Average Stack Cost of Büchi Pushdown Automata*†

Jakub Michaliszyn[1] **and Jan Otop**[2]

1 **University of Wrocław, Poland**
2 **University of Wrocław, Poland**

Abstract

We study the average stack cost of Büchi pushdown automata (Büchi PDA). We associate a non-negative price with each stack symbol and define the cost of a stack as the sum of costs of all its elements. We introduce and study the average stack cost problem (ASC), which asks whether there exists an accepting run of a given Büchi PDA such that the long-run average of stack costs is below some given threshold. The ASC problem generalizes mean-payoff objective and can be used to express quantitative properties of pushdown systems. In particular, we can compute the average response time using the ASC problem. We show that the ASC problem can be solved in polynomial time.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases pushdown automata, average stack cost, weighted pushdown systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.42

1 Introduction

Weighted pushdown systems (WPSs) combine finite-control, unbounded stack and weights on transitions. Weights are aggregated using semiring operations [10] or the long-run average [5]. These features make them a powerful formalism capable of expressing interesting program properties [10, 9]. Still, WPSs considered in the literature fall short of expressing the following basic quantitative specification.

Consider the following *client-server scenario*, consisting of two agents, a server and a client. The client sends requests (r), which are granted (g) by the server. Each grant satisfies all pending requests. All other events are abstracted to a null instruction (#). We are interested in checking properties of such systems over infinite runs. We are only interested in sequences with infinitely many requests and grants. The average workload property (AW) for client-server scenario, defined as the long-run average of the number of pending requests over all positions, was studied in [4].

WPSs can model the client-server scenario, but they cannot express AW for two reasons. First, WPSs considered in the literature [5] have no Büchi acceptance condition, and hence we cannot specify traces with infinitely many requests and grants. Second, weights in WPSs are bounded, and hence the long-run average is bounded by the maximal weight, whereas AW is unbounded.

In this paper we study WPSs with Büchi acceptance conditions (known as Büchi pushdown automata) and unbounded weights depending on the stack content, called stack costs. More

* This work was supported by the National Science Centre (NCN), Poland under grant 2014/15/D/ST6/04543.
† A full version of the paper is available at [7], `https://arxiv.org/abs/1710.04490`.

precisely, we define the stack cost as a non-negative linear combination of the number of occurrences of every stack letter, i.e., given stack pricing that assigns a non-negative cost with stack symbols, the stack cost is the sum of prices of its elements. We investigate the *average stack cost* (ASC) during infinite computations of an Büchi pushdown automaton. For a finite computation, the average stack cost is simply the sum of the stack costs in every position divided by the number of positions. It is extended to infinite computations by taking the limit of the average stack costs of all the (finite) prefixes of this infinite computation. As the limit may be undefined (when the sequence of prefixes diverge), we consider two values, the limit inferior and limit superior over all prefixes.

We argue that with the ASC problem we can express interesting system properties. In particular, we can express AW from the client-server scenario. Moreover, we can express a variant of AW where each grant satisfies only one request. This variant of AW cannot be specified with models from [4]. We can also use ASC to compute the average response time property [3], which asks for the average number of steps between a request and the corresponding grant. In this variant of the average response time, we can assume that each grant satisfies one request, which has not been possible in previous formalisms [3].

Contributions. The main results presented in this paper are as follows.

- The average stack cost problem can be solved in polynomial time assuming unary encoding of stack pricing.
- One-player games on WPSs with the conjunctions of mean-payoff and Büchi objectives can be solved in polynomial time, even assuming binary encoding of weights.
- The average response time property over WPSs in a variant of the client-server scenario where each grant satisfies only one request can be computed in polynomial time.

Overview. We start with basic definitions in Section 2. Next, in Section 3 we discuss convergence of the partial averages of the stack costs. In Section 4, we show that to solve ASC we can bound the stack costs along the whole run. This allows us to reduce ASC to the average letter cost problem, which is equivalent to one-player games on WPSs with the conjunction of mean-payoff and Büchi objectives. We chose letter-based formalization rather than WPSs with weights on transitions, as it allows us to use classical language-theoretic results on ω-PDA. We apply these results in Section 5 to show that the average letter cost problem can be solved in polynomial time. Finally, we discuss the connection between ASC and the average response time property (Section 6).

The detailed proofs are presented in the full version of this paper [7].

Related work. WPSs with weights from a *bounded idempotent semiring* and their applications have been studied in [10, 9]. In bounded idempotent semirings there are no infinite descending chains, e.g., the natural numbers, in contrast to the integers. The results from [9] have been generalized to WPSs over indexed domains [8], which still do not capture the integers. WPSs with integer weights aggregated with the long-run average operation (a.k.a. mean-payoff objective) have been studied in [5]. It has been shown that one-player games on WPSs with mean-payoff objective can be solved in polynomial time.

The average stack cost is closely related to the average energy objective studied over finite graphs [1]. In contrast to stack cost, energy levels are not observable, i.e., transitions do not depend on energy levels. One player energy games are decidable in polynomial time. As we can express energy levels using stack costs, the results of this paper can be considered as a generalization of the average-energy objective in the one-player case. However, two-player energy games are decidable in NP ∩ coNP [1], while even mean-payoff games on WPSs are undecidable [5]. Since the average stack cost generalizes the mean-payoff objectives, two-player average-stack-cost games are undecidable.

2 Preliminaries

Words and automata. Given a finite alphabet Σ of letters, a *word* w is a finite or infinite sequence of letters. We denote the set of all finite words over Σ by Σ^*, and the set of all infinite words over Σ by Σ^ω. We use ϵ to denote the empty word.

For a word w, we define $w[i]$ as the i-th letter of w, and we define $w[i,j]$ as the subword $w[i]w[i+1]\dots w[j]$ of w. We allow $j = \infty$ in $w[i,j]$. By $|w|$ we denote the length of w. We use the same notation for sequences that start from 0.

A *(non-deterministic) pushdown automaton* (PDA) is a tuple $(\Sigma, \Gamma, Q, Q_0, Q_F, \delta)$, where Σ is the input alphabet, Γ is a finite stack alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $Q_F \subseteq Q$ is a set of accepting states, and $\delta \subseteq Q \times \Sigma \times (\Gamma \cup \{\bot\}) \times Q \times \Gamma^*$ is a finite transition relation. We define Büchi-PDA (called ω-PDA for short) in the same way; these automata differ in semantics. The size of an automaton $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, Q_F, \delta)$, denoted by $|\mathcal{A}|$, is $|Q| + |\delta|$.

Assume a PDA (resp., ω-PDA) $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, Q_F, \delta)$. A *configuration* of $\mathcal{A}$ is a tuple $(q, a, u) \in Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\bot\})^*$, where $\bot$ occurs only once in u; it occurs as its first symbol. A *run* π of $\mathcal{A}$ is a sequence of configurations such that $\pi[0] = (q_0, \epsilon, \bot)$ for some $q_0 \in Q_0$ and for every $i < |\pi|$, if $\pi[i, i+1] = (q, a, u)(q', a', u')$, we have $\delta(q, a', x, q', y)$ for some x, y such that either $x = u = \bot$ and $u' = y$ or $x \neq \bot$, $u = u_s x$ for some u_s and $u' = u_s y$. Runs of PDA are finite sequences, while runs of ω-automata are infinite.

A run $\pi = (q^0, a^0, u^0)(q^1, a^1, u^1)\dots$ *gives* the word $a^0 a^1 \dots$. A finite run π of a PDA is *accepting* if the last state in π belongs to Q_F. An infinite run π of an ω-PDA is *accepting* if it visits Q_F infinitely often, i.e., satisfies the Büchi acceptance condition, and gives an infinite word. The *language recognized (or accepted) by the PDA* $\mathcal{A}$ (resp., ω-PDA $\mathcal{A}$), denoted $\mathcal{L}(\mathcal{A})$, is the set of all words given by accepting runs of $\mathcal{A}$.

Weighted pushdown systems. A weighted pushdown system (WPS) $\mathcal{P}$ is pair $(\mathcal{A}, \mathrm{wt})$ such that (1) $\mathcal{A}$ is a PDA (resp., ω-PDA) $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, Q_F, \delta)$, (2) the alphabet Σ is a singleton, (3) all states are accepting, i.e., $Q = Q_F$, and (4) wt is a cost function that maps transitions δ into a cost domain (which is $\mathbb{Z}$ in our case). The alphabet Σ and the set of accepting states are typically omitted.

Context-free grammars (CFG) and their languages. A context-free grammar (CFG) is a tuple $G = (\Sigma, V, S, P)$, where Σ is the alphabet, V is a set of *non-terminals*, $S \in V$ is a *start symbol*, and P is a set of *production rules*. Each production rule p has the following form $v \to u$, where $v \in V$ and $u \in (\Sigma \cup V)^*$. We define *derivation* $\to_G$ as a relation on $(\Sigma \cup V)^* \times (\Sigma \cup V)^*$ as follows: $w \to_G w'$ iff $w = w_1 v w_2$, $w' = w_1 u w_2$, and $v \to u$ is a production from G. We define $\to_G^*$ as the transitive closure of $\to_G$. The *language generated by* G, denoted by $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \to_G^* w\}$ is the set of words that can be derived from the start symbol S. CFGs and PDAs are language-wise polynomial equivalent (i.e., there is a polynomial time procedure that, given a PDA, outputs a CFG of the same language and vice versa) [6].

2.1 Basic problems

Let $\mathcal{A} = (\Sigma, \Gamma, Q, Q_0, Q_F, \delta)$ be an ω-PDA. A *stack pricing* is a function $\mathbf{c} : \Gamma \to \mathbb{N}$ that assigns each stack symbol with a natural number (we assume 0 is natural). We extend $\mathbf{c}$ to configurations (q, a, u) by setting $\mathbf{c}((q, a, u)) = \sum_{i=1}^{|u|} \mathbf{c}(u[i])$, where we assume that $\mathbf{c}(\bot) = 0$.

Given a run π of $\mathcal{A}$, a stack pricing $\mathbf{c}$ and $k > 0$, we define the *average stack cost* of the prefix of π of length k, denoted by $\mathsf{ASC}(\pi, \mathbf{c}, k)$, as $\frac{1}{k} \sum_{i=0}^{k-1} \mathbf{c}(\pi[i])$.

We are interested in establishing the average stack cost for the whole runs, which can be formalized in two ways. The *infimum-average stack cost* of π, denoted by $\mathsf{IASC}(\pi, \mathbf{c})$, and the *supremum-average stack cost* of π, denoted by $\mathsf{SASC}(\pi, \mathbf{c})$, are defined as

$$\mathsf{IASC}(\pi, \mathbf{c}) = \liminf_{k \to \infty} \mathsf{ASC}(\pi, \mathbf{c}, k) \qquad \mathsf{SASC}(\pi, \mathbf{c}) = \limsup_{k \to \infty} \mathsf{ASC}(\pi, \mathbf{c}, k)$$

If $\mathbf{c}$ is known from the contexts, we omit it and write $\mathsf{IASC}(\pi)$ instead of $\mathsf{IASC}(\pi, \mathbf{c})$ and similarly for SASC.

We define two decision questions collectively called the average stack cost problem.

▶ **The `IASC` problem**: given an ω-PDA $\mathcal{A}$, a stack pricing $\mathbf{c}$, $\bowtie \in \{<, \leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists an accepting run π of $\mathcal{A}$ such that $\mathsf{IASC}(\pi, \mathbf{c}) \bowtie \lambda$.

▶ **The `SASC` problem**: given an ω-PDA $\mathcal{A}$, a stack pricing $\mathbf{c}$, $\bowtie \in \{<, \leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists an accepting run π of $\mathcal{A}$ such that $\mathsf{SASC}(\pi, \mathbf{c}) \bowtie \lambda$.

We assume that the numbers in the stack pricing and the threshold are given in unary, i.e., in an instance I of the average stack cost problem, values of $\mathbf{c}$ and λ are polynomially bounded (in the size of the instance).

▶ Remark. Observe that the average stack cost problem generalizes WPSs with mean-payoff objectives. First, WPSs consists of a PDA (resp., ω-PDA) and a cost function from transitions into integers. We can however add a constant C to all weights, which change all mean-payoff values by C. Thus, we can assume that costs are non-negative. Second, we can emulate costs on transitions by extending the stack alphabet with letters corresponding to transitions, and storing the last taken transition at the top of the stack. Hence, allowing, in addition to stack costs, costs on transitions does not change the expressive power or the complexity. For simplicity, we do not consider costs on transitions. Finally, the average stack cost strictly generalizes WPSs with mean-payoff objectives as it can be unbounded whereas the mean-payoff is bounded by the maximal weight of the transition.

▶ **Example 1.** Recall the client-server scenario from the introduction. Assume that the stack alphabet is $\Gamma = \{r\}$ and all requests are pushed on the stack by the client. Then, upon a grant the server empties the stack. Observe that if the cost of a request on the stack is 1, i.e., $\mathbf{c}(r) = 1$, then the average stack cost equals AW.

We can modify this example to model that each grant satisfies a single request. Simply, we require the server to pop only a single request upon a grant. Again, the average stack cost equals AW.

3 Properties of Average Stack Cost

We extend the notion of the average stack cost to PDA, defining $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \inf\{\mathsf{IASC}(\pi, \mathbf{c}) \mid \pi \text{ is an accepting run of } \mathcal{A}\}$ and $\mathsf{SASC}(\mathcal{A}, \mathbf{c}) = \inf\{\mathsf{SASC}(\pi, \mathbf{c}) \mid \pi \text{ is an accepting run of } \mathcal{A}\}$.

We can easily construct a run π and stack pricing $\mathbf{c}$, such that $\mathsf{IASC}(\pi, \mathbf{c}) < \mathsf{SASC}(\pi, \mathbf{c})$. We now show an example proving a stronger claim, stating that even $\mathsf{IASC}(\mathcal{A}, \mathbf{c})$ and $\mathsf{SASC}(\mathcal{A}, \mathbf{c})$ can have different values.

▶ **Example 2.** Consider an automaton $\mathcal{A}$ with three states U, B, A, one alphabet symbol a and two stack symbols α, β, and the stack pricing such that $\mathbf{c}(\alpha) = 0$ and $\mathbf{c}(\beta) = 3$. State A is the only accepting and the only starting state. The transition function is as follows.

$$\delta(A, a, \bot, U, \alpha) \qquad \delta(U, a, \alpha, U, \alpha\alpha) \qquad \delta(U, a, \alpha, B, \beta)$$
$$\delta(B, a, \beta, B, \epsilon) \qquad \delta(B, a, \beta, A, \epsilon) \qquad \delta(B, a, \alpha, B, \beta)$$

Every accepting run of $\mathcal{A}$ starts in the state A, adds some number of symbols α to the stack in state U, and then goes to the state B, where it clears the stack, but to remove a symbol α, it first needs to convert it to (costly) β. Then it reaches A with empty stack and repeats.

Observe that $\mathsf{SASC}(\mathcal{A}, \mathbf{c}) = 1$. To see this, consider an accepting run π. For any position $p > 0$ with an accepting state we have that $\mathsf{ASC}(\pi, \mathbf{c}, p-1) = 1$. To show this, we assign to every β symbol that occur in $\pi[0, p-1]$ three positions: right before it was removed, right before it replaced some α, and right before this α was added. In this way we cover all the positions in $\pi[0, p-1]$, which means that the number of β symbols is three times the number of positions, so $\mathsf{ASC}(\pi, \mathbf{c}, p-1) = 1$.

In contrast, we show that $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = 0$. Let π_i be the sequence of configurations

$$(A, a, \bot)(U, a, \bot\alpha) \dots (U, a, \bot\alpha^i)(B, a, \bot\alpha^{i-1}\beta)(B, a, \bot\alpha^{i-1})(B, a, \bot\alpha^{i-2}\beta) \dots (B, a, \bot\beta)$$

For IASC, consider a sequence a_i defined recursively[1] as $a_1 = 1$, $a_{i+1} = i \cdot \sum_{j=1}^{i} a_j$ and a run $\pi = \pi_{a_1}\pi_{a_2}\pi_{a_3}\dots$. For each i, we have $\sum_{j=0}^{a_1+\dots+a_i} \mathbf{c}(\pi[j]) = 3(a_1 + \dots + a_i)$ (as in the SASC case, one can assign exactly three positions to each β). Therefore, at the position $3a_i + a_{i+1}$ in π, which is in $\pi_{a_{i+1}}$ and it is the first position there with B, the value $\mathsf{ASC}(\pi, \mathbf{c}, 3a_i + a_{i+1})$) can be bounded by $\frac{3(a_1+\dots+a_i)}{3(a_1+\dots+a_i)+a_{i+1}} = \frac{3(a_1+\dots+a_i)}{(3+i)(a_1+\dots+a_i)} = \frac{3}{3+i}$. The sequence $\frac{3}{3+i}$ converges to 0, and since we only have non-negative costs, $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = 0$.

The above example uses both non-accepting states (to ensure that the stack is emptied infinitely often) and zero costs. Both are needed; we show a no-free-lunch theorem, saying that we can only have two out of three things: (1) ω-PDA with non-accepting states, (2) stack symbols with cost 0, or (3) a guarantee that IASC and SASC coincide.

▶ **Theorem 3.** *Let $\mathcal{A}$ be an ω-PDA and $\mathbf{c}$ be a stack pricing $\mathbf{c}$. If $\mathcal{A}$ has only accepting states or $\mathbf{c}$ returns only positive values, then* $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$.

Proof sketch. In the only-accepting-runs case, the theorem follows from a reduction to one-player games on WPSs with mean-payoff objectives [5]. Mean-payoff objectives are considered in two variants: mean-payoff infimum corresponding to limit infimum of partial averages and mean-payoff supremum corresponding to limit supremum of parial averages. However, it is shown in [5] that winning against one objective (say, the mean-payoff infimum objective) is equivalent to winning against the other (the mean-payoff supremum objective). From that and our reduction, we conclude that $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$. In the only-positive-values case, we prove that it is enough to consider runs with bounded size of the stack; then, we reduce the average stack cost to the regular language case, in which the results on weighted automata [2] and simple arguments show that the infimum over all runs of a weighted automaton with accepting states is realized by a run, in which partial averages converge. We conclude that $\mathsf{IASC}(\mathcal{A}, \mathbf{c}) = \mathsf{SASC}(\mathcal{A}, \mathbf{c})$. ◀

[1] It can be defined explicitly as $a_i = ii!$, but the recursive definition is more convenient for us.

We conclude with a realisability theorem, stating that if $\mathsf{IASC}(\mathcal{A},\mathbf{c})$ and $\mathsf{SASC}(\mathcal{A},\mathbf{c})$ coincide, then there is a single run that witnesses both.

▶ **Theorem 4.** *For an ω-PDA $\mathcal{A}$ and a stack pricing $\boldsymbol{c}$ we have $\mathsf{IASC}(\mathcal{A},\boldsymbol{c}) = \mathsf{SASC}(\mathcal{A},\boldsymbol{c})$ iff there is a run π such that $\mathsf{IASC}(\pi,\boldsymbol{c}) = \mathsf{SASC}(\pi,\boldsymbol{c}) = \mathsf{IASC}(\mathcal{A},\boldsymbol{c})$.*

Proof sketch. We prove that $\mathsf{SASC}(\mathcal{A},\mathbf{c})$ is always realized, i.e., for every PDA $\mathcal{A}$ there exists π such that $\mathsf{SASC}(\pi,\mathbf{c}) = \mathsf{SASC}(\mathcal{A},\mathbf{c})$. This immediately implies the theorem. ◀

4 From Average Stack Cost to Average Letter Cost

The *average letter cost problem* takes an ω-PDA $\mathcal{A}$ and a cost function defined on letters, and asks whether there is a word in the language of $\mathcal{A}$ whose long-run average of costs of letters is below a given threshold. This section is devoted to a polynomial time reduction from the average stack cost problem to the average letter cost problem.

The reduction consists of two steps. First, we show that in the average stack cost problem, we can impose a bound B on the stack cost (which depends on the ω-PDA and the threshold). Next, we take the ω-PDA $\mathcal{A}$ from the average stack cost problem and define an ω-PDA $\mathcal{A}^M$, which recognizes words encoding the runs of $\mathcal{A}$. The words accepted by $\mathcal{A}^M$ correspond precisely to runs with stack height bounded by B and are annotated with the current stack cost along the run. These annotated costs are treated as costs of the letters, which completes the reduction.

Formally, a letter-cost function $\mathbf{lc}$ is a function from a finite alphabet of letters Σ into rationals. We assume the binary encoding of numbers. The letter-cost function extends naturally to words by $\mathbf{lc}(a_1 \dots a_n) = \mathbf{lc}(a_1) + \dots + \mathbf{lc}(a_n)$. For a finite word w, we define the average letter cost $\mathsf{avg}\mathbf{lc}(w)$ as $\frac{\mathbf{lc}(w)}{|w|}$. The average letter cost extends to infinite words as the low and the high average letter cost. For an infinite word w, we define the average low letter cost as $\mathsf{avgInf}\mathbf{lc}(w) = \liminf_{k\to\infty} \mathsf{avg}\mathbf{lc}(w[1,k])$ and the average high letter cost as $\mathsf{avgSup}\mathbf{lc}(w) = \limsup_{k\to\infty} \mathsf{avg}\mathbf{lc}(w[1,k]))$.

▶ **The `IALC` problem**: given an ω-PDA $\mathcal{A}$, a letter-cost function $\mathbf{lc}$, $\bowtie \in \{<,\leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that $\mathsf{avgInf}\mathbf{lc}(w) \bowtie \lambda$.

▶ **The `SALC` problem**: given an ω-PDA $\mathcal{A}$, a letter-cost function $\mathbf{lc}$, $\bowtie \in \{<,\leq\}$ and a threshold $\lambda \in \mathbb{Q}$, decide whether there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that $\mathsf{avgSup}\mathbf{lc}(w) \bowtie \lambda$).

In contrast to the average stack cost problem, we allow the binary encoding of numbers for $\mathbf{lc}$ and λ (a rational is encoded as a pair of integers, which are encoded in binary). The main result of this section is the following theorem:

▶ **Theorem 5.** *There are polynomial-time reductions from the `IASC` problem to the `IALC` problem and from the `SASC` problem to the `SALC` problem.*

We start the proof with auxiliary tools and lemmas.

4.1 Pumping lemma

For a run π and a position i, by $q_\pi[i]$, $a_\pi[i]$ and $u_\pi[i]$ we denote the state, letter and stack at position i of π, i.e., $(q_\pi[i], a_\pi[i], u_\pi[i]) = \pi[i]$.

Consider a run π. We define two useful functions, $\mathit{first}_\pi(i,j)$ and $\mathit{last}_\pi(i,j)$, that take a position i in π and a stack position j of the configuration $\pi[i]$ and return a position from π. Intuitively, $\pi[\mathit{first}_\pi(i,j)]$ is the configuration where the jth stack symbol in the ith configuration of π was added to the stack and $\pi[\mathit{last}_\pi(i,j)]$ is the configuration right before this stack symbol was removed from the stack. More formally, the functions first_π and last_π

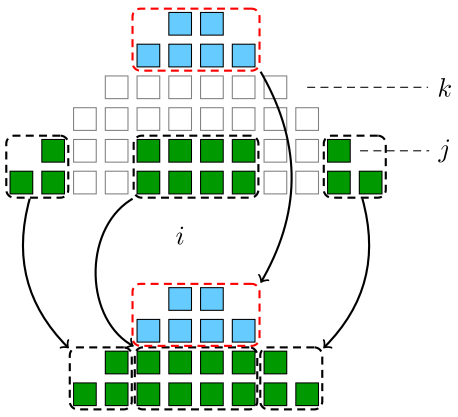

Figure 1 An example of i, j, k-contraction. The top part of the picture illustrates twelve consecutive stack contents; the sixth one is marked as i and contains two distinguished equivalent stack positions j and k. The bottom part of the picture is obtained by removing configurations 3, 4, 9 and 10 and removing stack positions 3 and 4 at positions 5-8 in the run.

are such that for each $i \in \mathbb{N}$ and each $j \in \{1, \ldots, |u_\pi[i]|\}$, $\mathit{first}_\pi(i,j)$ is a minimal number and $\mathit{last}_\pi(i,j)$ is a maximal number such that $\mathit{first}_\pi(i,j) \leq i \leq \mathit{last}_\pi(i,j)$ and all the stacks among $u_\pi[\mathit{first}_\pi(i,j)], \ldots, u_\pi[\mathit{last}_\pi](i,j)$ start with the same j stack symbols $u_1, \ldots, u_j$.

A stack position j in a configuration i is *persistent* if $\mathit{last}_\pi(i,j) = \infty$ (i.e., this symbol is never removed) and *ceasing* otherwise. We define a function *lifespan* $\mathit{ls}_\pi(i,j) = (\mathit{first}_\pi(i,j), \mathit{last}_\pi(i,j))$. For a finite word $w = w_1 w_2, \ldots, w_s$ let $w[l, \infty]$ denote the suffix $w_l w_{l+1} \ldots w_s$.

Assume a run π and a position $i \in \mathbb{N}$ such that $u_\pi[i] = u_1 \ldots u_n$. Two stack positions $j, k \in \{1, \ldots, n\}$ are *equivalent in* $\pi[i]$ if

- $u_j = u_k$ and they first appeared with the same symbols above, i.e., $u_\pi[\mathit{first}_\pi(i,j)][j,\infty] = u_\pi[\mathit{first}_\pi(i,k)][k,\infty]$, and
- $q_\pi[\mathit{first}_\pi(i,j)] = q_\pi[\mathit{first}_\pi(i,k)]$, and
- Either both j and k are persistent in i, or both are ceasing and then $q_\pi[\mathit{last}_\pi(i,j)] = q_\pi[\mathit{last}_\pi(i,k)]$.

Assume a run π, $i \in \mathbb{N}$ and two stack positions $j < k$ equivalent in $\pi[i]$. We define a i, j, k-contraction of π as a sequence π^C defined as follows:

- If j and k are ceasing, then $\pi^C = \pi[0, \mathit{first}_\pi(i,j)]\pi'\pi[\mathit{last}_\pi(i,j), \infty]$, where π' is the result of removing in $\pi[\mathit{first}_\pi(i,k)+1, \mathit{last}_\pi(i,k)-1]$ in each stack symbols at positions $j+1, j+2, \ldots, k$.
- If j and k are persistent, then $\pi^C = \pi[0, \mathit{first}_\pi(i,j)]\pi'$, where π' is the result of removing in $\pi[\mathit{first}_\pi(i,k)+1, \infty]$ in each stack symbols at positions $j+1, j+2, \ldots, k$.

The proof of the following lemma is now straightforward.

▶ **Lemma 6.** *A contraction of an accepting run is an accepting run.*

If the stack size is at least $3|Q| \cdot |\Gamma| \cdot |\delta|$, then either there are more than $2|\delta|$ persistent positions on the stack with the same stack symbol or more than $2|Q||\delta|$ ceasing positions with the same stack symbols; in both cases, one can always pick three equivalent positions among them. We state this observation as a lemma.

▶ **Lemma 7.** *Among any $3|Q| \cdot |\Gamma| \cdot |\delta|$ stack positions in any configuration $\pi[i]$ there are three stack positions pairwise equivalent in $\pi[i]$.*

4.2 The bounded stack cost property

We show the bounded stack cost property for the `IASC` and `SASC` problem.

▶ **Lemma 8.** *Assume an ω-PDA $\mathcal{A}$, stack pricing $\mathbf{c}$, $\bowtie \in \{<, \leq\}$, $\lambda \in \mathbb{Q}$ and an accepting run π such that $\mathsf{IASC}(\pi) \bowtie \lambda$. There is an accepting run π' such that $\mathsf{IASC}(\pi') \bowtie \lambda$ and for each i, $\mathbf{c}(\pi[i]) \leq \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$. The same holds for* SASC.

Proof sketch. We first show the proof for IASC and $\bowtie = \leq$. Assume an ω-PDA $\mathcal{A}$, stack pricing $\mathbf{c}$, $\lambda \in \mathbb{Q}$ and an accepting run π such that $\mathsf{IASC}(\pi) \leq \lambda$.

Let i be the smallest number such that $\mathbf{c}(\pi[j]) \leq i$ for infinitely many j. Clearly $i \leq \lambda$ since $\mathsf{IASC}(\pi) \leq \lambda$. If there are only finitely many positions where the stack cost exceeded $\max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$, then for every such a position i we can find, by Lemma 7, two equivalent stack positions $j < k$ and obtain the i, j, k-contraction of the run. We repeat it until the cost reaches the desired bound. Since we repeat this only finitely many times for the whole run, the obtained run π' is accepting and $\mathsf{IASC}(\pi') \leq \lambda$.

For the rest of this proof, we focus on the case where there are infinitely many positions with costly stack. There are two new challenges now: we need to make sure to preserve infinitely many accepting states, and guarantee that IASC stays within desired bound.

To preserve infinitely many accepting states, we decompose π as $\pi_1^{ok}\pi_1\pi_2^{ok}\pi_2\pi_3^{ok}\ldots$ such that every positions in π_i has stack cost exceeding λ and all the positions of π_i^{ok} cost at most λ. Our goal is to define a new run π' that is obtained from π by contracting some of the runs among $\pi_1, \pi_2, \ldots$. To preserve the acceptance condition, we mark one configuration with an accepting state in each π_i that has such a configuration and guarantee that this state will be retained.

Let $\pi' = \pi_1^{ok}\pi_1'\pi_2^{ok}\pi_2'\pi_3^{ok}\ldots$, where for each i, we define π_i' starting from π_i, and then by repeating the following procedure as long as needed. For any j such that $\mathbf{c}(\pi_i'[j]) > \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$, there are at least $3|Q| \cdot |\Gamma| \cdot |\delta|$ stack positions k whose symbols have positive cost and lifespan is within π', as the total cost of the remaining stack position is bounded by λ. Among them, we can find three pairwise equivalent stack positions, from which one can choose two positions k, l such that the j, k, l-contraction of π_i' retains the marked accepting state. We set π_i' to be the contraction.

For each position i in π', we define its origin $o(i)$ as the position in π from which i originates (o is a monotonic function). We argue that if $\mathsf{ASC}(\pi, \mathbf{c}, i) \leq \lambda$, then $\mathsf{ASC}(\pi', \mathbf{c}, o(i)) \leq \lambda$; the proof is based on the fact that we only remove or alter positions where the cost is greater than λ (see [7] for details). It follows that $\mathsf{IASC}(\pi') \leq \mathsf{IASC}(\pi)$, as required.

For the strict inequality assume that $\mathsf{IASC}(\pi) < \lambda$. Then for some $\epsilon > 0$, we have $\mathsf{IASC}(\pi) \leq \lambda - \epsilon$. By the above reasoning, there exists π' such that $\mathsf{IASC}(\pi') \leq \lambda - \epsilon$ and for each i, $\mathbf{c}(\pi[i]) \leq \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$. Then, $\mathsf{IASC}(\pi') < \lambda$.

The case of SASC uses the same construction, but the reasoning now is slightly more technical as we have to argue that all the subsequences of π' have the cost less than λ, but the idea is exactly the same, so we skip it here. ◀

We now prove Theorem 5.

Proof. We show the reduction of `IASC` to `IALC`. The reduction is through a construction of a *meta-automaton* defined below. Given an instance $I = \langle \mathcal{A}, \mathbf{c}, \bowtie, \lambda \rangle$ of the `IASC` problem, we define a *meta-automaton* $\mathcal{A}^M$ for I as an ω-PDA that recognizes the language of infinite words corresponding to accepting runs of $\mathcal{A}$. Formally, let $N = \max_{s\in\Gamma} \mathbf{c}(s) \cdot 3|Q| \cdot |\Gamma| \cdot |\delta| + \lambda$. The ω-PDA $\mathcal{A}^M$ works over the alphabet $\delta \times \{0, \ldots, N\}$, where δ is the transition relation of $\mathcal{A}$, and $\mathcal{A}^M$ accepts all words w such that (1) w encodes an accepting run π^w of $\mathcal{A}$, and

(2) the stack cost of π at position i is encoded as the second component of $w[i]$. In particular, (2) implies that the meta-automaton accepts words that correspond to runs of $\mathcal{A}$ in which the stack cost is bounded by N at every position.

To build a meta-automaton we need to track the current stack cost. However, even a finite-state automaton can store in its states the current stack cost, which belongs to $\{0,\dots,N\}$ and update it based in the current symbol being pushed to or popped from the stack. Therefore, a meta-automaton can be constructed in polynomial time.

Consider a letter-cost function defined over $\delta \times \{0,\dots,N\}$ such that $\mathbf{lc}(t,x) = x$, i.e., the letter cost of a transition is the current stack cost. Observe that each partial averages of stack cost in a run π coincides with the corresponding partial average of letter costs in the corresponding word. Therefore, if the instance $(\mathcal{A}^M, \mathbf{lc}, \bowtie, \lambda)$ of the `IALC` problem is solved by a word w, then the corresponding run π^w of $\mathcal{A}$ is a solution of I. Conversely, if I has a solution π, then by Lemma 8 it has a solution π', in which all stack costs are bounded by N. Then, there is a word w' accepted by $\mathcal{A}^M$ corresponding to the run π'. Observe that w' is a solution of the instance $(\mathcal{A}^M, \mathbf{lc}, \bowtie, \lambda)$ of the `IALC` problem.

The same construction gives us the reduction from `SASC` to `SALC`. The prove of correctness is the same as we only need to use different variant of Lemma 8. ◀

5 The average letter cost problem

We prove that the average letter cost problem can be solved in polynomial time. In Section 5.1 we study the finite-word variants of the average letter cost problem. Next, we use finite-word results to solve the average letter cost problem over infinite word (Section 5.2). We supplement this section with the comparison of the average letter cost problem and one-player games on WPSs with conjunctions of mean-payoff and Büchi objectives.

5.1 Average letter cost over finite words

We are interested in the average letter cost over all (finite) words accepted by a given PDA.

▶ **The avglc problem**: given a letter-cost function $\mathbf{lc}$, a PDA $\mathcal{A}$, $\bowtie \in \{<, \leq\}$ and a threshold λ, decide whether $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathsf{avglc}(w) \bowtie \lambda$.

To solve this problem, we first discuss how to compute the infimum of $\mathbf{lc}(w)$ over all words accepted by $\mathcal{A}$, i.e., $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathbf{lc}(w)$. Next, we solve the average letter cost problem by computing $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathbf{lc}^{\lambda}(w)$ for a modified letter cost function $\mathbf{lc}^{\lambda}$.

▶ **Lemma 9.** *Given a PDA $\mathcal{A}$ and a letter-cost function $\mathbf{lc}$, we can compute $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathbf{lc}(w)$, the infimum of $\mathbf{lc}(w)$ over all words accepted by $\mathcal{A}$, in polynomial time in $\mathcal{A}$.*

▶ Remark. The values of the letter-cost function in Lemma 9 can be represented in binary.

Proof sketch. To compute $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathbf{lc}(w)$, we transform $\mathcal{A}$ to a CFG G generating the same language. Then, we adapt the classic algorithm for checking the emptiness of the language generated by a CFG [6]. The algorithm from [6] marks iteratively non-terminals that derive some words. It starts by marking non-terminals that derive a single letter. Then it takes $|G|$ iterations of a loop, in which it applies all the rules of G, and marks non-terminals that derive marked non-terminals. Here, we associate with each non-terminal A, a variable v_A storing the minimal value of $\mathbf{lc}(w)$ for w derivable from A in G. We update values of v_A in each iteration, by putting $v_A = \min(v_A, v_B + v_C)$ for every rule $A \to BC$. The algorithm terminates if (1) there is an iteration, in which no variable has changed or (2) after $|G| + 1$ iterations. In the first case, we return v_S, the computed value for the start symbol. In the second case, we observe that no further iterations are necessary as there exists a derivation $B \to_G^* v_L B v_R$ with $\mathbf{lc}(v_L v_R) < 0$, and hence $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathbf{lc}(w) = -\infty$. ◀

Using Lemma 9, we can solve the average letter cost problem in the finite word case.

▶ **Lemma 10.** *The* avglc *problem can be solved in polynomial time.*

Proof sketch. First, if $\inf_{w \in \mathcal{L}(\mathcal{A})} \mathsf{avg}\mathbf{lc}(w) < \lambda$, then there is a word w with $\mathsf{avg}\mathbf{lc}(w) < \lambda$. Observe that the average of $a_1, \ldots, a_n$ is less than lambda if and only if the sum of $(a_1 - \lambda), \ldots, (a_n - \lambda)$ is less than 0. Therefore, to check existence of w with $\mathsf{avg}\mathbf{lc}(w) < \lambda$, we define $\mathbf{lc}^\lambda$ by substructing λ from each value of $\mathbf{lc}$ and apply Lemma 9 to check existence of w with $\mathbf{lc}^\lambda(w) < 0$. The case of the non-strict inequality is more difficult as the infimum need not be realized. However , we consider a CFG G generating the language of $\mathcal{A}$ and we show that $\inf_{u \in \mathcal{L}(\mathcal{A})} \mathsf{avg}\mathbf{lc}(u) \leq \lambda$ if and only if (1) there exists $u \in \mathcal{L}(\mathcal{A})$ with $\mathsf{avg}\mathbf{lc}(u) \leq \lambda$, or (2) there is a non-terminal A such that $A \to_G^* u_L A u_R$ and $\mathsf{avg}\mathbf{lc}(u_L u_R) \leq \lambda$. Both conditions can be checked in polynomial time using the letter-cost function $\mathbf{lc}^\lambda$ and Lemma 9. ◀

5.2 Average letter cost over infinite words

In this section, we discuss the average letter cost problem in the infinite-word case. We begin with the average letter cost problem with the limit supremum of partial averages.

▶ **Lemma 11.** *The* `SALC` *problem can be decided in polynomial time.*

Proof sketch. We reduce the problem to the finite-word case and use Lemma 10. Consider an instance of the `SALC` problem consisting of an ω-PDA $\mathcal{A}$, letter-cost function $\mathbf{lc}$, $\bowtie \in \{<, \leq\}$ and λ. Any context-free omega language $\mathcal{L}$ can be presented as $\mathcal{L} = \bigcup_{1 \leq i \leq k} V_i(U_i)^\omega$, where $V_1, U_1, \ldots, V_k, U_k$ are non-empty finite-word context-free languages. We can look for w in each $V_i(U_i)^\omega$ separately. Then, we show that there exists a word $w \in V_i(U_i)^\omega$ such that $\mathsf{avgSup}\mathbf{lc}(w) \bowtie \lambda$ if and only if $\inf_{u \in U_i} \mathsf{avg}\mathbf{lc}(u_i) \bowtie \lambda$. The latter condition can be checked in polynomial time (Lemma 10). ◀

Now, we consider the average letter cost problem with the limit infimum of partial averages. We again reduce it to the finite-word case, but now the reduction is not as straightforward. The following example explains the main difficulty.

▶ **Example 12.** Consider $\Sigma = \{0, 2\}$, and the letter-cost function $\mathbf{lc}$, which simply returns the value of a letter, i.e., for $x \in \Sigma$ we have $\mathbf{lc}(x) = x$. Now, let $\mathcal{A}$ be an ω-PDA accepting the language U_0^ω, where $U_0 = \{0^n 2^n \mid n \in \mathbb{N}\}$. Observe that for every $w \in \mathcal{L}(\mathcal{A})$ we have $\mathsf{avgSup}\mathbf{lc}(w) = 1$. However, for a word $w_0 = 020^2 2^2 \ldots 0^{2^{n^2}} 2^{2^{n^2}} \ldots$ we observe that $\mathsf{avg}\mathbf{lc}(020^2 2^2 \ldots 0^{2^{n^2}}) < 2^{(n-1)^2 + 1 - n^2} = 2^{-2(n-1)}$, and hence $\mathsf{avgInf}\mathbf{lc}(w_0) = 0$. Therefore, $\mathsf{avgSup}\mathbf{lc}(w_0) \neq \mathsf{avgInf}\mathbf{lc}(w_0)$. We conclude that for a language of the form U^ω, knowing average letter costs of words in U is insufficient to decide whether there is a word w with $\mathsf{avgInf}\mathbf{lc}(w) \leq \lambda$. Still, in the following we show how to decide $\mathsf{avgInf}\mathbf{lc}(w) \leq \lambda$ by examining the structure of U.

▶ **Lemma 13.** *The* `IALC` *problem can be decided in polynomial time.*

Proof sketch. Again we represent the language of an ω-PDA $\mathcal{A}$ as $\mathcal{L} = \bigcup_{1 \leq i \leq k} V_i(U_i)^\omega$, where $V_1, U_1, \ldots, V_k, U_k$ are non-empty finite-word context-free languages and look for w in each $V_i(U_i)^\omega$ separately. Let G_i be a CFG generating the language U_i. We assume that G_i is pruned, i.e., every non-terminal occurs in some derivation of some word. For

$\bowtie \in \{<, \leq\}$, we show that there exists a word $w \in V_i(U_i)^\omega$ such that $\mathsf{avgInflc}(w) \bowtie \lambda$ if and only if (1) $\inf_{u \in U_i} \mathsf{avglc}(u_i) \bowtie \lambda$ or (2) there exists a non-terminal A in G_i, such that $\inf\{\mathsf{avglc}(u_L) \mid A \rightarrow^*_{G_i} u_L A u_R\} \bowtie \lambda$. Both conditions can be checked in polynomial time using Lemma 10. Condition (1) is inherited from the $\mathsf{avgSuplc}$. For condition (2), observe that $\mathsf{avgInflc}(w) \leq \lambda$ if there is a subsequence of partial averages that converges to a value at most λ. For a word $vu_1u_2 \ldots \in V_i(U_i)^\omega$, the subsequence from the limit infimum may pick only positions inside words $u_1, u_2, \ldots$ (as in Example 12), and the subsequence of partial averages at boundaries of words may converge to a higher value. Condition (2) covers this case. In Example 12, U_0 is generated by a grammar $S \rightarrow 0S2, S \rightarrow \epsilon$ and observe that this grammar satisfies condition (2) with $\lambda = 0$, i.e., for $S \rightarrow 0S2$ we have $\mathsf{avglc}(0) \leq 0$. ◀

5.3 Weighted pushdown systems with fairness

We briefly discuss the connection between the average letter cost problem and one-player games on WPSs with conjunctions of mean-payoff and Büchi objectives.

A *WPS-game* consists of a WPS $\mathcal{P} = (\mathcal{A}, \mathrm{wt})$ and a *game objective.* In each WPS-game, the only player plays infinitely many rounds selecting consecutive transitions in order to obtain a run satisfying given objectives. A game objective is a conjunction of a *mean-payoff* objective and a Büchi objective, defined as follows.

A *mean-payoff* objective is of the form $\mathsf{LimAvgInf}(\pi) \bowtie \lambda$ or $\mathsf{LimAvgSup}(\pi) \bowtie \lambda$, where $\bowtie \in \{<, \leq\}$ and $\lambda \in \mathbb{Q}$. The interpretation of such an objective is as follows: each play constructs a run π of $\mathcal{P}$ and the *cost sequence* $\mathbf{wt}(\pi)$ of π which is the sequence of costs of the transitions of π. We interpret $\mathsf{LimAvgInf}(\pi)$ as $\liminf_{k\rightarrow\infty} \frac{1}{k} \sum_{i=1}^{k} \mathbf{wt}(\pi)[i]$ and $\mathsf{LimAvgSup}(\pi)$ as $\limsup_{k\rightarrow\infty} \frac{1}{k} \sum_{i=1}^{k} \mathbf{wt}(\pi)[i]$, and we say that a mean-payoff objective is satisfied if its inequality holds. A Büchi objective is a set of states; it is satisfied by π if π visits some state from Q_F infinitely often.

To *solve a WPS-game* is to determine whether the player can construct a run that satisfies all the game objectives. The following theorem extends [5, Theorem 1] by adding Büchi objectives.

▶ **Theorem 14.** *Each WPS-game can be solved in polynomial time.*

The proof follows from a reduction to the average letter cost problem that encodes the transitions costs in corresponding letter costs. A converse polynomial-time reduction is also possible; in this case, we encode letter costs in transition costs.

6 Average Response Time Example

In this section, we use the ASC problem to compute a variant of the average response time property [3]. In this variant, there are two agents: a *client* and a *server.* A client can state a request, which is later granted or rejected by the server. Requests are dealt with on the first-come, first-served basis, but not immediately — the server may need some time to issue a grant. We assume that both client and server are modeled as systems with finitely many states and can check whether the number of pending requests at a given moment is zero. We also assume a fairness condition stating that there are infinitely many requests and grants.

A trace of such system is a word over the alphabet $\{r, g, \#\}$, where r denotes a new request, g denotes a grant and $\#$ denotes a null instruction. We are interested in bounding the minimal possible average response time of such a model . In other words, we are interested checking, for a given λ and a model, whether

$$\liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} a_i < \lambda \tag{1}$$

for some computation of the model in which the ith request was realised after a_i steps of computations (our technique works for $\limsup$ as well, but we focus on $\liminf$).

Feasibility study. To apply our technique, we need to overcome two main difficulties. First, our technique only works for stacks, but requests are handled in a queue manner. In general, non-emptiness of automata with queue is undecidable. Second, the denominator in (1) refers only to the number of requests, not the number of positions in words (they may differ because of the letter #).

Dealing with queues. We abstract the counter to a stack over a unary alphabet $\{P\}$ whose size equals the value of the counter in the straightforward way.

We claim that there is a run satisfying (1) if and only if there is a run satisfying

$$\liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) < \lambda \tag{2}$$

where G_n denotes the position of the nth grant in the run (we assume that each grant correspond to a request).

We present the main idea (see [7] for details). Consider a position in a run n where there are no pending requests; then, the total waiting time of all requests up to this position is equal to the sum of the number of waiting processes in each position up to n. At a position with unfulfilled requests, this is no longer guaranteed, as the pending processes may have some waiting time in the future. However, it can be shown that a run for (2) can be chosen in a way that guarantees that the difference between the two numbers is bounded by some constant, and therefore can be neglected in the $\liminf$.

Selected positions. We argue that (2) is equivalent to

$$\liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda \tag{3}$$

where $\mathbf{c}^*(\pi[i])$ equals $\mathbf{c}(\pi[i])$ if the position i corresponds to a grant and $\mathbf{c}(\pi[i]) + \lambda$ otherwise.

Observe that $\frac{1}{n}\sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) < \lambda$ iff $\sum_{i=1}^{g_n} \mathbf{c}(\pi[i]) - n\lambda + g_n\lambda < g_n\lambda$ iff $\frac{1}{g_n}\sum_{i=1}^{G_n} \mathbf{c}^*(\pi[i]) < \lambda$. The last equivalence follows from the fact that there are n grants, and so $\sum_{i=1}^{G_n} \mathbf{c}^*(\pi[i]) = \sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) + (G_n - n)\lambda$.

If (2), then there is an infinite sequence of positions where $\frac{1}{n}\sum_{i=1}^{G_n} \mathbf{c}(\pi[i]) < \lambda$, and by the above reasoning each position in this sequence satisfies $\frac{1}{G_n}\sum_{i=1}^{G_n} \mathbf{c}^*(\pi[i]) < \lambda$. This means that (2) implies (3). The converse if also true. To see this, observe that if at a position $n > 0$ that does not correspond to a grant we have $\frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda$, then also $\frac{1}{n-1}\sum_{i=1}^{n-1} \mathbf{c}^*(\pi[i]) < \lambda$ as $\mathbf{c}^*(\pi[n]) \geq \lambda$. If we have an infinite sequence of positions with $\frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda$ and infinitely many grants, we can select an infinite sequence of positions n corresponding to grants such that $\frac{1}{n}\sum_{i=1}^{n} \mathbf{c}^*(\pi[i]) < \lambda$.

Putting it all together. From the above consideration, we know that (1) if and only if (3). Therefore, to verify (1), we modify the automaton as follows: we add an additional stack symbol $\bullet$ of weight $\lambda + 1$ that can only appear at the top of the stack. We modify the transition function to stipulate that whenever the automaton is in a position that does not correspond to a grant, then the topmost symbol is $\bullet$. By our results, checking whether there is a run with the average stack cost less than λ (and therefore whether the average waiting time is less than λ) can be done in polynomial time.

References

1 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G Larsen, and Simon Laursen. Average-energy games. *Acta Informatica*, pages 1–37, 2015.

2 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.

3 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. In *LICS 2015*, pages 725–737, 2015.

4 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Bidirectional nested weighted automata. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.5.

5 Krishnendu Chatterjee and Yaron Velner. Mean-payoff pushdown games. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 195–204. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.30.

6 John E. Hopcroft and Jefferey D. Ullman. *Introduction to Automata Theory, Languages, and Comput ation.* Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.

7 Jakub Michaliszyn and Jan Otop. Average stack cost of buechi pushdown automata. *CoRR*, abs/1710.04490, 2017. URL: http://arxiv.org/abs/1710.04490.

8 Yasuhiko Minamide. Weighted pushdown systems with indexed weight domains. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 230–244. Springer, 2013.

9 Thomas W. Reps, Akash Lal, and Nicholas Kidd. Program analysis using weighted pushdown systems. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 23–51. Springer, 2007. doi:10.1007/978-3-540-77050-3_4.

10 Thomas W. Reps, Stefan Schwoon, Somesh Jha, and David Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005. doi:10.1016/j.scico.2005.02.009.