A Counting Logic for Structure Transition Systems

Łukasz Kaiser[1] and Simon Leßenich*[2]

1 **LIAFA, CNRS & Université Paris Diderot – Paris 7, France**
2 **Mathematische Grundlagen der Informatik, RWTH Aachen**

Abstract

Quantitative questions such as "what is the maximum number of tokens in a place of a Petri net?" or "what is the maximal reachable height of the stack of a pushdown automaton?" play a significant role in understanding models of computation. To study such problems in a systematic way, we introduce structure transition systems on which one can define logics that mix temporal expressions (e.g. reachability) with properties of a state (e.g. the height of the stack). We propose a counting logic $\mathsf{Q}\mu[\#\mathsf{MSO}]$ which allows to express questions like the ones above, and also many boundedness problems studied so far. We show that $\mathsf{Q}\mu[\#\mathsf{MSO}]$ has good algorithmic properties, in particular we generalize two standard methods in model checking, decomposition on trees and model checking through parity games, to this quantitative logic. These properties are used to prove decidability of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ on tree-producing pushdown systems, a generalization of both pushdown systems and regular tree grammars.

1998 ACM Subject Classification F.4.1, I.2.4

Keywords and phrases Logic in Computer Science, Quantitative Logics, Model Checking

1 Introduction

Models of computation describe temporal changes of a state of a system or a machine. For example, pushdown automata describe transformations of the stack, term rewriting systems capture modifications of terms, and Turing machines specify changes of the tape. Questions about computations of systems often ask about temporal events intertwined with properties of the state, e.g. reachability (temporal) of an empty stack (state property) or never encountering a tree of height less than one. We propose an abstract definition which distinguishes temporal transitions from the state of the system and allows to investigate properties of such systems and the corresponding logics in a uniform and systematic way.

▸ **Definition 1.** A *structure transition system* (STS) is a labeled transition system (Kripke structure) with an additional assignment $\mathfrak{m}$ of finite relational structures to the nodes of the system. Formally, given a set of transition labels R and a relational signature τ, an STS is a tuple $\mathfrak{S} = (S, \Delta, \mathfrak{m})$ where S is a set of states, $\Delta \subseteq S \times R \times S$ is a set of transitions, and $\mathfrak{m} : S \to \mathsf{FinStr}(\tau)$ assigns a finite τ-structure to each state.

Computing machines of various kinds can be viewed as finite objects which represent and generate infinite structure transition systems. Let us consider a few prominent examples.

- *Pushdown automata* induce structure transition systems in which the relational structures assigned to the nodes represent the current stack of the automaton, i.e. are words over the stack alphabet. The transition relation in the STS copies the one in the automaton.

* This author was supported by the ESF Research Networking Programme GAMES and the DFG Research Training Group 1298 (AlgoSyn).

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- *Turing machines* generate STSs in a similar way to pushdown automata: the structure assigned to each node of the STS is again a word, and it represents the tape of the machine at that time. Transitions are induced from the ones of the machine.
- *Petri nets* give rise to STSs in which elements of the relational structures correspond to tokens in the net. Elements labeled by a predicate P_l correspond to tokens in place l in the net and firing of the net results in transitions of the STS.
- *Term rewriting systems* (TRSs) produce STSs in which one assigns a term to each node, i.e. the relational structure $\mathfrak{m}(s)$ is always a labeled tree, representing the term in s. The transition relation Δ is derived from the application of rewriting rules.
- *Graph rewriting systems* induce STSs in a similar way as TRSs, but the structures assigned to nodes are arbitrary graphs, or even hypergraphs in case of hypergraph rewriting.

Interesting properties of structure transition systems mix temporal events with state attributes. To specify them, we thus compose a temporal logic with another logic for the states. In such composition, the predicates of the temporal logic are replaced by *sentences* of the state logic, which in turn are evaluated on the relational structure assigned by $\mathfrak{m}$.

Consider the logic LTL[MSO], which allows to use MSO sentences in place of predicates in LTL. For example, the formula $G(\exists x\, a(x))$ expresses that an a-labeled element exists in the relational structure assigned to each reachable state of an STS. Since MSO sentences define regular languages of words and trees, one can express in LTL[MSO] over an STS the reachability of a state in which the stack belongs to a regular language (for pushdown systems) or in which a regular configuration appears on the tape (for Turing machines) or in which the term belongs to a regular tree language (for TRSs). Note that a LTL[MSO] formula defines a property in a uniform way, for pushdown systems, Turing machines and TRSs at the same time. Moreover, we can systematically inspect which temporal logic and which state logic can be combined to an efficient formalism on which classes of STS.

- If the state logic is trivial, i.e. consists only of the two formulas true and false, one obtains classical action-based temporal logics like LTL or the μ-calculus Lμ. Model-checking these logics is decidable on pushdown systems [13] and for linear-time logics (e.g. LTL) it is also decidable on Petri nets [5]. On the other hand, already model-checking action-based branching-time logics, e.g. Lμ, is undecidable on Petri nets [5].
- One can consider the logic Reach[MSO] in which formulas have the form Reach(ψ) for some $\psi \in$ MSO and express that an MSO-definable configuration is reachable. Model-checking this logic is decidable on pushdown systems [13] and also on Petri nets [14].
- Model-checking LTL[MSO] is decidable on STSs generated by pushdown systems [6], but the use of MSO makes it undecidable on STSs corresponding to Petri nets [5].
- For Lμ[MSO], the modal μ-calculus with MSO sentences as predicates, the model checking problem is decidable on context-free and prefix-recognizable rewrite systems, and thus also on pushdown systems [13]. It is also decidable on some classes of graph or structure rewriting systems, e.g. for separated structure rewriting [11].

Combinations of temporal and state logics, as the ones above, allow to express interesting properties of structure transition systems, but, since the formulas of these logics are Boolean, they are limited to yes-or-no answers. For example, it is not possible to ask "how high will the stack get on all runs?" of a pushdown automaton or "how many tokens will there maximally be in a place?" of a Petri net. Such questions are often very important for understanding the behavior of the system. Note also that the answer to a question of this kind might be either an integer or $\pm\infty$, in case the stack size or the number of tokens is unbounded. Such boundedness problems have been studied extensively for many models.

- The question whether the maximal stack size on runs of a pushdown system is bounded or not, intertwined with temporal properties, has been studied in [2, 9], and is a special case of the problem solved in this work.
- The boundedness problem for Petri nets was shown to be decidable in [12] and became one of the most important tools in Petri net analysis.
- On Turing machines, establishing the bound on the size of the tape during a computation is the same as determining its space complexity.
- Graph rewriting systems are used to model e.g. biochemical processes and one often asks for the number of particles of certain kind produced in the process.

In the next section, we introduce a counting logic Qμ[#MSO] which allows to express queries like the ones discussed above. Fundamental algorithmic techniques from model checking generalize to this logic, as we show in Section 3. We apply these methods in Section 4 to compute the value of Qμ[#MSO] formulas on tree-producing pushdown systems. In the proof, we use two key lemmas, proved in Section 5 and Section 6.

2 Counting μ-Calculus on Structure Transition Systems

To express questions of the above form, we propose the *counting μ-calculus*, a *quantitative* logic in which each formula has not just a Boolean value, but it evaluates to a number in $\mathbb{Z}_\infty := \mathbb{Z} \cup \{-\infty, \infty\}$. This logic, denoted Q$\mu$[#MSO], allows to use counting terms on state structures and maximum, minimum and fixed-point operations in the temporal part. This suffices to express all the example questions presented above and to query for boundedness. Note that it is not a probabilistic logic, and we do not introduce operators for sums over different paths. Qμ[#MSO] is composed of the quantitative μ-calculus [7], and for quantitative predicates uses counting terms on top of MSO formulas, defined as follows.

▸ **Definition 2.** An *MSO counting term* has the form $\#_{x_1 \cdots x_n} \varphi(x_1, \cdots, x_n)$, where $\{x_1, \cdots, x_n\}$ is the set of *all* free variables of the MSO formula φ. On a finite structure $\mathfrak{A}$, the term represents the number of tuples $a_1, \cdots, a_n$ such that $\mathfrak{A} \vDash \varphi(a_1, \cdots, a_n)$,

$$[\![\#_{\overline{x}} \varphi(\overline{x})]\!]^{\mathfrak{A}} := |\{\overline{a} \mid \mathfrak{A} \vDash \varphi(\overline{a})\}|.$$

For a formula φ without free variables, we set $\#\varphi = 1$ if φ holds and $\#\varphi = 0$ in the other case.

Using the above counting terms as predicate symbols, formulas of Qμ[#MSO] are built according to the following grammar, analogous to [7].

$$\psi \ ::= \ \#_{\overline{x}} \varphi(\overline{x}) \mid X \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \ \Box_r \psi \mid \ \Diamond_r \psi \mid \mu X.\psi \mid \nu X.\psi,$$

where $r \in R$ are labels of the transitions and each $X \in \mathrm{Var}$ is a fixed point variable and must appear positively in φ, i.e. under an even number of negations. We will often write $\Diamond$ for the disjunction of $\Diamond_r$ over all $r \in R$ for a finite R, and $\Box$ analogously. The semantics of Qμ[#MSO] combines the quantitative μ-calculus [7] with MSO counting terms.

▸ **Definition 3.** Let $\mathfrak{S} = (S, \Delta, \mathfrak{m})$ be a structure transition system and $\mathcal{F} := \{f : S \to \mathbb{Z}_\infty\}$ the set of quantitative assignments to the states of $\mathfrak{S}$. Given an evaluation of fixed point variables $\varepsilon : \mathrm{Var} \to \mathcal{F}$ we define the evaluation of a Qμ[#MSO] formula ψ, denoted $[\![\psi]\!]^{\mathfrak{S}}_{\varepsilon} : S \to \mathbb{Z}_\infty$, in the following inductive way.

- $[\![X]\!]^{\mathfrak{S}}_{\varepsilon} = \varepsilon(X)$
- $[\![\#_{\overline{x}} \varphi(\overline{x})]\!]^{\mathfrak{S}}_{\varepsilon}(s) = [\![\#_{\overline{x}} \varphi(\overline{x})]\!]^{\mathfrak{m}(s)} = |\{\overline{a} \mid \mathfrak{m}(s) \vDash \varphi(\overline{a})\}|$

- $[\![\neg\psi]\!]^{\mathfrak{S}}_{\varepsilon} = -1 \cdot [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon}$
- $[\![\psi_1 \wedge \psi_2]\!]^{\mathfrak{S}}_{\varepsilon} = \min([\![\psi_1]\!]^{\mathfrak{S}}_{\varepsilon}, [\![\psi_2]\!]^{\mathfrak{S}}_{\varepsilon})$, $[\![\psi_1 \vee \psi_2]\!]^{\mathfrak{S}}_{\varepsilon} = \max([\![\psi_1]\!]^{\mathfrak{S}}_{\varepsilon}, [\![\psi_2]\!]^{\mathfrak{S}}_{\varepsilon})$
- $[\![\Diamond_r \psi]\!]^{\mathfrak{S}}_{\varepsilon}(s) = \sup_{\{s' \mid (s,r,s') \in \Delta\}} [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon}(s')$, $[\![\Box_r \psi]\!]^{\mathfrak{S}}_{\varepsilon}(s) = \inf_{\{s' \mid (s,r,s') \in \Delta\}} [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon}(s')$
- $[\![\mu X.\psi]\!]^{\mathfrak{S}}_{\varepsilon}$ is the least and $[\![\nu X.\psi]\!]^{\mathfrak{S}}_{\varepsilon}$ the greatest fixed point of the operator $f \mapsto [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon[X \leftarrow f]}$

To compute the fixed point, we consider $\mathcal{F}$ as a complete lattice with pointwise order, i.e. $f \leq g$ if and only if $f(s) \leq g(s)$ for all states $s \in S$.

Note that the above definition is very similar to the classical, Boolean μ-calculus. As in the standard case, one can evaluate fixed points inductively, e.g. the least fixed point starting from a function which assigns $-\infty$ to all states. The Boolean logic $\mathsf{L}\mu[\mathsf{MSO}]$ can in fact be embedded in $\mathsf{Q}\mu[\#\mathsf{MSO}]$ as follows: take a formula ψ of $\mathsf{L}\mu[\mathsf{MSO}]$ in negation normal form and replace each literal $\neg\varphi$ by $\#\neg\varphi$ and each φ by $\#\varphi$. These terms have now value 1 if φ holds and 0 in the other case, and since the semantics coincide, the $\mathsf{Q}\mu[\#\mathsf{MSO}]$ formula obtained in this way will evaluate to ∞ or 1 if ψ holds and to 0 or $-\infty$ otherwise. In this sense the logic $\mathsf{Q}\mu[\#\mathsf{MSO}]$ subsumes $\mathsf{L}\mu[\mathsf{MSO}]$, and therefore also several other Boolean logics, e.g. $\mathsf{LTL}[\mathsf{MSO}]$ and $\mathsf{CTL}[\mathsf{MSO}]$. But, of course, in addition to Boolean ones, $\mathsf{Q}\mu[\#\mathsf{MSO}]$ allows to express quantitative properties, e.g. the following.

- The formula $\psi_{\#} = \mu X.(\#_x(x = x) \vee \Diamond X)$ calculates the bound on the size of structures appearing on all runs of the STS from where it is evaluated. Therefore $\varphi_{\#}(s) \neq \infty$ if and only if there is a bound on the size of the structures on all runs from s, and checking if $\varphi_{\#}(s) \neq \infty$ answers the boundedness problems mentioned before.
- The formula $\psi_{\times} = \nu X.(\#_{x,y}(a(x) \wedge b(y)) \wedge \Box X)$ computes the minimal *product* of the number of a-labeled elements and b-labeled ones on all paths from the node in which it is evaluated.
- For two atoms φ_1 and φ_2, we will denote the formula $\mu X.(\varphi_2 \vee (\varphi_1 \wedge \Diamond X))$ by $\varphi_1 \,\mathsf{until}\, \varphi_2$, as it is the standard $\mathsf{L}\mu$ formula expressing the LTL until modality. In the quantitative setting, this formula calculates the maximal value of φ_1 at the last node in which $\varphi_1 > \varphi_2$ on all paths. If we set $\varphi_1 = \#_x a(x)$ and $\varphi_2 = \#_x b(x)$ then $\varphi_1 \,\mathsf{until}\, \varphi_2$, on words, computes the maximal number of as reached on prefixes of runs on which there are more as than bs.

3 Model Checking Games and Decomposition

The logic $\mathsf{Q}\mu[\#\mathsf{MSO}]$ is a composition of a quantitative extension of the μ-calculus $\mathsf{L}\mu$ and a counting extension of MSO. Good algorithmic properties of $\mathsf{L}\mu$ stem from its connection to parity games, and decidability of MSO on linear orders and trees has its roots in the decomposition property. It is therefore natural to ask whether these basic methods generalize to $\mathsf{Q}\mu[\#\mathsf{MSO}]$. We give a positive answer, showing both model-checking games for $\mathsf{Q}\mu$ on structure transition systems and a decomposition theorem for $\#\mathsf{MSO}$. These two tools will be crucial in the decidability proof in the next section.

3.1 Model-Checking Games

To model-check $\mathsf{Q}\mu$ one can use quantitative parity games, as shown in [7]. We use an almost identical notion (except for discounts) for $\mathsf{Q}\mu[\#\mathsf{MSO}]$ on structure transition systems.

▸ **Definition 4.** A *quantitative parity game* (QPG) $\mathcal{G}$ is a tuple $\mathcal{G} = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ such that (V, E) is a directed graph whose vertices V are partitioned into positions $V_{\max}$ of Maximizer and positions $V_{\min}$ of Minimizer. Every vertex is assigned a color by $\Omega\colon V \to \{0, \cdots, d\}$ and terminal vertices $T = \{v \in V \mid vE = \varnothing\}$ are labeled by the payoff function $\lambda\colon T \to \mathbb{Z}$.

How to play. Every play starts at some vertex $v \in V$. For every vertex in $V_{\max}$, Maximizer chooses a successor vertex and the play proceeds from that vertex (analogously for Minimizer). If the play reaches a terminal vertex, it ends. We denote by $\pi = v_0 v_1 \dots$ the (possibly infinite) play through vertices $v_0 v_1 \dots$, given that $(v_n, v_{n+1}) \in E$ for every n. The outcome $p(\pi)$ of a finite play $\pi = v_0 \dots v_k$ is given by $\lambda(v_k)$. The outcome of an infinite play depends only on the lowest priority seen infinitely often. We will assign the value $-\infty$ to every infinite play where the lowest priority seen infinitely often is odd, and ∞ to those where it is even.

Goals. The two players have opposing objectives regarding the outcome of the play. Maximizer wants to maximize the outcome, while Minimizer wants to minimize it.

Strategies. A strategy of Maximizer (Minimizer) is a function $s : V^* V_{\max} \to V$ ($s : V^* V_{\min} \to V$) with $(v, s(hv)) \in E$ for each h, v. A play $\pi = v_0 v_1 \dots$ is *consistent with a strategy* s of Maximizer if $v_{n+1} = s(v_0 \dots v_n)$ for every n such that $v_n \in V_{\max}$, and dually for strategies of Minimizer. For strategies f, g of the two players, we denote by $\alpha_{f,g}(v)$ the unique play starting at v which is consistent with both f and g.

Determinacy. A game is *determined* if, for each position v, the highest outcome Maximizer can assure from this position and the lowest outcome Minimizer can assure coincide,

$$\sup_{f \in \Sigma_{\max}} \inf_{g \in \Sigma_{\min}} p(\alpha_{f,g}(v)) \quad = \quad \inf_{g \in \Sigma_{\min}} \sup_{f \in \Sigma_{\max}} p(\alpha_{f,g}(v)) =: \mathsf{val}\,\mathcal{G}(v),$$

where $\Sigma_{\min}, \Sigma_{\max}$ are the sets of all possible strategies for Minimizer and Maximizer and the achieved outcome is called the *value of* $\mathcal{G}$ *at* v.

As shown in [7], quantitative parity games are *determined* and can be used for model-checking. We adapt the construction from [7] and construct a model-checking game satisfying the following properties.

▸ **Theorem 5** (c.f. [7]). *For every $\psi \in$ Qμ[#MSO] and every STS $\mathfrak{S}$ one can construct the quantitative parity game $\mathsf{MC}(\mathfrak{S}, \psi) = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ which is a* model-checking game *for ψ and $\mathfrak{S}$, i.e. it satisfies the following properties:*

(1) $V = (S \times \mathsf{Sub}(\psi)) \cup \{(\infty), (-\infty)\}$, where $\mathsf{Sub}(\psi)$ is the set of subformulas of ψ.
(2) In terminal positions (s, ψ) the formula ψ has the form $\#_{\overline{x}}\varphi(\overline{x})$ or $\neg\#_{\overline{x}}\varphi(\overline{x})$.
(3) If $((s, \psi), (s', \psi')) \in E$ then either $s = s'$ or $(s, s') \in \Delta$.
(4) $\Omega(s, \psi)$ depends only on ψ, not on s.
(5) $\mathsf{val}\,\mathsf{MC}(\mathfrak{S}, \psi)(s, \psi) = [\![\psi]\!]^{\mathfrak{S}}(s)$.

The construction of the model-checking game corresponds to the one in [7], only that in the setting of STS, discounts are not needed and have thus been removed.

In this work, we will be especially interested in *pushdown quantitative parity games*. For a finite stack alphabet Γ, bottom symbol $\perp \notin \Gamma$ and a finite state set Q, we define a pushdown process $\mathcal{A} = (Q, \hookrightarrow)$ in the standard way (see Definition 1 in [16]) and we denote by $E_{\hookrightarrow}$ the corresponding one-step transition relation. We say that a QPG is a pushdown game over $\mathcal{A}$ if it has positions from $\mathcal{A}$, i.e. of the form (q, s) for $s \in \Gamma^*$ and $q \in Q$, moves given by $E_{\hookrightarrow}$, and the partition into $V_{\max}$ and $V_{\min}$ and the color Ω of a position (q, s) depend only on q and not on s. We say that a pushdown process or QPG is *pop-free* if $\hookrightarrow$ contains no pop-rules, equivalently if for each edge in $E = E_{\hookrightarrow}$ from (q, s) to (q', s') it holds that $|s'| \geq |s|$. In Section 5 we adapt the ideas from [19] to prove the following reduction.

▸ **Theorem 6.** *For every pushdown QPG $\mathcal{G}$ over $\mathcal{A} = (Q, \hookrightarrow)$ one can compute a finite set Q', $q_0' \in Q'$, a pop-free pushdown process $\mathcal{A}' = (Q \times Q', \hookrightarrow')$ and a QPG $\mathcal{G}'$ over $\mathcal{A}'$ such that $\mathsf{val}\,\mathcal{G}(q, \varepsilon) = \mathsf{val}\,\mathcal{G}'((q, q_0'), \varepsilon)$ for each $q \in Q$.*

3.2 #MSO Decomposition on Trees

The technique above allows us to reduce model-checking of the temporal part of a $\mathsf{Q}\mu[\#\mathsf{MSO}]$ formula to solving a QPG. But to provide algorithms for $\mathsf{Q}\mu[\#\mathsf{MSO}]$ we also need a method to handle the counting terms, at least on structures such as words and trees. For MSO, e.g. on trees, this can be done by *decomposing* a formula: instead of checking φ on the whole tree, one can compute tuples of formulas to check on the subtrees. Here we show that this method can be extended to MSO counting terms.

Trees. We consider at most k-branching finite trees with nodes labeled by symbols from a finite alphabet Γ. We accordingly represent them by relational structures over the signature $\tau = \{S_1, \cdots, S_k\} \cup \{P : P \in \Gamma\}$, where each S_i is a binary relation representing the i-th successor. We use P_i and Q_j for the symbols from Γ and write $P - t_1 \cdots t_l$ for the tree with P-labeled root and subtrees t_i. We write t_Q for the tree of height 1 consisting of only the root labeled by Q.

Types. Recall that an m-type in n variables $\tau_{m,n} \subseteq \mathsf{MSO}$ is a maximal satisfiable set of formulas with quantifier rank at most m and free variables among $x_1, \ldots, x_n$. We will use Hintikka formulas to finitely represent types, as in the following lemma.

▸ **Lemma 7** (Hintikka Lemma [10]). *Given $m \in \mathbb{N}$ and variables $\overline{x} = x_1, \cdots, x_n$, one can compute a* finite *set $H_{m,n}$ of formulas with quantifier rank m and free variables $\overline{x}$ such that:*

- *For every tree t and vertices $v_1, \cdots, v_n \in t$ there is a* unique *$\tau \in H_{m,n}$ such that $t \vDash \tau(\overline{v})$.*
- *If $\tau_1, \tau_2 \in H_{m,n}$ and $\tau_1 \neq \tau_2$ then $\tau_1 \wedge \tau_2$ is unsatisfiable.*
- *If $\tau \in H_{m,n}$ and $\varphi(\overline{x})$ is a formula with $qr(\varphi) \leq m$, then either $\tau \vDash \varphi$ or $\tau \vDash \neg\varphi$.*

Furthermore, given such τ and φ, it is computable which of these two possibilities holds. Elements of $H_{m,n}$ are called (m,n)-Hintikka formulas *and $H_{m,\leq n} = \bigcup_{i \leq n} H_{m,i}$.*

Let us fix a counting term $\#_{\overline{x}}\varphi(\overline{x})$, which we will decompose. In this section, if we omit the quantifier rank m, we mean $m = \mathrm{qr}(\varphi)$. For a tuple $\overline{x} = (x_1, \cdots, x_n)$ of variables we write $[\overline{x}]_l = (\overline{x}_0, \cdots, \overline{x}_l)$ for a partition of $\overline{x}$ into $l+1$ disjoint sets, and $\{[\overline{x}]_l\}$ for the set of all such partitions. Let us first recall the standard MSO decomposition theorem on trees.

▸ **Theorem 8** ([18, 8]). *Let $t = Q - t_1 \cdots t_l$ be a tree and $\varphi(\overline{x})$ an MSO formula. One can compute a finite set $\Phi = \{(\varphi_0(\overline{x}_0), \varphi_1(\overline{x}_1), \cdots, \varphi_l(\overline{x}_l)) \mid (\overline{x}_0, \cdots, \overline{x}_l) \in \{[\overline{x}]_l\}\}$, where all φ_i have a quantifier rank not exceeding that of φ, such that*

$$t \vDash \varphi(\overline{x}) \iff \text{there ex. a } \overline{\varphi} \in \Phi \text{ with } t_Q \vDash \varphi_0(\overline{x}_0) \text{ and } t_i \vDash \varphi_i(\overline{x}_i) \text{ for all } i \in \{1, \ldots, l\}.$$

Observe that the condition above is a disjunction over the $(l+1)$-tuples in Φ of conjunctions over each tuple. We prove a similar decomposition theorem for #MSO in which, intuitively, the disjunctions are replaced by sums and the conjunctions by products. Note that counting terms can count, e.g., the product of the number of as in the left subtree and the number of bs in the right one. But in such case, the free variables in the decomposition are split, and thus the number of terms in a product is limited by the number of free variables. To ensure that assignments are not counted twice, we decompose to Hintikka formulas.

▸ **Theorem 9.** *Let $t = Q - t_1 \cdots t_l$ be a tree and let $\#_{\overline{x}}\varphi(\overline{x})$ be a counting term. Then, for every partition $[\overline{x}]_l = (\overline{x}_0, \cdots, \overline{x}_l)$ of $\overline{x}$, one can compute a finite set $\Psi_{[\overline{x}]_l}$ of $(l+1)$-tuples of Hintikka formulas from $H_{qr(\varphi),\leq|\overline{x}|}$ such that*

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^t = \sum_{[\overline{x}]_l \in \{[\overline{x}]_l\}} \sum_{\overline{\tau} \in \Psi_{[\overline{x}]}} [\![\#_{\overline{x}_0}\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot [\![\#_{\overline{x}_1}\tau_1(\overline{x}_1)]\!]^{t_1} \cdot \ldots \cdot [\![\#_{\overline{x}_l}\tau_l(\overline{x}_l)]\!]^{t_l}.$$

4 Qμ[#MSO] on Tree-Producing Pushdown Systems

In this section we show our main result, namely that Qμ[#MSO] can be effectively evaluated on STSs generated by tree-producing pushdown systems, which generalize both pushdown processes and regular tree grammars with control states and universal application. This is therefore an extension of the classical decidability results for MSO and Lμ on pushdown systems and regular trees to the quantitative setting.

▸ **Definition 10.** An *increasing tree-rewriting rule* for $P \in \Gamma$ has the form $P \leftarrow t$, where t is a Γ-labeled tree of height ≤ 2, i.e. $t = Q - Q_1 \cdots Q_k$ or $t = t_Q$.

Note that increasing tree-rewriting rules are exactly the same as productions in a normalized regular tree grammar. But we apply these rules *universally*, i.e. always to *all* leaves to which a rule can be applied. Formally, for two Γ-labeled trees t_1, t_2 and a rule $r : P \leftarrow t$, we write $t_1 \xrightarrow{r} t_2$, or $r(t_1) = t_2$, if t_2 is obtained from t_1 by replacing every P-labeled leaf by t. We denote the set of all increasing tree-rewriting rules for Γ-labeled trees by $\mathcal{R}_\Gamma$, or just $\mathcal{R}$.

Let us take a starting tree, say t_Q, and apply a sequence of rules $\overline{r} = r_1, \ldots, r_n$ resulting in the tree $t = r_n \cdots r_1(t_Q)$. In the qualitative setting, given an MSO sentence φ, one asks which sequences of rules lead to a tree t such that $t \vDash \varphi$. The set of such sequences of rules turns out to be regular (c.f. [11] Theorem 2) and one can effectively construct an automaton to recognize it. Our main technical result, stated below, generalizes this to the quantitative setting of MSO counting terms using integer counters and affine update functions. Recall that a function $f : \mathbb{N}^k \to \mathbb{N}^k$ is affine if $f(\overline{c}) = A\overline{c} + B$ for some matrix $A \in \mathbb{N}^{k \times k}$ and $B \in \mathbb{N}^k$.

▸ **Theorem 11.** *For all $Q \in \Gamma$, $\varphi \in$ MSO, one can compute $k \in \mathbb{N}$, an initial value $I \in \mathbb{N}^k$, an evaluation vector $E \in \mathbb{N}^{1 \times k}$ and an affine update function $\mathrm{up}_r : \mathbb{N}^k \to \mathbb{N}^k$ for each $r \in \mathcal{R}$ such that, for all finite sequences $r_1, \cdots, r_n \in \mathcal{R}^*$,*

$$[\![\#_{\overline{x}} \varphi(\overline{x})]\!]^{r_n \cdots r_1(t_Q)} = E \cdot (\mathrm{up}_{r_1} \circ \cdots \circ \mathrm{up}_{r_n})(I) = E \cdot (\mathrm{up}_{r_n}(\ldots(\mathrm{up}_{r_1}(I)))\ldots).$$

Also, k, E and the functions up_r depend only on the quantifier rank and free variables of φ.

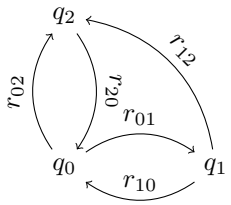
The proof of this theorem is given in Section 6, but we will first show how it can be applied to evaluate Qμ[#MSO] on tree-producing pushdown systems.

▸ **Definition 12.** A *tree-producing pushdown system* (TPPDS) $\mathfrak{T} = (Q, E)$ is a directed graph with $(\mathcal{R} \cup \{\bot\}) \times (\mathcal{R} \cup \{\mathsf{pop}, \varepsilon\})$-labeled edges, i.e. $E \subseteq Q \times (\mathcal{R} \cup \{\bot\}) \times (\mathcal{R} \cup \{\mathsf{pop}, \varepsilon\}) \times Q$.

A TPPDS $\mathfrak{T} = (Q, E)$ with initial tree t_P gives rise to the infinite structure transition system $\mathfrak{S}(\mathfrak{T}) = (S, \Delta, \mathfrak{m})$ with states $S = Q \times \mathcal{R}^*$, the structure assignment $\mathfrak{m}(q, \overline{r}) = \overline{r}(t_P)$ and transitions as in a pushdown process:

$$\begin{aligned}
\Delta = \quad & \big\{\big((q,\varepsilon), r', (q', r')\big) \mid q, q' \in Q,\ (q, \bot, r', q') \in E\big\} \\
& \cup \big\{\big((q, \overline{r}r), r', (q', \overline{r}rr')\big) \mid q, q' \in Q,\ (q, r, r', q') \in E\big\} \\
& \cup \big\{\big((q, \overline{r}r), \varepsilon, (q', \overline{r}r)\big) \mid q, q' \in Q,\ (q, r, \varepsilon, q') \in E\big\} \\
& \cup \big\{\big((q, \overline{r}r), \mathsf{pop}, (q', \overline{r})\big) \mid q, q' \in Q,\ (q, r, \mathsf{pop}, q') \in E\big\}.
\end{aligned}$$

Observe that standard pushdown systems are subsumed by TPPDS. To obtain the stack in the corresponding STS one uses rules where the right-hand side is a tree of branching degree 1, i.e. a word. Properties like stack unboundedness can be formulated in Qμ[#MSO] as was shown in Section 2.



$r_{01} = P \leftarrow P - PPP$
$r_{02} = P \leftarrow Q - PP$
$r_{10} = \mathsf{pop}$
$r_{12} = P \leftarrow P$
$r_{20} = P \leftarrow Q - PP$

run	$q_0q_1q_0q_2$	$q_0q_1q_2$
tree	Q P P	P P P P

Figure 1 Example of a TPPDS with 2 initial runs from (q_0, t_P).

▸ **Example 13.** Consider the TPPDS in Figure 1. The resulting trees of two initial runs, starting in state q_0 with initial tree t_P, are given in the table on the right.

Let us remark that the underlying transition graph of the STS generated by a TPPDS is always a pushdown graph, no matter whether one uses rules that generate trees or only words. But there is a substantial difference when quantitative questions are asked, e.g. about the size of the structure generated by a run. Trees allow to model richer classes of systems (c.f. Corollary 16), but their size can grow exponentially in the number of rewriting steps and thus they require more refined counting techniques than words.

Our main decidability result, stated below, is a consequence of Theorem 11, the existence of model-checking games (Theorem 5), pop elimination (Theorem 6), and an algorithm to solve counter parity games [1].

▸ **Theorem 14.** *Given a formula $\psi \in Q\mu[\#MSO]$, a TPPDS $\mathfrak{T} = (Q, E)$, a state $q \in Q$ and a starting symbol $P \in \Gamma$, one can compute $[\![\psi]\!]^{\mathfrak{G}(\mathfrak{T})}(q, t_P)$.*

Proof. First, we transform ψ to negation normal form. By Theorem 5, to compute $[\![\psi]\!]^{\mathfrak{G}(\mathfrak{T})}(q, t_P)$ it suffices to determine the value of the game $\mathsf{MC} = \mathsf{MC}(\mathfrak{G}(\mathfrak{T}), (q, t_P)) = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ from $((q, \varepsilon), \psi)$. By Theorem 5 (1), the vertices in V have the form $((q, \overline{r}), \psi)$ or $(\pm\infty)$, and V can be infinite. Observe that MC is a pushdown QPG with states $Q \times \mathsf{Sub}(\psi)$. By Theorem 6, MC can be transformed into an equivalent pop-free pushdown QPG MC' with states Q'. We define a finite game $\widehat{\mathsf{MC}}$ with labeled edges which removes the dependence of moves on the top stack symbol and still represents MC' in the sense that the payoff of a play in MC' can be computed from the sequence of visited labels in $\widehat{\mathsf{MC}} := (\widehat{V}, \widehat{V}_{\max}, \widehat{V}_{\min}, \widehat{E}, \widehat{\Omega})$. The positions $\widehat{V} := (Q' \times (\mathcal{R} \cup \{\varepsilon\})) \cup \{(\infty), (-\infty)\}$ have generally the form (q, r) and simply omit all but the last rule r from the stack in the position in MC'. The $(\mathcal{R} \cup \{\varepsilon\})$-labeled edge relation:

$$\begin{aligned}\widehat{E} \;:=\; & \big\{\big((q,r), r', (q', r')\big) \mid \big((q, \overline{r}r), (q', \overline{r}rr')\big) \in E'\big\} \\ & \cup \big\{\big((q,r), \varepsilon, (q', r)\big) \mid \big((q, \overline{r}r), (q', \overline{r}r)\big) \in E'\big\} \\ & \cup \big\{\big((q,r), r, (\pm\infty)\big) \mid \big((q, \overline{r}r), (\pm\infty)\big) \in E'\big\}.\end{aligned}$$

Note that from each position (q, r) in $\widehat{\mathsf{MC}}$ there are exactly the same possible moves as from each position $(q, \overline{r}r)$ in MC'. Since the stack in a position in MC' is a function of the starting tree t_P and the observed rules $\overline{r}$, there is a one-to-one correspondence between plays and strategies in both games.

We define the coloring function $\widehat{\Omega}(q, r) = \Omega'(q, \overline{r}r)$ for any tree $\overline{r}$, and it is well defined since in a QPG the value $\Omega(q, \overline{r}r)$ depends only on q, not on the stack. The payoff of an infinite play in $\widehat{\mathsf{MC}}$ depends on the minimal color seen infinitely often, exactly as in MC', and is thus equal to the payoff of the corresponding play in the original model-checking game. For a finite play $\pi = v_0 \cdot r_1 \cdot v_1 \cdot r_2 \cdot \cdots \cdot r_n \cdot v_n$, the last position, by Theorem 5 (2), either

corresponds to a formula $\#_{\overline{x}}\varphi(\overline{x})$, or to a formula $\neg\#_{\overline{x}}\varphi(\overline{x})$, or to $(\pm\infty)$. In the first case we set the payoff in $\widehat{\mathsf{MC}}$ to $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)}$, in the second one to $-1\cdot[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)}$ and for $(\pm\infty)$ to $\pm\infty$. This definition of payoffs clearly ensures that corresponding plays in MC' and $\widehat{\mathsf{MC}}$ result in the same outcome, and due to the one-to-one correspondence mentioned above, the value of MC' from $((q,\varepsilon),\psi)$ is the same as the value of $\widehat{\mathsf{MC}}$ from $((q,\varepsilon),\psi)$, which we compute below, thus also solving MC.

Let $\{\varphi_1,\dots,\varphi_l\}$ be an enumeration of the MSO formulas in ψ which are counted, i.e. of $\{\varphi \mid \#_{\overline{x}}\varphi(\overline{x}) \in \mathsf{Sub}(\psi)\}$. By Theorem 11, for each such φ_i there is the corresponding dimension k^i, initial values I^i, evaluation vector E^i and update functions up_r^i. We combine the initial values to an aggregate initial $I = \langle I^1,\dots,I^l\rangle$, which is a vector of dimension $k = k^1+\ldots+k^l$. The aggregate update functions are also composed component-wise:

$$\mathrm{up}_r(\langle\overline{c}^1,\dots,\overline{c}^l\rangle) = \langle \mathrm{up}_r^1(\overline{c}^1),\dots \mathrm{up}_r^l(\overline{c}^l)\rangle,$$

and we extend each evaluation vector E^i to a vector $\widehat{E}^i$ of dimension k by filling it with 0s on all dimensions except for the k^i ones. By Theorem 11, we have that

$$[\![\#_{\overline{x}}\varphi_i(\overline{x})]\!]^{r_n\cdots r_1(t_P)} = \widehat{E}^i\cdot(\mathrm{up}_{r_1}\circ\cdots\circ\mathrm{up}_{r_s})(I). \tag{1}$$

Let us thus use the aggregate functions to transform $\widehat{\mathsf{MC}}$ into a game $\widetilde{\mathsf{MC}}$, played on the same arena, in which in each move an affine function is applied to a vector of k integers. To construct $\widetilde{\mathsf{MC}}$, we replace every edge label r in $\widehat{\mathsf{MC}}$ by the function up_r and the payoff function λ in $\widetilde{\mathsf{MC}}$, depending on the current value of the vector $\overline{c}$, is given by

$$\lambda(s,\overline{c}) = \begin{cases} \pm\infty & \text{if } s = (\pm\infty),\\ E^i\cdot\overline{c} & \text{if } s = (p,\ \#_{\overline{x}}\varphi_i(\overline{x})),\\ -E^i\cdot\overline{c} & \text{if } s = (p,\ \neg\#_{\overline{x}}\varphi_i(\overline{x})).\end{cases}$$

By (1), the payoffs of corresponding plays in $\widehat{\mathsf{MC}}$ and in $\widetilde{\mathsf{MC}}$ are the same, and due to the one-to-one correspondence of moves, plays and strategies we also get that the value of $\widetilde{\mathsf{MC}}$ from $((q,\varepsilon),\psi)$ starting with vector I is the same as the value of $\widehat{\mathsf{MC}}$ from this position, and thus equal to $[\![\psi]\!]^{\mathfrak{S}(\mathfrak{T})}(q,t_P)$. But the game $\widetilde{\mathsf{MC}}$ is a special case of a *counter parity game* with k counters [1], and, as proved in [1], its value can be computed. ◂

The above theorem demonstrates that $\mathsf{Q}\mu[\#\mathsf{MSO}]$ indeed allows to apply the methods known for qualitative logics to the quantitative case. As TPPDS subsume pushdown systems, we obtain the following corollary.

▸ **Corollary 15.** *Given a formula $\psi \in \mathsf{Q}\mu[\#\mathsf{MSO}]$, a pushdown process $\mathcal{A} = (Q,\hookrightarrow)$ generating an STS $\mathfrak{P}$, and a state $q_0 \in Q$, one can compute $[\![\psi]\!]^{\mathfrak{P}}(q_0,\bot)$.*

Furthermore, the class of tree-producing pushdown systems includes finite systems and, as shown in [11], MSO formulas on separated structure rewriting systems can also be reduced to formulas to be checked on a TPPDS.

▸ **Corollary 16.** *Given a formula $\psi \in \mathsf{Q}\mu[\#\mathsf{MSO}]$, a separated structure rewriting system (c.f. [11]) generating an STS $\mathfrak{S}$, and an initial state $s \in \mathfrak{S}$, one can compute $[\![\psi]\!]^{\mathfrak{S}}(s)$.*

Before, we reduced model-checking $\mathsf{L}\mu[\mathsf{MSO}]$ to computing the values of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ formulas. Thus, we can say that Corollary 15 strictly generalizes previous results on model-checking $\mathsf{LTL}[\mathsf{MSO}]$ and $\mathsf{L}\mu[\mathsf{MSO}]$ on pushdown systems [13, 6] and Corollary 16 subsumes the result from [11] for $\mathsf{L}\mu[\mathsf{MSO}]$ on separated structure rewriting systems. On the quantitative side, these corollaries also subsume the result from [7] for $\mathsf{Q}\mu$ on finite systems and model-checking parity conditions with unboundedness on pushdown systems [2, 9].

5 Eliminating pop from Pushdown QPGs

In this section, we prove Theorem 6 using methods similar to the ones developed in [19] and later in [3] and [15], but with a construction symmetric with respect to the players.

▸ **Theorem 6.** *For every pushdown QPG $\mathcal{G}$ over $\mathcal{A} = (Q, \hookrightarrow)$ one can compute a finite set Q', $q'_0 \in Q'$, a pop-free pushdown process $\mathcal{A}' = (Q \times Q', \hookrightarrow')$ and a QPG $\mathcal{G}'$ over $\mathcal{A}'$ such that $\mathsf{val}_{\mathcal{G}}(q, \varepsilon) = \mathsf{val}_{\mathcal{G}'}((q, q'_0), \varepsilon)$ for each $q \in Q$.*

Construction of the game $\mathcal{G}'$. Intuitively, $\mathcal{G}'$ simulates $\mathcal{G}$, but in each push-move the player makes claims about minimal colors that will be seen in $\mathcal{G}$ until the stack pops back to the same content, if it does. The opponent is then asked to either proceed as if the claim happened – moving to a claimed state with one of the claimed colors – or to allow the push and accept to lose if a pop satisfying the claim occurs later.

To construct the set Q', let d be the maximal color assigned by Ω in $\mathcal{G}$ and $[d] = \{0, \dots, d\}$. We consider the set of *claims* defined as $\mathcal{C} = \{C : Q \to \mathcal{P}([d])\}$, i.e. a claim C assigns to each state a set of colors. We define Q' as the disjoint union of three kinds of states, $Q' = \{\bot\} \cup Q'_0 \cup Q'_1$: the state $\bot$ for the empty stack, $Q'_0 = [d] \times \mathcal{C} \times \{\max, \min\}$ for positions where players make claims, and the set $Q'_1 = \Gamma \times \mathcal{C} \times \{\max, \min\} \times (\{\bot\} \cup Q'_0)$ for positions where players answer to claims.

We construct the relation $\hookrightarrow'$ and the game $\mathcal{G}'$ in the following way. Positions $(q, \bot, \bot)$ and (q, m, C, p, s) belong to the same player to whom q belongs in $\mathcal{G}$. The epsilon-moves from $\hookrightarrow$, i.e. the ones which do not change the stack, are preserved in $\mathcal{G}'$ and lead to $(q', \bot, \bot)$ or, respectively, to $(q', \min(m, \Omega(q')), C, p, s)$ updating the minimal color for C. The pop-moves do not occur in $\mathcal{G}'$, but, if a pop-move to q' was allowed in $\mathcal{G}$ from (q, s), then in $\mathcal{G}'$ there is a move from (q, m, C, p, s) to a sink position. This sink position is winning for player p who made the claim if the claim was true, and winning for the opponent otherwise. Formally, it has payoff $+\infty$ if $p = \max$ and $\min(m, \Omega(q')) \in C(q')$ (true claim) or if $p = \min$ and $\min(m, \Omega(q')) \notin C(q')$ (false claim), and $-\infty$ otherwise. The payoff at a terminal position (q, x, s) in $\mathcal{G}'$ is the same as the payoff in (q, s) in $\mathcal{G}$.

The push-moves from $\mathcal{G}$ translate to claims made in $\mathcal{G}'$ as depicted in Figure 2. If $\hookrightarrow$ allowed to push $a \in \Gamma$ in $\mathcal{G}$ from (q, s) leading to a state q', then in $\mathcal{G}'$ we add the moves from (q, x, s), for $x = \bot$ or $x \in Q'_0$, to (q', a, C, p, x, s), where C is any claim and p is the player to whom q belongs in $\mathcal{G}$, i.e. the one who makes the claim.

Positions where claims are answered, i.e. of the form (q', a, C, p, x, s), belong to the opponent of the player p who just made the claim. From each such position there is exactly one possible push-move leading to $(q', \Omega(q'), C, p, sa)$, i.e. the push is made as intended. Additionally, for each color d_c and state $q_c \in Q$ such that $d_c \in C(q_c)$, there is a move to the position (q_c, x', s) which corresponds to playing as in the claim. In this case, if $x = \bot$ then $x' = \bot$ as well and we set $\Omega(q_c, \bot, \bot) = \min(\Omega(q_c), d_c)$. If $x = (m, C_0, p_0)$, then $x' = (\min(m, d_c), C_0, p_0)$, i.e. the minimal color is updated, and we again set $\Omega(q_c, x', s) = \min(\Omega(q_c), d_c)$.

Correctness of the construction. Let σ be a strategy of one of the players in $\mathcal{G}$. We define the corresponding truthful strategy σ' in $\mathcal{G}'$ by induction on the length of play prefixes. Also, to each play prefix π' consistent with σ' we assign a corresponding play prefix π in $\mathcal{G}$ consistent with σ. Intuitively, when the player is supposed to make a claim in $\mathcal{G}'$, he considers all plays extending π in $\mathcal{G}$ and consistent with σ and, in σ', chooses a claim with the colors that really occur. When the opponent makes a claim in $\mathcal{G}'$ and indeed there is an extension consistent with σ which returns to the same stack in the claimed state and

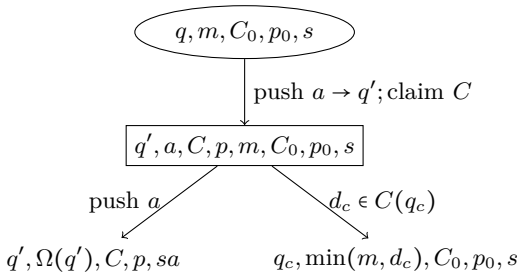


Figure 2 Claims and moves corresponding to push operations.

color, then σ' accepts this claim, and the corresponding play in $\mathcal{G}$ is prolonged to the point in which the stack pops back to the claimed state. If no play is consistent with the claim of the opponent, i.e. it is false, then the push-move is made. A formal proof of correctness is similar to the one in Chapter 5 of [17].

6 #MSO Evaluation Using Counters

In this section we prove Theorem 11. Let us therefore fix a symbol $P \in \Gamma$ and an MSO formula φ. This also fixes the quantifier rank $m = \mathrm{qr}(\varphi)$ and $\overline{x}$ as the free variables of φ. We will prove, for an arbitrary sequence $r_1, \dots, r_n \in \mathcal{R}^*$, that $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_P)} = E \cdot (\mathrm{up}_{r_1} \circ \cdots \circ \mathrm{up}_{r_n})(I)$.

The proof will be by induction on the length of the sequence $r_1, \dots, r_n$; the appropriate I, E and up_r will be constructed in the process. For clarity, we omit the easy case of rules $P \leftarrow t_Q$ and we first assume that the types of the subtrees $r_n \cdots r_{i+1}(t_P)$ are known. In subsection 6.1 we show how these types can be guessed and checked afterwards, and in 6.2 we provide the final construction and proof of Theorem 11.

Notation. We consider the sequence of rules $r_1, \dots, r_n$ and denote the tree after i rewritings by t_i, i.e. the rewriting sequence can be written as $t_P \xrightarrow{r_1} t_1 \xrightarrow{r_2} \cdots \xrightarrow{r_n} t_n$. We will also evaluate terms on the subtrees which result from rewriting a symbol $P \in \Gamma$ from the i-th step on, i.e. on $r_n \cdots r_{i+1}(t_P)$. For a Hintikka formula $\tau \in H_{m,\leq|\overline{x}|}$ and a symbol $P \in \Gamma$ we write $[\![\tau(\overline{y}), P]\!]^i := [\![\#_{\overline{y}}\tau(\overline{y})]\!]^{r_n \cdots r_{i+1}(t_P)}$, i.e. $[\![\tau, P]\!]^i$ is the number of tuples $\overline{a}$ such that $\tau(\overline{a})$ holds on the subtree generated from P in the last $n - i$ rewriting steps.

By Λ we denote the set of all unordered sequences $\left(\overline{\tau, P}\right) = \{(\tau_1(\overline{y}_1), P_1), \cdots, (\tau_k(\overline{y}_k), P_k)\}$, where $P_i \in \Gamma$, $(\overline{y}_1, \cdots, \overline{y}_k)$ is a partition into *nonempty* sets $\overline{y}_i \neq \varnothing$ of a subset $\overline{y} \subseteq \overline{x}$ of free variables of φ, and each $\tau_i \in H_{m,\leq|\overline{x}|}$ is a Hintikka formula with free variables $\overline{y}_i$. We will use elements of Λ as *indices*, i.e. we will operate on a vector $c \in \mathbb{N}^{|\Lambda|}$ and we will write $c\left[\left(\overline{\tau, P}\right)\right]$ for the number in c corresponding to position $\left(\overline{\tau, P}\right)$. Let us prove the first induction step.

▸ **Lemma 17** (First step). *There exists a $c \in \mathbb{N}^{|\Lambda|}$ such that*

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_P)} = \sum_{\left(\overline{\tau,P}\right) \in \Lambda} c\left[\left(\overline{\tau, P}\right)\right] \cdot \prod_{(\tau,P) \in \left(\overline{\tau,P}\right)} [\![\tau, P]\!]^1.$$

Proof. Assume that $r_1 = P \leftarrow Q - Q_1 \cdots Q_l$. By Lemma 9, $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_P)}$ equals

$$\sum_{[\overline{x}]_l \in \{[\overline{x}]_l\}} \sum_{\overline{\tau} \in \Psi_{[\overline{x}]_l}} [\![\#_{\overline{x}_0}\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot [\![\#_{\overline{x}_1}\tau_1(\overline{x}_1)]\!]^{r_n \cdots r_2(Q_1)} \cdot \ldots \cdot [\![\#_{\overline{x}_l}\tau_l(\overline{x}_l)]\!]^{r_n \cdots r_2(Q_l)}.$$

Using the notation introduced above, each product in the inner sum can be written as $[\![\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot \prod_{i=1}^{l} [\![\tau_i(\overline{x}_i), Q_i]\!]^1$.

In each inner product $\prod_{i=1}^{l}[\![\tau_i(\overline{x}_i), Q_i]\!]^1$, the factors $[\![\tau_i(\overline{x}_i), Q_i]\!]^1$ in which τ_i is a sentence are known to be either 0 or 1. If they are 1, they can safely be omitted, otherwise the whole product becomes 0. Furthermore, if, after the removal, two inner products coincide for two different τ_0 and τ_0', then the sum of the two outer products can be written as

$$[\![\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot \prod_{i=1}^{l}[\![\tau_i(\overline{x}_i), Q_i]\!]^1 + [\![\tau_0'(\overline{x}_0)]\!]^{t_Q} \cdot \prod_{i=1}^{l}[\![\tau_i(\overline{x}_i), Q_i]\!]^1$$
$$= ([\![\tau_0]\!]^{t_Q} + [\![\tau_0']\!]^{t_Q}) \cdot \prod_{i=1}^{l}[\![\tau_i(\overline{x}_i), Q_i]\!]^1.$$

We set $c\big[\big(\overline{\tau, P}\big)\big]$ to the sum of the $[\![\tau_0(\overline{x}_0)]\!]^{t_Q}$ over all products where the inner products coincide with $\big(\overline{\tau, P}\big)$, i.e. to the sum of those where $\{(\tau_i, Q_i) \mid 1 \le i \le l, \overline{x}_i \neq \varnothing\} = \big(\overline{\tau, P}\big)$. ◂

Note that the vector c constructed above depends only on the first rule r_1 and which *sentences* τ_i hold in $r_n \cdots r_2(t_P)$, for each P. We will remove the dependence on the types τ_i in subsection 6.1. First, we show that the sum of the above form can be maintained further into the rewriting sequence and that the vector c only needs to be updated in a linear way.

▸ **Lemma 18** (Induction step). *Let $c \in \mathbb{N}^{|\Lambda|}$ be a vector such that*

$$(S) := [\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_P)} = \sum_{(\overline{\tau,P}) \in \Lambda} c\big[\big(\overline{\tau, P}\big)\big] \cdot \prod_{(\tau,P) \in (\overline{\tau,P})} [\![\tau, P]\!]^{i-1}.$$

Then there exists a vector $\widetilde{c} \in \mathbb{N}^{|\Lambda|}$ such that

$$(S) = \sum_{(\overline{\tau,P}) \in \Lambda} \widetilde{c}\big[\big(\overline{\tau, P}\big)\big] \cdot \prod_{(\tau,P) \in (\overline{\tau,P})} [\![\tau, P]\!]^{i},$$

and each $\widetilde{c}\big[\big(\overline{\tau, P}\big)\big]$ can be computed as a linear combination of the numbers $c\big[\big(\overline{\tau, P}\big)\big]$.

Proof. Let $r_i = R \leftarrow Q - Q_1 \cdots Q_l$ be the i-th rule. By applying Lemma 17 to the sequence $r_n \cdots r_i(t_R)$, we get that for all τ there exists c' such that:

$$[\![\tau(\overline{y}), R]\!]^{i-1} = \sum_{(\overline{\tau,P}) \in \Lambda} c'\big[\big(\overline{\tau, P}\big)\big] \prod_{(\tau,P) \in (\overline{\tau,P})} [\![\tau, P]\!]^{i}. \tag{2}$$

For symbols $P \neq R$, the trees $r_n \cdots r_i(t_P)$ and $r_n \cdots r_{i+1}(t_P)$ coincide, thus $[\![\tau, P]\!]^{i-1} = [\![\tau, P]\!]^{i}$.

Recall the sum for position $i-1$. This sum can be split into two parts, namely over those $\big(\overline{\tau, P}\big)$ where R does not occur, and those where it does:

$$(S) = \underbrace{\sum_{(\overline{\tau,P}) \in \Lambda, R \notin \overline{P}} c\big[\big(\overline{\tau, P}\big)\big] \cdot \prod_{(\tau,P) \in (\overline{\tau,P})} [\![\tau, P]\!]^{i-1}}_{(A)}$$
$$+ \underbrace{\sum_{(\overline{\tau,P}) \in \Lambda, R \in \overline{P}} c\big[\big(\overline{\tau, P}\big)\big] \cdot \prod_{(\tau,P) \in (\overline{\tau,P})} [\![\tau, P]\!]^{i-1}}_{(B)}.$$

For (A), it holds that all $[\![\tau, P]\!]^{i-1}$ are equal to their corresponding $[\![\tau, P]\!]^{i}$. For (B), the sum is updated: when considering a summand $c\big[\big(\overline{\tau, P}\big)\big] \prod_{\tau,P} [\![\tau, P]\!]^{i-1}$ with $R \in \overline{P}$, each factor $[\![\tau, R]\!]^{i-1}$ is replaced by the corresponding sum from (2), resulting in:

$$c\big[\big(\overline{\tau, P}\big)\big] \left(\prod_{\tau, P, P \neq R} [\![\tau, P]\!]^{i} \right) \cdot \left(\prod_{\tau, R} \underbrace{\left(\sum_{(\overline{\tau,P})'} c'\big[\big(\overline{\tau, P}\big)'\big] \prod_{\tau', P'} [\![\tau', P']\!]^{i} \right)}_{\text{right-hand side of (2) for } [\![\tau, R]\!]^{i-1}} \right).$$

As in the decomposition for a $[\![\tau, R]\!]^{i-1}$ the only free variables $\overline{z}$ are those of τ, it follows that $c'\left[\left(\overline{\tau, P}\right)\right] = 0$ for all $\left(\overline{\tau, P}\right)$ where some $(\tau, P) \in \left(\overline{\tau, P}\right)$ has free variables other than $\overline{z}$. Note that, for all $[\![\tau, P]\!]^i$ with $R \neq P$ that already were there before inserting the right-hand sides of (2), the τ have free variables disjoint from $\overline{z}$. Thus, when multiplied out after omitting all summands with $c'[\left(\overline{\tau, P}\right)] = 0$, we obtain a sum over Λ of c-weighted products. This can be joined with (A) to obtain $(S) = \sum_{\left(\overline{\tau,P}\right)\in\Lambda} \widetilde{c}\left[\left(\overline{\tau, P}\right)\right] \cdot \prod_{(\tau,P)\in\left(\overline{\tau,P}\right)} [\![\tau, P]\!]^i$, where $\widetilde{c}$ is as follows.

Calculating $\widetilde{c}$. We show now how to linearly combine different entries of c to get the entries of $\widetilde{c}$. To this end, we create, for all $\sigma \in H_{m,\leq|\overline{x}|}$, a set $\mathcal{D}_\sigma$, collecting all $\left(\overline{\tau, P}\right)$ from the right-hand side of (2) for $[\![\sigma, R]\!]^{i-1}$ along with the respective $c'\left[\left(\overline{\tau, P}\right)\right]$.

$$\mathcal{D}_\sigma = \left\{ \left(\left(\overline{\tau, P}\right), c'\left[\left(\overline{\tau, P}\right)\right]\right) \mid \left(\overline{\tau, P}\right) \text{occurs in right-hand side of (2) for } [\![\sigma, R]\!]^{i-1} \right\}.$$

Since only positive values of $c'\left[\left(\overline{\tau, P}\right)\right]$ are of interest, we collect all $\mathcal{D}_\sigma$ in the following set:

$$\mathcal{D} = \left\{ \left(\sigma, \left(\overline{\tau, P}\right), d\right) \mid \sigma \in H_{m,\leq|\overline{x}|},\ \left(\left(\overline{\tau, P}\right), d\right) \in \mathcal{D}_\sigma,\ d > 0 \right\}.$$

To determine the value of $\widetilde{c}\left[\left(\overline{\tau, P}\right)\right]$, all those $c\left[\left(\overline{\tau, P}\right)'\right]$ have to be considered in which a factor $[\![\sigma, R]\!]^{i-1}$ is replaced, and, when multiplying out, a product over $\left(\overline{\tau, P}\right)$ results. We do this by adding — for every subset $\left(\overline{\tau, P}\right)'$ of $\left(\overline{\tau, P}\right)$ and every τ such that $c'\left[\left(\overline{\tau, P}\right)'\right]$ appears on the right-hand side of (2) for $[\![\sigma, R]\!]^{i-1}$ — the entries in c for the set obtained from $\left(\overline{\tau, P}\right)$ by replacing $\left(\overline{\tau, P}\right)'$ with (σ, R), weighted with the respective $c'\left[\left(\overline{\tau, P}\right)'\right]$. This amounts to a sum over the set $\mathcal{D}$ defined above. In addition, if $R \notin \overline{P}$, we have to copy the old counter value, as in the respective subtrees nothing was changed. Thus, let $\mathrm{old}\left[\left(\overline{\tau, P}\right)\right] = c\left[\left(\overline{\tau, P}\right)\right]$ if $R \notin \overline{P}$ and $\mathrm{old}\left[\left(\overline{\tau, P}\right)\right] = 0$ otherwise. We obtain the following equation.

$$\widetilde{c}\left[\left(\overline{\tau, P}\right)\right] = \mathrm{old}\left[\left(\overline{\tau, P}\right)\right] + \sum_{\left(\overline{\tau,P}\right)'\subseteq\left(\overline{\tau,P}\right)} \sum_{\left(\sigma,\left(\overline{\tau,P}\right)',d\right)\in\mathcal{D}} d \cdot c\left[\left(\overline{\tau, P}\right) \setminus \left(\overline{\tau, P}\right)' + (\sigma, R)\right]. \tag{3}$$

◂

Note that the construction of $\widetilde{c}$ above depends only on the vectors c' from the right-hand sides of (2), which are in turn obtained from Lemma 17 and depend only on the rule r_i and which Hintikka sentences hold in $r_n \cdots r_{i+1}(t_P)$, for each P.

6.1 Guessing Types of Subtrees

In this subsection, we remove the dependency on the Hintikka sentences described above by *correctly guessing* the types of the subtrees. To this end, we will extend the vector c we operate on by additional entries, corresponding to *type functions* which guess the types.

We represent types by Hintikka sentences: for any tree t, the m-type $\mathrm{tp}^m(t)$ is the unique $\tau \in H_{m,0}$ such that $t \vDash \tau$.

▸ **Definition 19.** For a set of symbols Γ and a natural number m, a *type function* $L : \Gamma \to H_{m,0}$ assigns an m-type to every symbol from Γ.

Notice that, for every finite Γ and every m, there are only finitely many type functions, thus the set $\mathcal{L} = \{L : \Gamma \to H_{m,0}\}$ (for our fixed Γ and $m = \mathrm{qr}(\varphi)$) is finite. Denote by L_F the unique type function that assigns to every P the type of t_P. For a sequence $r_1, \ldots, r_n$ of rules from $\mathcal{R}$ we say that $L_0, \ldots, L_n$ is the *compatible* sequence of type functions if $L_i(P) = \mathrm{tp}^m(r_n \cdots r_{i+1}(t_P))$ for all P. Note that $L_n = L_F$. We show that compatible type functions can be computed *backwards* and in a *uniform way*.

▸ **Lemma 20.** *There exists a computable function* $\mathrm{pre}\colon \mathcal{L} \times \mathcal{R} \to \mathcal{L}$ *such that for all sequences* $r_1, \ldots, r_n$ *of rules,* pre *together with* L_F *induces a compatible sequence, i.e.* $L_0, \cdots, L_n$ *with* $L_n = L_F$ *and* $L_{i-1} = \mathrm{pre}(L_i, r_i)$ *is compatible.*

Instead of updating the vector c depending on the correct types, we will now maintain a separate vector c_L for every type function L. Each vector c_L will be updated as if the types given by L were the correct ones, and the types will be updated by pre, i.e. we define $\widetilde{c_L}\left[\left(\overline{\tau, P}\right)\right]$ exactly as in (3) but using $c_{\mathrm{pre}(L,r_i)}\left[\left(\overline{\tau, P}\right)\right]$ for $c\left[\left(\overline{\tau, P}\right)\right]$. At the end, the correct $L = L_F$ will be chosen and the above lemma guarantees correctness, as proved below.

6.2 Proof of Theorem 11

▸ **Theorem 11.** *For all* $Q \in \Gamma$, $\varphi \in$ *MSO, one can compute* $k \in \mathbb{N}$, *an initial vector* $I \in \mathbb{N}^k$, *an evaluation vector* $E \in \mathbb{N}^{1\times k}$ *and an affine update function* $\mathrm{up}_r\colon \mathbb{N}^k \to \mathbb{N}^k$ *for each* $r \in \mathcal{R}$ *such that, for all finite sequences* $r_1, \cdots, r_n \in \mathcal{R}^*$,

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_Q)} = E \cdot (\mathrm{up}_{r_1} \circ \cdots \circ \mathrm{up}_{r_n})(I).$$

Also, E, k *and the functions* up_r *depend only on the quantifier rank and free variables of* φ.

Proof. Let m be the quantifier rank of φ and Λ and $\mathcal{L}$ as defined above. We set $k := |\mathcal{L}| \cdot |\Lambda|$. To define the initial vector I, we compute the Hintikka disjunction $\tau_1 \vee \cdots \vee \tau_s \equiv \varphi$. For every $L \in \mathcal{L}$, we define the vector $I_L \in \mathbb{N}^{|\Lambda|}$ in which all positions are set to 0, except for those where $\left(\overline{\tau, P}\right) = (\tau_i, Q)$, for $1 \le i \le s$, which are set to 1. We fix an enumeration $L_1, \cdots, L_{|\mathcal{L}|}$ of $\mathcal{L}$, and set $I := \langle I_{L_1}, \cdots, I_{L_{|\mathcal{L}|}}\rangle \in \mathbb{N}^k$.

Let pre be the function from Lemma 20. For every $r \in \mathcal{R}$, we define the update function $\mathrm{up}_r(\langle c_1, \cdots, c_{|L|}\rangle) := \langle \widetilde{c_1}, \cdots, \widetilde{c_{|L|}}\rangle$, where $\widetilde{c_L}\left[\left(\overline{\tau, P}\right)\right]$ is set as in (3) but with $c_{\mathrm{pre}(L,r_i)}$ used in place of c. By Lemmas 18 and 20, after the last step, the vector c_{L_F} contains the entries evaluated for the compatible type sequence, and thus the value of the counting term

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_Q)} = \sum_{(\overline{\tau,P})\in\Lambda} c_{L_F}\left[\left(\overline{\tau, P}\right)\right] \cdot \prod_{(\tau,P)\in\left(\overline{\tau,P}\right)} [\![\tau]\!]^{t_P}.$$

Note that every $[\![\tau]\!]^{t_P}$ is a constant and so is $\pi_{\left(\overline{\tau,P}\right)} := \prod_{(\tau,P)\in\left(\overline{\tau,P}\right)} [\![\tau]\!]^{t_P}$ for every $\left(\overline{\tau, P}\right)$. Thus, the above sum is equal to the product of the updated vector c with the row vector E, which has $\pi_{\left(\overline{\tau,P}\right)}$ in the component corresponding to $c_{L_F}\left[\left(\overline{\tau, P}\right)\right]$ and 0 everywhere else. ◂

7 Conclusion

The question how well-known methods used for regular languages can be extended to the quantitative setting has been approached recently from several angles. From the side of automata, this problem has been investigated in [4] and related papers with focus on counting, and from the logic side, progress has been made in the study of quantitative temporal logics [7]. The counting logic Qμ[#MSO] introduced in this work is the first formalism which exhibits the desirable properties of a well-behaved quantitative temporal logic and allows at the same time to apply decomposition techniques, the logical counterpart of automata. As we show, the value of a Qμ[#MSO] formula is computable on an important class of infinite structure transition systems which generalize both pushdown systems and regular tree grammars. This raises interesting new questions whether this result can be extended, e.g. if the value of a Qμ[#MSO] formula is computable on higher-order pushdown systems or recursion schemes, and in general which results for Boolean logics can be generalized to the

counting case. The decomposition theorem we proved for $\mathsf{Q}\mu[\#\mathsf{MSO}]$ allows us to conjecture that many results which rely on automata techniques can indeed be extended to $\mathsf{Q}\mu[\#\mathsf{MSO}]$, and that it may lead to a canonical quantitative counting logic for regular languages.

Acknowledgment. We are very grateful to Igor Walukiewicz for pointing out that the methods from [19] can also be applied in the quantitative setting, as is done in Section 5.

References

1. D. Berwanger, Ł. Kaiser, and S. Lessenich. Solving counter parity games. In *Proc. of MFCS'12*, LNCS. Springer, 2012. To appear.
2. A.-J. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In *Proc. of FSTTCS'03*, vol. 2914 of *LNCS*, pp. 88–99. Springer, 2003.
3. A. Carayol, M. Hague, A. Meyer, C.-H. L. Ong, and O. Serre. Winning regions of higher-order pushdown games. In *Proc. of LICS'08*, pp. 193–204. IEEE Computer Society, 2008.
4. T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proc. of ICALP'09, Part II*, vol. 5556 of *LNCS*, pp. 139–150. Springer, 2009.
5. J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. of CAAP'94*, vol. 787 of *LNCS*, pp. 115–129. Springer, 1994.
6. J. Esparza, A. Kučera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355 – 376, 2003.
7. D. Fischer, E. Grädel, and Ł. Kaiser. Model checking games for the quantitative μ-calculus. *Theory Comput. Syst.*, 47(3):696–719, 2010.
8. T. Ganzow and Ł. Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In *Proc. of CSL'10*, vol. 6247 of *LNCS*, pp. 366–380. Springer, 2010.
9. H. Gimbert. Parity and exploration games on infinite graphs. In *Proc. of CSL'04*, vol. 3210 of *LNCS*, pp. 56–70. Springer, 2004.
10. J. Hintikka. Distributive normal forms in the calculus of predicates. *Acta Philosophica Fennica*, 6, 1953.
11. Ł. Kaiser. Synthesis for structure rewriting systems. In *Proc. of MFCS'09*, vol. 5734 of *LNCS*, pp. 415–427. Springer, 2009.
12. R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.
13. O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proc. of CAV'00*, vol. 1855 of *LNCS*, pp. 36–52. Springer, 2000.
14. E. W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. of STOC'81*, pp. 238–246. ACM, 1981.
15. S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proc. of ICALP'11 (2)*, vol. 6756 of *LNCS*, pp. 162–173. Springer, 2011.
16. O. Serre. Note on winning positions on pushdown games with ω-regular conditions. *Inf. Process. Lett.*, 85(6):285–291, 2003.
17. O. Serre. *Contribution à l'étude des jeux sur des graphes de processus à pile*. PhD thesis, Université Paris 7, 2004.
18. S. Shelah. The monadic theory of order. *Ann. Math.*, 102:379–419, 1975.
19. I. Walukiewicz. Pushdown processes: Games and model checking. In *Proc. of CAV '96*, vol. 1102 of *LNCS*, pp. 62–74. Springer, 1996.

A Relational Structures and MSO

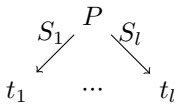
A relational signature $\tau = \{(R_1, r_1), \cdots, (R_n, r_n)\}$ is a set of relation symbols R_i, each with an arity $r_i \in \mathbb{N}, r_i > 0$. Let us fix a finite relational signature τ.

▸ **Definition 21.** A (relational) *τ-structure* $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}})$ consists of a set A, called the domain or universe of $\mathfrak{A}$, and relations $R_i^{\mathfrak{A}} \subseteq A^{r_i}$ corresponding to the symbols in τ.

We write $\mathsf{FinStr}(\tau)$ to denote the set of all finite relational τ-structures, i.e. τ-structures with a finite domain A. Relational structures can be used to represent words, trees and other objects, as shown in the following examples.

Finite Words. A finite word of length n over a finite alphabet $\Gamma = \{P_0, \cdots, P_m\}$ is represented by the relational structure $\mathfrak{W} = (\{0, \cdots, n-1\}, S, P_0, \cdots, P_m\})$, where $P_0^{\mathfrak{W}}, \cdots, P_m^{\mathfrak{W}}$ form a partition of $\{0, \cdots, n-1\}$ and S is the successor relation $S := \{(i, i+1) : i \in \{0, \cdots, n-2\}\}$.

Finite Trees. A finite tree t with maximal branching degree k over an alphabet $\Gamma = \{P_0, \cdots, P_m\}$ is a structure $\mathfrak{T} = (T, S_1, \cdots, S_k, P_0, \cdots, P_m)$ such that $P_0^{\mathfrak{T}}, \cdots, P_m^{\mathfrak{T}}$ form a partition of T, the S_i are pairwise disjoint, the directed graph $G = (T, S_1 \cup \cdots \cup S_k)$ is cycle-free, there exists a unique vertex r such that $(s, r) \notin S_j$ for all $j < k, s \in T$, and for all elements s, if $(s, s') \in S_j$ then there exist elements $s_1, \cdots, s_{j-1}$ with $(s, s_i) \in S_i$. We denote by $P - t_1 \cdots t_l$ the tree with root r labeled by P and subtrees t_i. The root of each t_i is in the relation S_i with r, as depicted below.



Petri Net Markings. A Petri net marking is a function $m \colon L \to \mathbb{N}$, assigning to every place $l \in L$ the number of tokens in this place. We encode a marking as a relational structure over the signature L, i.e. using a monadic predicate P_l for every place l: $\mathfrak{M} = (M, \{P_l\}_{l \in L})$, where M is the set of all tokens and P_l form a partition, such that $m(l) = |\{t \in M \mid t \in P_l^{\mathfrak{M}}\}|$. We express properties of relational structures in first-order and monadic second-order logic.

▸ **Definition 22.** Formulas of *monadic second-order logic*, MSO, are defined inductively by the following grammar, where lower-case letters x correspond to first-order variables, representing elements in the structure, and upper-case letters X correspond to second-order variables, representing sets of elements.

$$\varphi ::= x = x \mid x \in X \mid R_i(x_1, \cdots, x_{r_i}) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \forall x\varphi \mid \exists X\varphi \mid \forall X\varphi$$

Whether a formula $\varphi(\overline{x}, \overline{X})$ holds in a structure $\mathfrak{A}$, given an assignment $\theta : \overline{x} \to A$ for first-order variables and an assignment $\Theta : \overline{X} \to \mathcal{P}(A)$ for second-order ones, denoted $\mathfrak{A}, \theta, \Theta \vDash \varphi$, is defined inductively. In the inductive definition of the semantics below we write $\theta[x \leftarrow a]$ (or analogous for Θ) for the assignment $\theta' : \overline{x} \cup \{x\} \to A$ that maps x to a and every other variable x' to $\theta(x')$.

- $\mathfrak{A}, \theta, \Theta \vDash x_1 = x_2$ if $\theta(x_1) = \theta(x_2)$
- $\mathfrak{A}, \theta, \Theta \vDash R_i(x_1, \ldots, x_{r_i})$ if $(\theta(x_1), \ldots, \theta(x_{r_i})) \in R_i^{\mathfrak{A}}$
- $\mathfrak{A}, \theta, \Theta \vDash x \in X$ if $\theta(x) \in \Theta(X)$
- $\mathfrak{A}, \theta, \Theta \vDash \neg\varphi$ if it is not the case that $\mathfrak{A}, \theta, \Theta \vDash \varphi$
- $\mathfrak{A}, \theta, \Theta \vDash \varphi \wedge \psi$ if $\mathfrak{A}, \theta, \Theta \vDash \varphi$ and $\mathfrak{A}, \theta, \Theta \vDash \psi$
- $\mathfrak{A}, \theta, \Theta \vDash \varphi \vee \psi$ if $\mathfrak{A}, \theta, \Theta \vDash \varphi$ or $\mathfrak{A}, \theta, \Theta \vDash \psi$
- $\mathfrak{A}, \theta, \Theta \vDash \exists x\varphi$ if $\mathfrak{A}, \theta[x \leftarrow a], \Theta \vDash \varphi$ for some $a \in A$

- $\mathfrak{A}, \theta, \Theta \vDash \forall x \varphi$ if $\mathfrak{A}, \theta[x \leftarrow a], \Theta \vDash \varphi$ for all $a \in A$
- $\mathfrak{A}, \theta, \Theta \vDash \exists X \varphi$ if $\mathfrak{A}, \theta, \Theta[X \leftarrow B] \vDash \varphi$ for some $B \subseteq A$
- $\mathfrak{A}, \theta, \Theta \vDash \forall X \varphi$ if $\mathfrak{A}, \theta, \Theta[X \leftarrow B] \vDash \varphi$ for all $B \subseteq A$

If we forbid the use of second-order variables and quantifiers, we obtain the logic FO. We say that a variable in φ is *bound* if it appears in the scope of a quantifier and a variable which appears unbound is *free*. When writing $\varphi(x_1, \dots, x_n)$ (or short $\varphi(\overline{x})$) we mean that the free variables of φ are exactly $\{x_1, \dots, x_n\}$. *Sentences* are formulas without free variables.

MSO on finite words captures regular languages: the set of word models of an MSO sentence forms a regular language, and for every regular language, such a sentence exists [21, 23, 20]. In a similar way, MSO on finite trees captures regular tree-languages [22]. On the representation of Petri net markings given above, one can express in MSO conditions of the form "at places $S \subseteq P$, there are at least/at most/exactly k tokens", or, equivalently, Boolean combinations of constraints of the form $\sum_{p \in S} m(p) \vartriangle k$ for $\vartriangle \in \{\leq, \geq, =\}$), where $S \subseteq P$ is a set of places and k is a constant. (This expressivity result can be obtained using the Ehrenfeucht-Fraïssé method, as there are only monadic predicates with disjoint interpretations.) Note that this is as expressive as Boolean combinations of statements comparing the number of tokens in a place with a constant.

B Proof of Theorem 9

▸ **Theorem 9.** *Let $t = Q - t_1 \cdots t_l$ be a tree and let $\#_{\overline{x}} \varphi(\overline{x})$ be a counting term. Then, for every partition $[\overline{x}]_l = (\overline{x}_0, \cdots, \overline{x}_l)$ of $\overline{x}$, one can compute a finite set $\Psi_{[\overline{x}]_l}$ of $(l+1)$-tuples of Hintikka formulas from $H_{qr(\varphi), \leq |\overline{x}|}$ such that*

$$[\![\#_{\overline{x}} \varphi(\overline{x})]\!]^t = \sum_{[\overline{x}]_l \in \{[\overline{x}]_l\}} \sum_{\overline{\tau} \in \Psi_{[\overline{x}]}} [\![\#_{\overline{x}_0} \tau_0(\overline{x}_0)]\!]^{t_Q} \cdot [\![\#_{\overline{x}_1} \tau_1(\overline{x}_1)]\!]^{t_1} \cdot \ldots \cdot [\![\#_{\overline{x}_l} \tau_l(\overline{x}_l)]\!]^{t_l}.$$

Proof. Let Φ be the set from Theorem 8. By Lemma 7, every $\varphi_i(\overline{x}_i)$ is effectively equivalent to a disjunction of Hintikka formulas. As noted before, the outermost existential condition in Theorem 8 is semantically a disjunction over tuples in Φ. Thus, when we replace every φ_i by its Hintikka disjunction, these disjunctions can be multiplied out and yield a set of Hintikka formulas with the same property:

$$\Psi = \big\{(\tau_0(\overline{x}_0), \cdots, \tau_l(\overline{x}_l)) \mid \tau_i \text{ is a Hintikka formula}, (\overline{x}_0, \cdots, \overline{x}_l) \in \{[\overline{x}]_l\}\big\}.$$

To obtain $\Psi_{[\overline{x}]_l}$ for a partition $[\overline{x}]_l$, we take the subset of Ψ where $(\overline{x}_0, \cdots, \overline{x}_l) = [\overline{x}]_l$.

Observe first that, for every tree t and every partition $[\overline{x}]_l$ such that $t \vDash \varphi(\overline{a})$ and $\overline{a}$ is distributed among the subtrees according to $[\overline{x}]_l$, there exists exactly one $\overline{\tau} \in \Psi_{[\overline{x}]_l}$ satisfied by the respective subtrees.

By the above observation, for every valid assignment $\varepsilon : \overline{x} \to t$ of $\overline{x}$ such that $t \vDash \varphi(\varepsilon(\overline{x}))$, there exists a unique tuple in Ψ which is satisfied by t and this assignment. Furthermore, for a single tuple $(\tau_0, \cdots, \tau_l)$ in Ψ and two evaluations $\varepsilon_1 : \overline{x} \mapsto \overline{a}$ and $\varepsilon_2 : \overline{x} \mapsto \overline{a}'$ such that $t_i \vDash \tau_i(\overline{a}_i)$ and $t_i \vDash \tau_i(\overline{a}'_i)$ for all i (i.e. ε_1 and ε_2 are valid assignments for $\overline{\tau}$), for every i, $\varepsilon_1 \restriction (\overline{x} \setminus \overline{x}_i) \cup \varepsilon_2 \restriction (\overline{x}_i)$ is again a valid assignment. Thus, for a single tuple $\overline{\tau}$, the number of assignments of $\overline{x}$ for which the tuple is satisfied is the product of the numbers of assignments for each $\overline{x}_i$, symbolically written:

$$\begin{aligned} &\#_{\overline{x}}\big(t_Q \vDash \tau_0(\overline{x}_0) \text{ and } t_1 \vDash \tau_1(\overline{x}_1) \text{ and } \cdots \text{ and } t_l \vDash \tau_l(\overline{x}_l)\big) \\ &= \#_{\overline{x}_0}\big(t_Q \vDash \tau_0(\overline{x}_0)\big) \cdot \#_{\overline{x}_1}\big(t_1 \vDash \tau_1(\overline{x}_1)\big) \cdots \#_{\overline{x}_l}\big(t_l \vDash \tau_l(\overline{x}_l)\big). \end{aligned}$$

Since, for every valid assignment of $\overline{x}$, the actual variables can be separated into groups according to which subtree (or the root) they are assigned in, and different $\overline{\tau} \neq \overline{\tau}'$ are mutually contradicting, the lemma follows. ◂

C Model-Checking Games

▸ **Theorem 5** (c.f. [7]). *For every $\psi \in$ Qμ[#MSO] and every STS $\mathfrak{S}$ one can construct the quantitative parity game $\mathsf{MC}(\mathfrak{S}, \psi) = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ which is a* model-checking game *for ψ and $\mathfrak{S}$, i.e. it satisfies the following properties:*

(1) $V = (S \times \mathsf{Sub}(\psi)) \cup \{(\infty), (-\infty)\}$, where $\mathsf{Sub}(\psi)$ is the set of subformulas of ψ.
(2) In terminal positions (s, ψ) the formula ψ has the form $\#_{\overline{x}}\varphi(\overline{x})$ or $\neg\#_{\overline{x}}\varphi(\overline{x})$.
(3) If $((s, \psi), (s', \psi')) \in E$ then either $s = s'$ or $(s, s') \in \Delta$.
(4) $\Omega(s, \psi)$ depends only on ψ, not on s.
(5) $\mathsf{val}\mathcal{G}(s, \psi) = [\![\psi]\!]^{\mathfrak{S}}(s)$.

Proof. The game $\mathsf{MC}(\mathfrak{S}, \psi) = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ is defined as follows.

- $V = (S \times \mathsf{Sub}(\psi)) \cup \{(-\infty), (\infty)\}$, where $\mathsf{Sub}(\psi)$ is the set of subformulas of ψ.
- $V_{\max} = \{(s, \vartheta) \in V : \vartheta = \vartheta_1 \vee \vartheta_2 \text{ or } \vartheta = \Diamond\vartheta'\}$
- $V_{\min} = V \setminus V_{\max}$
- Terminal vertices:
  $T = \{(-\infty), (\infty)\} \cup \{(s, \#_{\overline{x}}\varphi(\overline{x}))\} \cup \{(s, \neg\#_{\overline{x}}\varphi(\overline{x}))\}$
- $E = \{((s, \vartheta_1 \oplus \vartheta_2), (s, \vartheta_i)) \mid \oplus \in \{\vee, \wedge\}\}$
  $\cup\{((s, \dagger\vartheta), (s', \vartheta)) \mid \dagger \in \{\Box_r, \Diamond_r\}, (s, r, s') \in \Delta\}$
  $\cup\{((s, \Diamond\vartheta), (-\infty)) \mid \Delta(s) = \varnothing\}$
  $\cup\{((s, \Box\vartheta), (\infty)) \mid \Delta(s) = \varnothing\}$
  $\cup\{((s, X), (s, \mu X.\vartheta)) \mid X \text{ is a lfp-variable}\}$
  $\cup\{((s, X), (s, \nu X.\vartheta)) \mid X \text{ is a gfp-variable}\}$
  $\cup\{((s, \beta X.\vartheta), (s, \vartheta)) : \beta \in \{\mu, \nu\}\}$

Positions. Let $\mathsf{Sub}(\psi)$ be the set of subformulas of ψ. The set of positions is defined as $V = (S \times \mathsf{Sub}(\psi)) \cup \{(-\infty), (\infty)\}$. Positions of Maximizer are those where the current subformula is of the form $\varphi_1 \vee \varphi_2$ or $\Diamond\varphi$, while all other positions belong to Minimizer.

Moves. The positions $(-\infty)$ and (∞) are terminal, and so are vertices with formulas $\#_{\overline{x}}\varphi(\overline{x})$ or $\neg\#_{\overline{x}}\varphi(\overline{x})$. At positions $(t, \varphi_1 \vee \varphi_2)$ and $(t, \varphi_1 \wedge \varphi_2)$ one can move to (t, φ_1) or (t, φ_2). At positions $(t, \mu X.\varphi)$ and $(t, \nu X.\varphi)$ the only successor is (t, φ). At positions $(t, \Diamond_r\varphi)$ and $(t, \Box_r\varphi)$ two cases are distinguished. If, in $\mathfrak{S}$, the vertex t does not have a r successor, then the only successor in the game is $(-\infty)$ – for $\Diamond_r\varphi$ – or (∞) – for $\Box_r\varphi$. Otherwise, there are edges to every (t', φ) such that $(t, r, t') \in \Delta$.

Payoffs. At positions $(t, \#_{\overline{x}}\varphi(\overline{x}))$ the payoff is $\lambda(t, \#_{\overline{x}}\varphi(\overline{x})) = [\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{\mathfrak{m}(t)}$. At positions $(t, \neg\#_{\overline{x}}\varphi(\overline{x}))$, it is $-[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{\mathfrak{m}(t)}$. At positions $(-\infty)$ and (∞) it is $-\infty$ or ∞, respectively.

Priorities. The priority function Ω is defined as in the model-checking game for the classical μ-calculus, i.e. even priorities at fixed point variables of greatest fixed points, odd ones at those of least fixed points and such that variables with a higher alternation level get higher priorities. All other positions are assigned the maximal priority.

Correctness. It was shown in [7] (even for systems with discounts) that the game $\mathsf{MC}(\mathfrak{S}, \psi)$ is a model-checking game, i.e. that for all $s \in S$, $\mathsf{val}\,\mathsf{MC}(s, \psi) = [\![\psi]\!]^{\mathfrak{S}}(s)$. ◂

D Pushdown Processes and Games

▸ **Definition 23** (c.f. Definition 1 in [16]). A *pushdown process* over a stack alphabet Γ with a bottom symbol $\perp \notin \Gamma$ is a tuple $(Q, \hookrightarrow)$ where Q is a finite set of states and $\hookrightarrow$ are transition rules of the form:

- *Push:* $(p,a) \hookrightarrow (q,ab)$, where $p,q \in Q$, $a \in \Gamma \cup \{\perp\}$, $b \in \Gamma$,
- *Pop:* $(p,a) \hookrightarrow (q,\varepsilon)$, where $p,q \in Q$ and $a \in \Gamma$,
- *Epsilon:* $(p,a) \hookrightarrow (q,a)$, for $p,q \in Q$ and $a \in \Gamma \cup \{\perp\}$.

A configuration of $\mathcal{A}$ is a pair (q,u) with $q \in Q$ and $u \in \Gamma^*$. The one-step transition relation $E_{\hookrightarrow}$ is defined naturally as

$$\begin{aligned}
&\big\{\big((p,ua),(q,uab)\big) \mid p,q \in Q,\, u \in \Gamma^*, a \in \Gamma,\, (p,a) \hookrightarrow (q,ab)\big\} \\
\cup&\big\{\big((p,\varepsilon),(q,b)\big) \mid p,q \in Q,\, (p,\perp) \hookrightarrow (q,\perp b)\big\} \\
\cup&\big\{\big((p,ua),(q,u)\big) \mid p,q \in Q,\, u \in \Gamma^*, a \in \Gamma,\, (p,a) \hookrightarrow (q,\varepsilon)\big\} \\
\cup&\big\{\big((p,ua),(q,ua)\big) \mid p,q \in Q,\, u \in \Gamma^*, a \in \Gamma,\, (p,a) \hookrightarrow (q,a)\big\}
\end{aligned}$$

A pushdown system $\mathcal{A}$ is *pop-free* if $\hookrightarrow$ contains no pop-rules. In such case, if $\big((s,q),(s',q')\big) \in E_{\hookrightarrow}$ then s' is at least as long as s.

A *quantitative parity game* $\mathcal{G} = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ is a pushdown game over a pushdown process $\mathcal{A} = (Q, \hookrightarrow)$ if there exists a partition of Q into $Q_{\max}$ and $Q_{\min}$ such that $V = Q \times \Gamma^*$, $V_{\max} = Q_{\max} \times \Gamma^*$ and $V_{\min} = Q_{\min} \times \Gamma^*$. Moreover, $E = E_{\hookrightarrow}$ and $\Omega(q,u) = \Omega(q,v)$ for all $u,v \in \Gamma^*$ and each $q \in Q$, thus one can simply write $\Omega(q)$.

D.1 Strategy Construction for pop-Elimination

Here we present the proof of correctness of the construction given in Section 5. Recall that σ is a strategy of one of the players in $\mathcal{G}$. We define the corresponding truthful strategy σ' in $\mathcal{G}'$ by induction on the length of play prefixes. Also, to each play prefix π' consistent with σ' we assign a corresponding play prefix π in $\mathcal{G}$ consistent with σ. Intuitively, when the player is supposed to make a claim in $\mathcal{G}'$, he considers all plays extending π in $\mathcal{G}$ and consistent with σ and, in σ', chooses a claim with the colors that really occur. When the opponent makes a claim in $\mathcal{G}'$ and indeed there is an extension consistent with σ which returns to the same stack in the claimed state and color, then σ' accepts this claim, and the corresponding play in $\mathcal{G}$ is prolonged to the point in which the stack pops back to the claimed state. If no play is consistent with the claim of the opponent, i.e. it is false, then the push-move is made.

We present formally the construction for Maximizer, the case for Minimizer is analogous. For a thorough explanation with examples of a similar construction see Chapter 5 of [17]. We show that each play π' consistent with the constructed strategy σ' has at least the same payoff as the corresponding play in $\mathcal{G}$, which is consistent with σ. Let π' be a play prefix in $\mathcal{G}'$ ending in (q,x,s) and let π be the corresponding prefix in $\mathcal{G}$. The corresponding prefixes in $\mathcal{G}$ will always end in the same Q-state and with the same stack, thus π ends in (q,s). Moreover, if $x = (m,C,p)$ then we will inductively guarantee that the claim is *positive for Maximizer*, i.e. Maximizer wins if a pop occurs, meaning that $\min(m,\Omega(q)) \in C(q)$ if $p = \max$ and $\min(m,\Omega(q)) \notin C(q)$ if $p = \min$. We consider four cases.

Case 1: $x \in \{\perp\} \cup Q'_0$ and Maximizer moves in (q,s).
In this case Maximizer also moves in (q,x,s). If $\sigma(\pi)$ prescribes an epsilon-move or a pop-move, then we set $\sigma'(\pi')$ to the same move. In the case of a pop-move, Maximizer wins in $\mathcal{G}'$ as the claim in x is positive for him, and for an epsilon-move to (q',x',s) in $\mathcal{G}'$ we define the play corresponding to $\pi' \cdot (q',x',s)$ as $\pi \cdot (q',s)$. Consider the case when

$\sigma(\pi)$ makes a push-move, pushing a and moving to a state q'. Let P be the set of all pairs (q'', m) such that $\pi \cdot (q', sa) \cdots (q'', s)$ is a play prolonging π, consistent with σ, returning to stack s, and m is the minimal color on $(q', sa) \cdots (q'', s)$. We define the claim C' by $C'(q) = \{c \mid (q, c) \in P\}$. Observe that all pop-moves in plays consistent with σ and returning to stack s will be consistent with the claim C'. Thus C' is positive for Maximizer and we can set $\sigma'(\pi') = (q', a, C', \max, x, s)$.

Case 2: $x \in \{\bot\} \cup Q'_0$ and Minimizer moves in (q, s).
In this case Minimizer also moves in (q, x, s). If he chooses a pop-move, then he loses in the resulting sink position as the claim was assumed to be positive for Maximizer. If he chooses an epsilon-move to (q', x', s), then the same move can be made in $\mathcal{G}$ and we define $\pi \cdot (q', s)$ as the play corresponding to $\pi' \cdot (q', x', s)$. If he chooses a move to (q', x', s), then we set π as the play corresponding to the prolonged $\pi' \cdot (q', x', s)$; see Case 4 for how Maximizer responds to this claim.

Case 3: $x \in Q'_1$ and Maximizer moves in (q, s).
In this case Minimizer moves in (q, x, s). If he moves to (q', x', sa), making the push-move, then we set $\pi \cdot (q, sa)$ as the play corresponding to $\pi' \cdot (q', x', sa)$. If he uses a claim (made in Case 1) and moves to (q', x', s), then – by the construction in Case 1 – there is a play $\pi_1 = \pi \cdot (q, sa) \cdots (q', s)$ in $\mathcal{G}$ consistent with σ. We set π_1 as the play corresponding to $\pi' \cdot (q', x', s)$. Note that it is not only consistent with σ, but the priority $\Omega(q', x', s)$ is set to m, i.e. to the minimal priority seen on $(q, sa) \cdots (q', s)$, the part by which π was extended.

Case 4: $x \in Q'_1$ and Minimizer moves in (q, s).
Since $x \in Q'_1$, it is Maximizer who moves in (q, x, s) and must respond to the claim C made by Minimizer in the previous move. As in Case 1, consider all plays $\pi_1 = \pi \cdot (q, sa) \cdots (q', s)$ prolonging π, consistent with σ, and returning to stack s, and let m be the minimal color on $(q, sa) \cdots (q', s)$. If there exists such a play π_1 for which $m \in C(q')$, and $x = (m_x, C_0, p_0)$, then set $\sigma'(\pi') = (q', \min(m_x, m), C_0, p_0, s)$ and let π_1 be the play corresponding to $\pi' \cdot \sigma(\pi')$. In the other case, set $\sigma(\pi') = (q, \Omega(q), C, \min, sa)$ and let $\pi \cdot (q, sa)$ be the corresponding play. Since the claim C is satisfied in *no* play consistent with σ, it is again positive for Maximizer.

Consider a play π' in $\mathcal{G}$ consistent with the strategy σ' and the corresponding play π in $\mathcal{G}$, as constructed above. If π' ends in a terminal position, the payoff in π' is the same as in π. If π' ends in a sink-position, Maximizer wins and the payoff is $+\infty$. Finally if π' is infinite then the minimal color seen infinitely often in π' is the same as in π, as we always collected the minimal color when prolonging π. Therefore σ' guarantees a payoff at least as big as the one guaranteed by σ. Since the same construction is valid for Minimizer as well and both $\mathcal{G}$ and $\mathcal{G}'$ are determined, we get that the values of $\mathcal{G}$ and of $\mathcal{G}'$ are the same, completing the proof of Theorem 6.

E Proof of Lemma 20

▸ **Lemma 20.** *There exists a computable function* $\mathrm{pre} \colon \mathcal{L} \times \mathcal{R} \to \mathcal{L}$ *such that for all sequences* $r_1, \ldots, r_n$ *of rules,* pre *together with* L_F *induces a compatible sequence, i.e.* $L_0, \cdots, L_n$ *with* $L_n = L_F$ *and* $L_{i-1} = \mathrm{pre}(L_i, r_i)$ *is compatible.*

Proof. For a type function L and a rule $r = P \leftarrow Q - Q_1 \cdots Q_l$ we define $\mathrm{pre}(L, r)$ as follows:

$$\mathrm{pre}(L, r) := \begin{cases} L(R) & \text{if } R \neq P, \\ \bigoplus(\mathrm{tp}^m(t_Q), L(Q_1), \cdots, L(Q_l)) & \text{if } R = P, \end{cases}$$

where the sum $\bigoplus(\mathrm{tp}^m(t_Q), L(Q_1), \cdots, L(Q_l))$ is the type of the tree $Q - t_1 \cdots t_l$ with $\mathrm{tp}^m(t_i) = L(Q_i)$, which is computable by Ehrenfeucht-Fraïssé methods.

Let $L_0, \cdots, L_n$ be the sequence with $L_n = L_F$ and $L_{i-1} = \mathrm{pre}(L_i, r_i)$. We prove compatibility of the sequence via backwards induction. For $i = n$, we have $L_n(P) = L_F(P) = \mathrm{tp}^m(t_P)$. Let thus L_i be compatible, let $r_i = P \leftarrow Q - Q_1 \cdots Q_l$ and let $L_{i-1} = \mathrm{pre}(L_i, r_i)$. By definition, $r_n \cdots r_{i+1}(t_R) = r_n \cdots r_i(t_R)$ and $L_{i-1}(R) = L_i(R)$ for all $R \neq P$. For P, the subtree of P-leaves is build according to r_i, thus $r_n \cdots r_i(t_P) = Q - r_n \cdots r_{i+1}(t_{Q_1}) \cdots r_n \cdots r_{i+1}(t_{Q_l})$. By induction hypothesis, $L_i(Q_j) = \mathrm{tp}^m(r_n \cdots r_{i+1}(t_{Q_j}))$, and because $L_{i-1}(P)$ equals the sum of $\mathrm{tp}^m(t_Q)$ and the $L_i(Q_j)$, $L_{i-1}(P)$ is indeed the type of $r_n \cdots r_i(t_P)$. ◂

Appendix References

20 J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 6:66–92, 1960.

21 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

22 J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

23 B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Sibirian Mathematical Journal*, 13:103–131, 1962. (in Russian).