications. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, pp. 262–277. Springer, Heidelberg (2002)

[KSV02] Kupferman, O., Sattler, U., Vardi, M.Y.: The complexity of the graded μ-calculus. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, pp. 423–437. Springer, Heidelberg (2002)

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

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

[QS81] Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in cesar. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)

[SV01] Sattler, U., Vardi, M.Y.: The hybrid mu-calculus. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 76–91. Springer, Heidelberg (2001)

[Var98] Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)