Ramsey Goes Visibly Pushdown

Oliver Friedmann[1], Felix Klaedtke[2], and Martin Lange[3]

[1] LMU Munich, [2] ETH Zurich, and [3] University of Kassel

Abstract. Checking whether one formal language is included in another is vital to many verification tasks. In this paper, we provide solutions for checking the inclusion of the languages given by visibly pushdown automata, over both finite and infinite words. Visibly pushdown automata are a richer automaton model than the classical finite-state automata, which allows one, e.g., to reason about the nesting of procedure calls in the executions of recursive imperative programs. The highlight of our solutions is that they do not comprise automata constructions for determinization and complementation. Instead, our solutions are more direct and generalize the so-called Ramsey-based inclusion-checking algorithms, which apply to classical finite-state automata and proved effective there, to visibly pushdown automata. We also experimentally evaluate our algorithms thereby demonstrating the virtues of avoiding determinization and complementation constructions.

1 Introduction

Various verification tasks can be stated more or less directly as inclusion problems of formal languages or comprise inclusion problems as subtasks. For example, the model-checking problem of non-terminating finite-state systems with respect to trace properties boils down to the question whether the inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for two Büchi automata $\mathcal{A}$ and $\mathcal{B}$ holds, where $\mathcal{A}$ describes the traces of the system and $\mathcal{B}$ the property [21]. Another application of checking language inclusion for Büchi automata appears in size-change termination analysis [13,18]. Inclusion problems are in general difficult. For Büchi automata it is PSPACE-complete.

From the closure properties of the class of ω-regular languages, i.e., those languages that are recognizable by Büchi automata it is obvious that questions like the one above for model checking non-terminating finite-state systems can be effectively reduced to an emptiness question, namely, $L(\mathcal{A}) \cap L(\mathcal{C}) = \emptyset$, where $\mathcal{C}$ is a Büchi automaton that accepts the complement of $\mathcal{B}$. Building a Büchi automaton for the intersection of the languages and checking its emptiness is fairly easy: the automaton accepting the intersection can be quadratically bigger, the emptiness problem is NLOGSPACE-complete, and it admits efficient implementations, e.g., by a nested depth-first search. However, complementing Büchi automata is challenging. One intuitive reason for this is that not every Büchi automaton has an equivalent deterministic counterpart. Switching to a richer acceptance condition like the parity condition so that determinization would be possible is currently not an option in practice. The known determinization constructions for richer acceptance conditions are intricate, although complementation would then be easy by dualizing the acceptance condition. A lower bound on the complementation problem with respect to the automaton size is $2^{\Omega(n \log n)}$. Known constructions for complementing Büchi automata that match this lower bound are also intricate. As a matter of fact, all attempts so far that explicitly construct the automaton $\mathcal{C}$ from $\mathcal{B}$ scale poorly. Often, the implementations produce automata for the complement language that are huge, or they even fail to produce an output at all in reasonable time and space if the input automaton has more than 20 states, see, e.g., [5,20].

Other approaches for checking the inclusion of the languages given by Büchi automata or solving the closely related but simpler universality problem for Büchi automata have recently gained considerable attention [1,2,8–10,13,14,18]. In the worst case, these algorithms have exponential running times, which are often worse than the $2^{\Omega(n \log n)}$ lower bound on complementing Büchi automata. However, experimental results, in particular, the ones for the so-called Ramsey-based algorithms show that the performance of these algorithms is superior. The name *Ramsey-based* stems from the fact that their correctness is established by relying on Ramsey's theorem [19].[1]

The Ramsey-based algorithms for checking universality $L(\mathcal{B}) = \Sigma^\omega$ iteratively build a set of finite graphs starting from a finite base set and closing it off under a composition operation. These graphs capture $\mathcal{B}$'s essential behavior on finite words. The language of $\mathcal{B}$ is not universal iff this set contains graphs with certain properties that witness the existence of an infinite word that is not accepted by $\mathcal{B}$. First, there must be a graph that is idempotent with respect to the composition operation. This corresponds to the fact that all the runs of $\mathcal{B}$ on the finite words described by the graph loop. We must also require that no accepting state occurs on these loops. Second, there must be another graph for the runs on a finite word that reach that loop. To check the inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ the graphs are annotated with additional information about runs of $\mathcal{A}$ on finite words. Here, in case of $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$, the constructed set of graphs contains graphs that witness the existence of at least one infinite word that is accepted by $\mathcal{A}$ but all runs of $\mathcal{B}$ on that word are rejecting. The Ramsey-based approach generalizes to parity automata [15]. The parity condition is useful in modeling reactive programs in which certain modules are supposed to terminate and others are not supposed to terminate. Also, certain Boolean combinations of Büchi (non-termination) and co-Büchi (termination) conditions can easily be expressed as a parity condition. Although parity automata can be translated into Büchi automata, it algorithmically pays off to handle parity automata directly [15].

In this paper, we extend the Ramsey-based analysis to visibly pushdown automata (VPAs) [4]. This automaton model restricts nondeterministic pushdown automata in the way that the input symbols determine when the pushdown automaton pushes or pops symbols from its stack. In particular, the stack heights are identical at the same positions in every run of any VPA on a given input. It is because of this syntactic restriction that the class of visibly pushdown lan-

[1] Büchi's original complementation construction, which also relies on Ramsey's theorem, shares similarities with these algorithms. However, there is significantly less overhead when checking universality and inclusion directly and additional heuristics and optimizations are applicable [1,5].

guages retains many closure properties like intersection and complementation. VPAs allow one to describe program behavior in more detail than finite-state automata. They can account for the nesting of procedures in executions of recursive imperative programs. Non-regular properties like "an acquired lock must be released within the same procedure" are expressible by VPAs. Model checking of recursive state machines [3] and Boolean programs, which are widely used as abstractions in software model checking, can be carried out in this refined setting by using VPAs for representing the behavior of the programs and the properties. Similar to the automata-theoretic approach to model checking finite-state systems, checking the inclusion of the languages of VPAs is vital here. This time, the respective decision problem is even EXPTIME-complete. Other applications for checking language inclusion of VPAs when reasoning about recursive imperative programs also appear in conformance checking [11] and in the counterexample-guided-abstraction-refinement loop [16].

A generalization of the Ramsey-based approach to VPAs is not straightforward since the graphs that capture the essential behavior of an automaton must also account for the stack content in the runs. Moreover, to guarantee termination of the process that generates these graphs, an automaton's behavior of all runs must be captured within finitely many such graphs. In fact, when considering pushdown automata in general such a generalization is not possible since the universality problem for pushdown automata is undecidable. We circumvent this problem by only considering graphs that differ in their stack height by at most one, and by refining the composition of such graphs in comparison to the unrestricted way that graphs can be composed in the Ramsey-based approach to finite automata. Then the composition operation only needs to account for the top stack symbols in all the runs described by the graphs, which yields a finite set of graphs in the end.

The main contribution of this paper is the generalization of the Ramsey-based approach for checking universality and language inclusion for VPAs over infinite inputs, where the automata's acceptance condition is stated as a parity condition. This approach avoids determinization and complementation constructions. The respective problems where the VPAs operate over finite inputs are special cases thereof. We also experimentally evaluate the performance of our algorithms showing that the Ramsey-based inclusion checking for is more efficient than methods that are based on determinization and complementation.

The remainder of this paper is organized as follows. In Sect. 2, we recall the framework of VPAs. In Sect. 3, we provide a Ramsey-based universality check for VPAs. Note that universality is a special case of language inclusion. We treat universality in detail to convey the fundamental ideas first. In Sect. 4, we extend this to a Ramsey-based inclusion check for the languages given by VPAs. In Sect. 5, we report on the experimental evaluation of our algorithms. In Sect. 6, we draw conclusions. Additional details are given in the appendix.

2 Preliminaries

Words. The set of finite words over the alphabet Σ is Σ^* and the set of infinite words over Σ is Σ^ω. Let $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$, where ε is the empty word. The length

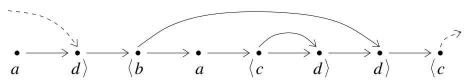

Fig. 1. Nested word $w = adbacddbc$ with $\Sigma_{\mathsf{int}} = \{a\}$, $\Sigma_{\mathsf{call}} = \{b, c\}$, and $\Sigma_{\mathsf{ret}} = \{d\}$. Its pending positions are 1 and 7 with $w_1 = d$ and $w_7 = c$. The call position 2 with $w_2 = b$ matches with the return position 6 with $w_6 = d$. The positions 4 and 5 also match.

of a word w is written as $|w|$, where $|w| = \omega$ when w is an infinite word. For a word w, w_i denotes the letter at position $i < |w|$ in w. That is, $w = w_0 w_1 \dots$ if w is infinite and $w = w_0 w_1 \dots w_{n-1}$ if w is finite and $|w| = n$. With $\inf(w)$ we denote the set of letters of Σ that occur infinitely often in $w \in \Sigma^\omega$.

Nested words [4] are linear sequences equipped with a hierarchical structure, which is imposed by partitioning an alphabet Σ into the pairwise disjoint sets Σ_{int}, Σ_{call}, and Σ_{ret}. For a finite or infinite word w over Σ, we say that the position $i \in \mathbb{N}$ with $i < |w|$ is an *internal position* if $w_i \in \Sigma_{\mathsf{int}}$. It is a *call position* if $w_i \in \Sigma_{\mathsf{call}}$ and it is a *return position* if $w_i \in \Sigma_{\mathsf{ret}}$. When attaching an opening bracket $\langle$ to every call position and closing brackets $\rangle$ to the return positions in a word w, we group the word w into subwords. This grouping can be nested. However, not every bracket at a position in w needs to have a matching bracket. The call and return positions in a nested word without matching brackets are called *pending*. To emphasize this hierarchical structure imposed by the brackets $\langle$ and $\rangle$, we also refer to the words in $\Sigma^* \cup \Sigma^\omega$ as *nested words*. See Fig. 1 for illustration.

To ease the exposition, we restrict ourselves in the following to nested words without pending positions.[2] For $\sharp \in \{*, \omega\}$, $NW^\sharp(\Sigma)$ denotes the set of words in $\Sigma^\sharp$ with no pending positions. These words are also called *well-matched*.

Automata. A *visibly pushdown automaton* [4], VPA for short, is a tuple $\mathcal{A} = (Q, \Gamma, \Sigma, \delta, q_I, \Omega)$, where Q is a finite set of states, Γ is a finite set of stack symbols with $\bot \notin \Gamma$, $\Sigma = \Sigma_{\mathsf{int}} \cup \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$ is the input alphabet, δ consists of three transition functions $\delta_{\mathsf{int}} : Q \times \Sigma_{\mathsf{int}} \to 2^Q$, $\delta_{\mathsf{call}} : Q \times \Sigma_{\mathsf{call}} \to 2^{Q \times \Gamma}$, and $\delta_{\mathsf{ret}} : Q \times (\Gamma \cup \{\bot\}) \times \Sigma_{\mathsf{ret}} \to 2^Q$, $q_I \in Q$ is the initial state, and $\Omega : Q \to \mathbb{N}$ is the priority function. We sometimes write $\Gamma_\bot$ as a short form for $\Gamma \cup \{\bot\}$. We write $\Omega(Q)$ to denote the set of all priorities used in $\mathcal{A}$, i.e. $\Omega(Q) := \{\Omega(q) \mid q \in Q\}$. The *size* of $\mathcal{A}$ is $|Q|$ and its *index* is $|\Omega(Q)|$.

A *run* of $\mathcal{A}$ on $w \in \Sigma^\omega$ is a word $(q_0, \gamma_0)(q_1, \gamma_1) \dots \in (Q \times \Gamma_\bot^+)^\omega$ with $(q_0, \gamma_0) = (q_I, \bot)$ and for each $i \in \mathbb{N}$, the following conditions hold:

1. If $w_i \in \Sigma_{\mathsf{int}}$ then $q_{i+1} \in \delta_{\mathsf{int}}(q_i, w_i)$ and $\gamma_{i+1} = \gamma_i$.
2. If $w_i \in \Sigma_{\mathsf{call}}$ then $(q_{i+1}, B) \in \delta_{\mathsf{call}}(q_i, w_i)$ and $\gamma_{i+1} = B\gamma_i$, for some $B \in \Gamma$.
3. If $w_i \in \Sigma_{\mathsf{ret}}$ and $\gamma_i = Bu$ with $B \in \Gamma_\bot$ and $u \in \Gamma_\bot^*$ then $q_{i+1} \in \delta_{\mathsf{ret}}(q_i, B, w_i)$ and $\gamma_{i+1} = u$ if $u \neq \varepsilon$ and $\gamma_{i+1} = \bot$, otherwise.

The run is *accepting* if $\max\{\Omega(q) \mid q \in \inf(q_0 q_1 \dots)\}$ is even. Runs of $\mathcal{A}$ on finite words are defined as expected. In particular, a run on a finite word is accepting if the last state in the run has an even priority. For $\sharp \in \{*, \omega\}$, we define

$$L^\sharp(\mathcal{A}) := \bigl\{ w \in NW^\sharp(\Sigma) \bigm| \text{there is an accepting run of } \mathcal{A} \text{ on } w \bigr\} .$$

Priority and Reward Ordering. For an arbitrary set S, we always assume that $\dagger$ is a distinct element not occurring in S. We write $S_\dagger$ for $S \cup \{\dagger\}$. We use $\dagger$ to explicitly speak about partial functions into S, i.e., $\dagger$ denotes undefinedness.

[2] Our results extend to nested words with pending positions; see appendix.

We define the following two orders on $\mathbb{N}_\dagger$. The *priority ordering* is denoted $\sqsubseteq$ and is the standard order of type $\omega + 1$. Thus, we have $0 \sqsubset 1 \sqsubset 2 \sqsubset \cdots \sqsubset \dagger$. The *reward ordering* $\preceq$ is defined by $\dagger \prec \cdots \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec \cdots$. Note that $\dagger$ is maximal for $\sqsubseteq$ but minimal for $\preceq$. For a finite nonempty set $S \subseteq \mathbb{N}_\dagger$, $\bigsqcup S$ and $\bigcurlyvee S$ denote the maxima with respect to the priority ordering $\sqsubseteq$ and the reward ordering $\preceq$, respectively. Furthermore, we write $c \sqcup c'$ for $\bigsqcup\{c, c'\}$.

The reward ordering reflects the intuition of how valuable a priority of a VPA's state is for acceptance: even priorities are better than odd ones, and the bigger an even one is the better, while small odd priorities are better than bigger ones because it is easier to subsume them in a run with an even priority elsewhere. The element $\dagger$ stands for the non-existence of a run.

3 Universality Checking

Throughout this section, we fix a VPA $\mathcal{A} = (Q, \Gamma, \Sigma, \delta, q_I, \Omega)$. We describe an algorithm that determines whether $L^\omega(\mathcal{A}) = NW^\omega(\Sigma)$, i.e., whether $\mathcal{A}$ accepts all well-matched infinite nested words over Σ.[3]

Central to the algorithm are so-called transition profiles. They capture $\mathcal{A}$'s essential behavior on finite words.

Definition 1. There are three kinds of *transition profiles*, TP for short. The first one is an int-TP, which is a function of type $Q \times Q \to \Omega(Q)_\dagger$. We associate with a symbol $a \in \Sigma_{\mathsf{int}}$ the int-TP f_a. It is defined as

$$f_a(q, q') \;:=\; \begin{cases} \Omega(q') & \text{if } q' \in \delta_{\mathsf{int}}(q, a) \text{ and} \\ \dagger & \text{otherwise.} \end{cases}$$

A call-TP is a function of type $Q \times \Gamma \times Q \to \Omega(Q)_\dagger$. With a symbol $a \in \Sigma_{\mathsf{call}}$ we associate the call-TP f_a. It is defined as

$$f_a(q, B, q') \;:=\; \begin{cases} \Omega(q') & \text{if } (q', B) \in \delta_{\mathsf{call}}(q, a) \text{ and} \\ \dagger & \text{otherwise.} \end{cases}$$

Finally, a ret-TP is a function of type $Q \times \Gamma_\perp \times Q \to \Omega(Q)_\dagger$. With a symbol $a \in \Sigma_{\mathsf{ret}}$ we associate the ret-TP f_a. It is defined as

$$f_a(q, B, q') \;:=\; \begin{cases} \Omega(q') & \text{if } q' \in \delta_{\mathsf{ret}}(q, B, a) \text{ and} \\ \dagger & \text{otherwise.} \end{cases}$$

A TP of the form f_a for an $a \in \Sigma$ is also called *atomic*. For $\tau \in \{\mathsf{int}, \mathsf{call}, \mathsf{ret}\}$, we define the set of atomic TPs as $T_\tau := \{f_a \mid a \in \Sigma_\tau\}$.

The above TPs describe $\mathcal{A}$'s behavior when $\mathcal{A}$ reads a single letter. In the following, we define how TPs can be composed to describe $\mathcal{A}$'s behavior on words of finite length. The composition, written $f \circ g$, can only be applied to TPs of certain kinds. This ensures that the resulting TP describes the behavior on a word w such that, after reading w, $\mathcal{A}$'s stack height has changed by at most one.

[3] An extension of the algorithm to account for non-well-matched nested words and a universality check for VPAs over finite words is given in App. E and F, respectively. Moreover, in App. C, we present a complementation construction for VPAs based on determinization and compare it to the presented algorithm.

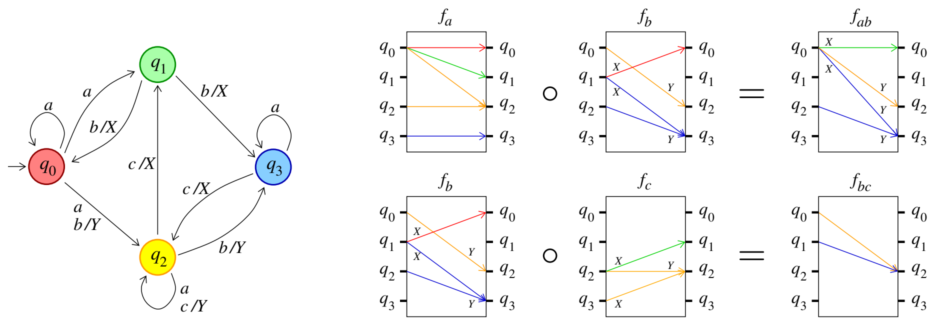

Fig. 2. VPA (left) and the TPs (right) from Example 4.

Definition 2. Let f and g be TPs. There are six different kinds of compositions, depending on the TPs' kind of f and g, which we define in the following. If f and g are both int-TPs, we define

$$(f \circ g)(q, q') \;:=\; \bigcurlyvee \big\{ f(q, q'') \sqcup g(q'', q') \,\big|\, q'' \in Q \big\} .$$

If f is an int-TP and g is either a call-TP or a ret-TP, we define

$$(f \circ g)(q, B, q') \;:=\; \bigcurlyvee \big\{ f(q, q'') \sqcup g(q'', B, q') \,\big|\, q'' \in Q \big\} \qquad \text{and}$$

$$(g \circ f)(q, B, q') \;:=\; \bigcurlyvee \big\{ g(q, B, q'') \sqcup f(q'', q') \,\big|\, q'' \in Q \big\} .$$

If f is a call-TP and g a ret-TP, we define

$$(f \circ g)(q, q') \;:=\; \bigcurlyvee \big\{ f(q, B, q'') \sqcup g(q'', B, q') \,\big|\, q'' \in Q \text{ and } B \in \Gamma \big\} .$$

Intuitively, the composition of two TPs f and g is obtained by following any edge through f from some state q to another state q'', then following any edge through g to some other state q'. The value of this path is the maximum of the two values encountered in f and g with respect to the priority ordering $\sqsubseteq$. Then one takes the maximum over all such possible values with respect to the reward ordering $\preceq$ and obtains a weighted path from q to q' in the composition.

We associate finite words with TPs as follows. With a letter $a \in \Sigma$ we associate the TP f_a as done in Def. 1. If the words $u, v \in \Sigma^+$ are associated with the TPs f and g, respectively, we associate the word uv with the TP $f \circ g$, provided that $f \circ g$ is defined. A word cannot be associated with two distinct TPs. This follows from the following lemma, which is easy to prove.

Lemma 3. *Let f, g, h, and k be TPs. If $(h \circ f) \circ (g \circ k)$ and $h \circ \big((f \circ g) \circ k\big)$ are both defined then $(h \circ f) \circ (g \circ k) = h \circ \big((f \circ g) \circ k\big)$.*

If the word $u \in \Sigma^+$ is associated with the TP f, we write f_u for f. Note that two distinct words can be associated with the same TP, i.e., it can be the case that $f_u = f_v$, for $u, v \in \Sigma^+$ with $u \neq v$. Intuitively, if this is the case then $\mathcal{A}$'s behavior on u is identical to $\mathcal{A}$'s behavior on v.

The following example illustrates TPs and their composition.

Example 4. Consider the VPA on the left in Fig. 2 with the states q_0, q_1, q_2, and q_3. The states' priorities are the same as their indices, and is also shown as a color: red (0), green (1), yellow (2), and blue (3) represent priorities. We assume that $\Sigma_{\mathsf{int}} = \{a\}$, $\Sigma_{\mathsf{call}} = \{b\}$, and $\Sigma_{\mathsf{ret}} = \{c\}$. The stack alphabet is $\Gamma = \{X, Y\}$. We can ignore the stack symbol $\bot$ since the VPA has no transitions for c and $\bot$.

Fig. 2 also depicts the TPs f_a, f_b, f_c and their compositions $f_a \circ f_b = f_{ab}$ and $f_b \circ f_c = f_{bc}$. The VPA's states are in-ports and out-ports of a TP. Assume that f is a call-TP. An in-port q is connected with an out-port q' if $f(q, B, q') \neq \dagger$, for some $B \in \Gamma$. Moreover, this connection of the two ports is labeled with the stack symbol B and the priority. Again, we use the colors to represent the priorities of the connections between the in-ports and the out-ports. For example, the connection in the TP f_a from the in-port q_0 to the out-port q_0 is red since $f_a(q_0, q_0) = 0$. Since f_a is an int-TP, connections are not labeled with stack symbols.

In a composition $f \circ g$, we plug f's out-ports with g's in-ports together. The priority from an in-port of $f \circ g$ to an out-port of $f \circ g$ is the maximum with respect to the priority ordering $\sqsubseteq$ of the priorities of the two connections in f and g. However, if f is a call-TP and g a ret-TP, we are only allowed to connect the ports in $f \circ g$, if the stack symbols of the connections in f and g match. Finally, since there can be more than one connection between ports in $f \circ g$, we take the maximum with respect to reward ordering $\preceq$.

We extend the composition operation $\circ$ to sets of TPs in the natural way, i.e., we define $F \circ G := \{f \circ g \mid f \in F \text{ and } g \in G \text{ for which } f \circ g \text{ is defined}\}$.

Definition 5. Define $\mathfrak{T}$ as the least solution to the equation

$$\mathfrak{T} \;=\; T_{\mathsf{int}} \;\cup\; T_{\mathsf{call}} \circ T_{\mathsf{ret}} \;\cup\; T_{\mathsf{call}} \circ \mathfrak{T} \circ T_{\mathsf{ret}} \;\cup\; \mathfrak{T} \circ \mathfrak{T}\,.$$

Note that the operations $\circ$ and $\cup$ are monotonic, and the underlying lattice of the powerset of all int-TPs is finite. Thus, the least solution always exists and can be found using fixpoint iteration in a finite number of steps.

The following lemma is helpful in proving that the elements of $\mathfrak{T}$ can be used to characterize (non-)universality of $\mathcal{A}$.

Lemma 6. *For every TP f, we have $f \in \mathfrak{T}$ only if there is a well-matched $w \in \Sigma^+$ with $f = f_w$.*

We need the following notions to characterize universality in terms of the existence of TPs with certain properties.

Definition 7. Let f be an int-TP.

(i) f is *idempotent* if $f \circ f = f$. Note that only an int-TP can be idempotent.

(ii) For $q \in Q$, we write $f(q)$ for the set of all $q' \in Q$ that are connected to q in this TP, i.e., $f(q) := \{q' \in Q \mid f(q, q') \neq \dagger\}$. Moreover, for $Q' \subseteq Q$, we define $f(Q') := \bigcup_{q \in Q'} f(q)$.

(iii) f is *bad* for the set $Q' \subseteq Q$ if $f(q, q)$ is either $\dagger$ or odd, for every $q \in f(Q')$. A *good* TP is a TP that is not bad. Note that any TP is bad for $\emptyset$. In the following, we consider bad TPs only in the context of idempotent TPs.

```
1 $N \leftarrow T_{\mathsf{int}} \cup T_{\mathsf{call}} \circ T_{\mathsf{ret}}$
2 $T \leftarrow N$
3 while $N \neq \emptyset$ do
4     forall $(f_u, f_v) \in N \times T \cup T \times N$ do
5         if $f_v$ idempotent and $f_v$ bad for $f_u(q_I)$ then
6             return universality does not hold, witnessed by $uv^\omega$
7     $N \leftarrow \big(N \circ T \cup T \circ N \cup T_{\mathsf{call}} \circ N \circ T_{\mathsf{ret}}\big) \setminus T$
8     $T \leftarrow T \cup N$
9 return universality holds
```

Fig. 3. Universality check UNIV for VPAs with respect to well-matched words.

Example 8. Reconsider the VPA from Example 4 and its TPs. It is easy to see that TP $g := f_a \circ f_a$ is idempotent. Since $g(q_2, q_2) = 2$, g is good for any $Q' \subseteq \{q_0, q_1, q_2, q_3\}$ with $q_2 \in Q'$. The intuition is that there is at least one run on $(aa)^\omega$ that starts in q_2 and loops infinitely often through q_2. Moreover, on this run 2 is the highest priority that occurs infinitely often. So, if there is a prefix $v \in \Sigma^+$ with a run that starts in the initial state and ends in q_2, we have that $v(aa)^\omega$ is accepted by the VPA. The TP g is bad for $\{q_1, q_3\}$, since $g(q_1, q_1) = \dagger$ and $g(q_3, q_3) = 3$. So, if there is prefix $v \in \Sigma^+$ for which all runs that start in the initial state and end in q_1 or q_3 then $v(aa)^\omega$ is not accepted by the VPA. Another TP that is idempodent is the TP $g' := f_b \circ \big((f_b \circ f_c) \circ f_c\big)$. Here, we have that $g'(q_1, q_1) = 2$ and $g'(q, q') = \dagger$, for all $q, q' \in \{q_0, q_1, q_2, q_3\}$ with not $q = q' = q_1$. Thus, g' is bad for every $Q' \subseteq Q$ with $q_1 \notin Q'$.

The following theorem characterizes universality of the VPA $\mathcal{A}$ in terms of the TPs that are contained in the least solution of the equation from Def. 5.

Theorem 9. *$L^\omega(\mathcal{A}) \neq NW^\omega(\Sigma)$ iff there are TPs $f, g \in \mathfrak{T}$ such that g is idempotent and bad for $f(q_I)$.*

Thm. 9 can be used to decide universality for VPAs with respect to the set of well-matched infinite words. The resulting algorithm, which we name UNIV, is depicted in Fig. 3. It computes $\mathfrak{T}$ by least-fixpoint iteration and checks at each stage whether two TPs exist that witness non-universality according to Thm. 9. The variable T stores the generated TPs and the variable N stores the newly generated TPs in an iteration. UNIV terminates if no new TPs are generated in an iteration. Termination is guaranteed since there are only finitely many TPs. For returning a witness of the VPA's non-universality, we assume that we have a word associated with a TP at hand. UNIV's asymptotic time complexity is as follows, where we assume that we use hash tables to represent T and N.

Theorem 10. *Assume that the given VPA $\mathcal{A}$ has $n \geq 1$ states, index $k \geq 2$, and $m = \max\{1, |\Sigma|, |\Gamma|\}$, where Σ is the VPA's input alphabet and Γ its stack alphabet. The running time of the algorithm UNIV is in $m^3 \cdot 2^{\mathcal{O}(n^2 \cdot \log k)}$.*

There are various ways to tune UNIV. For instance, we can store the TPs in a single hash table and store pointers to the newly generated TPs. Furthermore, we can store pointers to idempotent TPs. Another optimization also concerns the badness check in the line 4 to 6. Observe that it is sufficient to know the sets

$f_u(q_I)$, for $f_u \in T$, i.e, the sets $Q' \subseteq Q$ for which all runs for some well-matched word end in a state in Q'. We can maintain a set R to store this information. We initialize R with the singleton set $\big\{\big(\varepsilon, \{q_I\}\big)\big\}$. We update it after line 8 in each iteration by assigning the set $R \cup \big\{\big(uv, f_v(Q')\big) \,\big|\, (u, Q') \in R \text{ and } f_v \in T\big\}$ to it. After this update, we can optimize R by removing an element (u, Q') from it if there is another element (u', Q'') in R with $Q'' \subseteq Q'$. These optimizations do not improve UNIV's worst-case complexity but they are of great practical value.

4 Inclusion Checking

In this section, we describe how to check language inclusion for VPAs. For the sake of simplicity, we assume a single VPA and check for inclusion of the languages that are defined by two states q_I^1 and q_I^2. It should be clear that it is always possible to reduce the case for two VPAs to this one by forming the disjoint union of the two VPAs. Thus, for $i \in \{1, 2\}$, let $\mathcal{A}_i = (Q, \Gamma, \Sigma, \delta, q_I^i, \Omega)$ be the respective VPA. We describe how to check whether $L^\omega(\mathcal{A}_1) \subseteq L^\omega(\mathcal{A}_2)$ holds.

Transition profiles for inclusion checking extend those for universality checking. A *tagged transition profile* (TTP) of the int-type is an element of

$$\big(Q \times \Omega(Q) \times Q\big) \times \big(Q \times Q \to \Omega(Q)_\dagger\big)\,.$$

We write it as $f^{\langle p,c,p'\rangle}$ instead of (p, c, p', f) in order to emphasize the fact that the TP f is extended with a tuple of states and priorities. A call-TTP is of type

$$\big(Q \times \Gamma \times \Omega(Q) \times Q\big) \times \big(Q \times \Gamma \times Q \to \Omega(Q)_\dagger\big)$$

and a ret-TTP is of type

$$\big(Q \times \Omega(Q) \times \Gamma_\perp \times Q\big) \times \big(Q \times \Gamma \times Q \to \Omega(Q)_\dagger\big)\,.$$

Accordingly, they are written $f^{\langle p,B,c,p'\rangle}$ and $f^{\langle p,c,B,p'\rangle}$, respectively.

The intuition of an int-TTP $f^{\langle p,c,p'\rangle}$ is as follows. The TP f describes the essential information of *all* runs of the VPA $\mathcal{A}_2$ on a well-matched word $u \in \Sigma^+$. The attached information $\langle p, c, p'\rangle$ describes the existence of *some* run of the VPA $\mathcal{A}_1$ on u. This run starts in state p, ends in state p', and the maximal occurring priority on it is c. The intuition behind a call-TTP or a ret-TTP is similar. The symbol B in the annotation is the topmost stack symbol that is pushed or popped in the run of $\mathcal{A}_2$ for the pending position in the word u.

For $a \in \Sigma$, we now associate a set F_a of TTPs with the appropriate type. Recall that f_a stands for the TP associated to the letter a as defined in Def. 1.

- If $a \in \Sigma_{\mathsf{int}}$, let $F_a := \{f_a^{\langle p,\Omega(p'),p'\rangle} \mid p, p' \in Q \text{ and } p' \in \delta_{\mathsf{int}}(p, a)\}$.
- If $a \in \Sigma_{\mathsf{call}}$, let $F_a := \{f_a^{\langle p,B,\Omega(p'),p'\rangle} \mid p, p' \in Q,\ B \in \Gamma, \text{ and } (p', B) \in \delta_{\mathsf{call}}(p, a)\}$.
- If $a \in \Sigma_{\mathsf{ret}}$, let $F_a := \{f_a^{\langle p,\Omega(p'),B,p'\rangle} \mid p, p' \in Q,\ B \in \Gamma_\perp, \text{ and } p' \in \delta_{\mathsf{ret}}(p, B, a)\}$.

As with TPs, the composition of TTPs is only allowed in certain cases. They are the same as for TPs, e.g., the composition of a call-TTP with an int-TTP results in a call-TTP, and with a ret-TTP it results in an int-TTP. However, the composition of TTPs is not a monoid operation but behaves like the composition of morphisms in a category in which the states in Q, respectively pairs of states and stack symbols in Γ, act as objects. A TTP $f^{\langle p,c,p'\rangle}$ for instance can be seen as a morphism from p to p', and it can therefore only be composed with a morphism from p' to anything else.

The composition of two TTPs extends the composition of the underlying TPs by explaining how the tag of the resulting TTP is obtained. For int-TTPs $f^{\langle p,c,p'\rangle}$ and $g^{\langle p',c',p''\rangle}$, we define

$$f^{\langle p,c,p'\rangle} \circ g^{\langle p',c',p''\rangle} \;:=\; (f \circ g)^{\langle p,c\sqcup c',p''\rangle}\,.$$

Composing an int-TTP $f^{\langle p,c,p'\rangle}$ and a call-TTP $g^{\langle q,B,c',q'\rangle}$ yields call-TTPs:

$$f^{\langle p,c,p'\rangle} \circ g^{\langle q,B,c',q'\rangle} \;:=\; (f \circ g)^{\langle p,B,c\sqcup c',q'\rangle} \quad \text{if } p' = q$$

$$g^{\langle q,B,c',q'\rangle} \circ f^{\langle p,c,p'\rangle} \;:=\; (g \circ f)^{\langle q,B,c\sqcup c',p'\rangle} \quad \text{if } q' = p\,.$$

The two possible compositions of an int-TTP with a ret-TTP are defined in exactly the same way. Finally, the composition of a call-TTP $f^{\langle p,B,c,p'\rangle}$ and a ret-TTP $g^{\langle p',c',B,p''\rangle}$ is defined as

$$f^{\langle p,B,c,p'\rangle} \circ g^{\langle p',c',B,p''\rangle} \;:=\; (f \circ g)^{\langle p,c\sqcup c',p''\rangle}\,.$$

Note that the stack symbol B is the same in both annotations. As for sets of TPs, we extend the composition of TTPs to sets.

Similar to Def. 5, we define a set $\mathfrak{T}$ to be the least solution to the equation

$$\mathfrak{T} \;=\; T_{\mathsf{int}} \,\cup\, T_{\mathsf{call}} \circ T_{\mathsf{ret}} \,\cup\, T_{\mathsf{call}} \circ \mathfrak{T} \circ T_{\mathsf{ret}} \,\cup\, \mathfrak{T} \circ \mathfrak{T}\,,$$

where $T_\tau := \bigcup\{F_a \mid a \in \Sigma_\tau\}$, for $\tau \in \{\mathsf{int}, \mathsf{call}, \mathsf{ret}\}$. This allows us to characterize language inclusion between two VPAs in terms of the existence of certain TTPs.

Theorem 11. *$L^\omega(\mathcal{A}_1) \not\subseteq L^\omega(\mathcal{A}_2)$ iff there are TTPs $f^{\langle q_I^1,c,p\rangle}$ and $g^{\langle p,d,p\rangle}$ in $\mathfrak{T}$ fulfilling the following properties:*
(1) The priority d is even.
(2) The TP g is idempotent and bad for $f(q_I^2)$.

Thm. 11 yields an algorithm INCL to check $L^\omega(\mathcal{A}_1) \not\subseteq L^\omega(\mathcal{A}_2)$, for given VPAs $\mathcal{A}_1$ and $\mathcal{A}_2$. It is along the same lines as the algorithm UNIV and we omit it.[4] The essential difference lies in the sets T_{int}, T_{call} and T_{ret}, which contain TTPs instead of TPs, and the refined way in which they are being composed. Each iteration now searches for two TTPs that witness the existence of some word of the form uv^ω that is accepted by $\mathcal{A}_1$ but not accepted by $\mathcal{A}_2$. Similar optimizations that we sketch for UNIV at the end of Sect. 3 also apply to INCL.

For the complexity analysis of the algorithm INCL below, we do not assume that the VPAs $\mathcal{A}_1$ and $\mathcal{A}_2$ necessarily share the state set, the priority function, the stack alphabet, and the transition functions as assumed at the beginning of this subsection. Only the input alphabet Σ is the same for $\mathcal{A}_1$ and $\mathcal{A}_2$.

Theorem 12. *Assume that for $i \in \{1,2\}$, the number of states of the VPA $\mathcal{A}_i$ is $n_i \geq 1$, $k_i \geq 2$ its index, and $m_i = \max\{1, |\Sigma|, |\Gamma_i|\}$, where Σ is the VPA's input alphabet and Γ_i its stack alphabet. The running time of the algorithm* INCL *is in $n_1^4 \cdot k_1^2 \cdot m_1 \cdot m_2^3 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$.*

5 Evaluation

Our prototype tool FADecider[5] implements the presented algorithms in the programming language OCaml. To evaluate the tool's performance we carried out

[4] See Fig. 4 in App. H.
[5] The tool (version 1.1) is publicly available at `www2.tcs.ifi.lmu.de/fadecider`.

Tab. 1. Statistics on the input instances. The first row lists the number of states of the respective VPAs from an input instance and their alphabet sizes. The number of stack symbols of a VPA and its index are not listed, since in these examples the VPA's stack symbol set equals its state set and states are either accepting or non-accepting. The second row lists whether the inclusions $L^*(\mathcal{A}) \subseteq L^*(\mathcal{B})$ and $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$ of the respective VPAs hold.

	ex	*ex-§2.5*	*gzip*	*gzip-fix*	*png2ico*
size $\mathcal{A}$ / size $\mathcal{B}$ / alphabet size	9 / 5 / 4	10 / 5 / 5	51 / 71 / 4	51 / 73 / 4	22 / 26 / 5
language relation	$\subseteq$ / $\subseteq$	$\not\subseteq$ / $\subseteq$	$\not\subseteq$ / ?	$\subseteq$ / $\subseteq$	$\subseteq$ / $\subseteq$

Tab. 2. Experimental results for the language-inclusion checks. The row "FADecider" lists the running times for the tool FADecider for checking $L^*(\mathcal{A}) \subseteq L^*(\mathcal{B})$ and $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$ The row "#TTPs" lists the number of encountered TTPs. The symbol ‡ indicates that FADecider ran out of time (2 hours). The row "OpenNWA" lists the running times for the implementation based on the OpenNWA library for checking inclusion on finite words and the VPA's size obtained by complementing $\mathcal{B}$.

	ex	*ex-§2.5*	*gzip*	*gzip-fix*	*png2ico*
FADecider	0.00s / 0.00s	0.00s / 0.00s	36s / ‡	42s / 294s	0.10s / 0.11s
#TTPs	6 / 6	18 / 19	694 / ‡	518 / 1,117	586 / 609
OpenNWA	0.16s / 27	0.04s / 11	49s / 27	1,104s / 176	74.70s / 543

the following experiments for which we used a 64-bit Linux machine with 4 GB of main memory and two dual-core Xeon 5110 CPUs, each with 1.6 GHz. Our benchmark suite consists of VPAs from [11], which are extracted from real-world recursive imperative programs. Tab. 1 describes the instances, each consisting of two VPAs $\mathcal{A}$ and $\mathcal{B}$, in more detail. Tab. 2 shows FADecider's running times for the inclusion checks $L^*(\mathcal{A}) \subseteq L^*(\mathcal{B})$ and $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$.[6] For comparison, we used the OpenNWA library [12]. The inclusion check there is implemented by a reduction to an emptiness check via a complementation construction. Note that OpenNWA does not support infinite nested words at all and has no direct support for only considering well-matched nested words. We used therefore OpenNWA to perform the language-inclusion checks with respect to all finite nested words.

FADecider outperforms OpenNWA, on most examples by magnitudes. Profiling the inclusion check based on the OpenNWA library yields that complementation requires about 90% of the overall running time. FADecider spends about 90% of its time on composing TPs and about 5% on checking equality of TPs. The experiments also show that FADecider's performance on inclusion checks for infinite words can be worse than the for finite words. Note that checking inclusion for infinite-word languages is more expensive than for finite-word languages, since, in addition to reachability, one needs to account for loops.

6 Conclusion

Checking universality and language inclusion for automata by avoiding determinization and complementation has recently attracted a lot of attention, see, e.g., [1, 9, 10, 13, 15]. We have shown that Ramsey-based methods for Büchi automata generalize to the richer automaton model of VPAs with a parity acceptance condition. Another competitive approach based on antichains has recently

[6] Results of additional conducted experiments are given in App. I.

also been extended to VPAs, however, only over finite words [6]. It remains to be seen if optimizations for the Ramsey-based algorithms for Büchi automata [1] extend, with similar speed-ups, to this richer setting. Another direction of future work is to investigate Ramsey-based approaches for automaton models that extend VPAs like multi-stack VPAs [17].

Acknowledgments. We are grateful to Evan Driscoll for providing us with VPAs.

References

1. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR'11*, LNCS 6901, pp. 187–202.
2. P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *TACAS'10*, LNCS 6015, pp. 158–174.
3. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Progr. Lang. Syst.*, 27(4):786–818, 2005.
4. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):1–43, 2009.
5. S. Breuers, C. Löding, and J. Olschewski. Improved Ramsey-based Büchi complementation. In *FOSSACS'12*, LNCS 7213, pp. 150–164.
6. V. Bruyère, M. Ducobu, and O. Gauwin. Visibly pushdown automata: universality and inclusion via antichains. In *LATA'13*. To appear.
7. J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. of the 1960 Internat. Congr. on Logic, Method, and Philosophy of Science*, pp. 1–11.
8. C. Dax, M. Hofmann, and M. Lange. A proof system for the linear time μ-calculus. In *FSTTCS'06*, LNCS 4337, pp. 273–284.
9. M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV'06*, LNCS 4144, pp. 17–30.
10. L. Doyen and J.-F. Raskin. Antichains for the automata-based approach to model-checking. *Log. Methods Comput. Sci.*, 5(1), 2009.
11. E. Driscoll, A. Burton, and T. Reps. Checking conformance of a producer and a consumer. In *ESEC/FSE'11*, pp. 113–123.
12. E. Driscoll, A. Thakur, and T. Reps. OpenNWA: A nested-word-automaton library. In *CAV'12*, LNCS 7358, pp. 665–671.
13. S. Fogarty and M. Y. Vardi. Büchi complementation and size-change termination. In *TACAS'09*, LNCS 5505, pp. 16–30.
14. S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS'10*, LNCS 6015, pp. 205–220.
15. O. Friedmann and M. Lange. Ramsey-based analysis of parity automata. In *TACAS'12*, LNCS 7214, pp. 64–78.
16. M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL'10*, pp. 471–482.
17. S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pp. 161–170.
18. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *POPL'01*, pp. 81–92.
19. F. P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30:264–286, 1928.

20. M.-H. Tsai, S. Fogarty, M. Y. Vardi, and Y.-K. Tsay. State of Büchi complementation. In *CIAA'10*, LNCS 6482, pp. 261–271.
21. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS'86*, pp. 332–344.

A Additional Proof Details for Theorem 9

Before launching into the proof of Thm. 9, we need the following two simple lemmas about the orderings $\preceq$ and $\sqsubseteq$.

The intuition about the reward ordering $\preceq$ given in Sect. 2 is reflected in the following lemma. It shows how the reward ordering $\preceq$ can be used to select a "most accepting" run of a VPA on an infinite word. Its proof is straightforward and therefore omitted. Its consequence is more important: Suppose there are two runs on the same word such that the priorities of the first one are always at most as high as the corresponding ones in the second run with respect to $\preceq$. Then the second run is accepting if the first one is.

Lemma 13. *Let $\rho, \rho' \in C^\omega$, where $C \subseteq \mathbb{N}$ is some finite set of priorities. Suppose that $\rho_i \preceq \rho'_i$, for all $i \in \mathbb{N}$. Then $\bigsqcup \inf(\rho) \preceq \bigsqcup \inf(\rho')$.*

The following lemma states that using the priority ordering $\sqsubseteq$ one can contract an infinite run of a VPA whilst preserving the maximal priority occurring infinitely often. Its proof is also straightforward and omitted.

Lemma 14. *Let $\rho \in C^\omega$, where $C \subseteq \mathbb{N}$ is some finite set of priorities. Take any strictly increasing sequence $i_0 < i_1 < \dots$ of natural numbers and consider $\rho' \in C^\omega$ with $\rho'_j := \bigsqcup\{\rho_{i_j}, \dots, \rho_{i_{j+1}-1}\}$, for $j \in \mathbb{N}$. We have that $\bigsqcup \inf(\rho) = \bigsqcup \inf(\rho')$.*

In the remainder of this section, we prove the equivalence stated in Thm. 9.

"$\Leftarrow$" Let f and g be TPs in $\mathfrak{T}$ with g idempotent and bad for $f(q_I)$. By Lem. 6, there are $u, v \in \Sigma^+$ such that $f = f_u$ and $g = f_v$ and both u and v contain no pending positions. Then uv^ω contains no pending positions either. It remains to be seen that $uv^\omega \notin L^\omega(\mathcal{A})$.

For the sake of contradiction assume that $w := uv^\omega \in L(\mathcal{A})$. Thus, there is an accepting run $\rho = (q_0, \gamma_0)(q_1, \gamma_1) \dots$ of $\mathcal{A}$ on w such that $q_0 = q_I$.

It must be the case that $f(q_0, q_{|u|}) \neq \dagger$, since there is a possibility to reach $q_{|u|}$ from $q_0 = q_I$, which is witnessed by the initial run fragment $(q_0, \gamma_0) \dots (q_{|u|}, \gamma_{|u|})$. Hence, $q_{|u|} \in f(q_0)$. Similarly, we conclude that $q_{|uv|} \in g(q_{|u|})$. This can be iterated to show that $g(q_{|u|+i|v|}, q_{|u|+(i+1)|v|}) \neq \dagger$, for all $i \geq 1$. Since Q is finite, there is a state $q \in Q$ such that $q = q_{|u|+i|v|}$, for infinitely many i. Assume that $i_0 < i_1 < \dots$ is such a sequence of indices. Define a sequence $c_0, c_1, \dots$ by $c_j := \bigsqcup\{\Omega(q_{i_j}), \dots, \Omega(q_{i_{j+1}-1})\}$, for $j \in \mathbb{N}$. According to Lem. 14, we have $\bigsqcup \inf_{j\to\infty} c_j = \bigsqcup \inf_{i\to\infty}(\Omega(q_i))$. Note that i_0 can be chosen large enough such that $\bigsqcup \inf_{j\to\infty} c_j = \bigsqcup_{i=j}^{\infty} c_j$. Let c be this value.

Since g is idempotent we have $g^{i_{j+1}-i_j} = g$, for every $j \in \mathbb{N}$ and therefore $c_j \preceq g(q,q)$, for every $j \in \mathbb{N}$. According to Lem. 13, $g(q,q)$ would have to be even, too, which contradicts the assumption that g is bad.

"⇒" Suppose that $w = a_0a_1a_2 \ldots \in NW^\omega(\Sigma) \setminus L^\omega(\mathcal{A})$. Note that w does not contain any pending positions by assumption. We inductively define an infinite sequence $i_0 < i_1 < \ldots$ of natural numbers with $i_0 := 0$ and for every $j \geq 0$, i_{j+1} is defined as follows:

– If $a_{i_j} \in \Sigma_{\mathsf{int}}$ then $i_{j+1} := i_j + 1$.

– If $a_{i_j} \in \Sigma_{\mathsf{call}}$ and a_h for $h > i_j$ is its matching return position then $i_{j+1} := h+1$.

Note that the position h above always exists since w does not contain any pending call positions. Furthermore, note that $a_{i_j} \in \Sigma_{\mathsf{ret}}$ is impossible. This follows from the assumption that w does not contain any pending return positions and can be proved easily by contradiction, assuming that i_j is the smallest position with $a_{i_j} \in \Sigma_{\mathsf{ret}}$. Finally, we remark that for every $j \geq 0$, the finite word $a_{i_j} \ldots a_{i_{j+1}-1}$ does not contain pending positions. In other words, the sequence $i_0 < i_1 < \ldots$ splits the infinite nested word w at the top level into infinitely many finite nested words $a_{i_0} \ldots a_{i_1-1}$, $a_{i_1} \ldots a_{i_2-1}$, ..., which are all well-matched.

Let $I := \{i_j \mid j \in \mathbb{N}\}$ and $I^{(2)} := \{(i_j, i_{j'}) \mid j, j' \in \mathbb{N} \text{ with } j < j'\}$, and consider the coloring $\chi : I^{(2)} \to \mathfrak{T}$ defined as $\chi(i, i') := f_{a_i \ldots a_{i'-1}}$. Note that χ is well-defined, since $a_i \ldots a_{i'-1}$ is a well-matched nested word, for $(i, i') \in I^{(2)}$, and hence $f_{a_i \ldots a_{i'-1}} \in \mathfrak{T}$. Furthermore, note that $\mathfrak{T}$ is finite. By Ramsey's theorem [19], there is an infinite subset J of I and a TP $g \in \mathfrak{T}$ such that $\chi(j, j') = g$, for all $j, j' \in J$ with $j < j'$. Without loss of generality, we assume that $J = \{j_0, j_1, \ldots\}$ with $0 < j_0 < j_1 < \ldots$. Furthermore, we define int-TP $f := \chi(0, j_0)$. Recall that $i_0 = 0$.

We first observe that g is idempotent because we have

$$g \circ g = \chi(j_0, j_1) \circ \chi(j_1, j_2) = \chi(j_0, j_2) = g\,.$$

Note that the composition of $\chi(j_0, j_1)$ and $\chi(j_1, j_2)$ is defined since g is an int-TP.

It remains to be seen that g is bad for $f(q_I)$. Suppose it is not. Then there is some $q' \in f(q_I)$ and some $q \in g(q')$ such that $g(q, q) = c$, for some even c. Let $c' := f(q_I, q')$, $c'' := g(q', q)$, $u := a_0 \ldots a_{j_1-1}$ and $v_i := a_{j_i} \ldots a_{j_{i+1}-1}$, for every $i \geq 1$. Note that $w = uv_1v_2v_3\ldots$.

We construct an accepting run of $\mathcal{A}$ on w as follows. We start with an infinite sequence

$$(q_I, \bot) \xrightarrow{u,c'}_f (q', \bot) \xrightarrow{v_1,c''}_g (q, \bot) \xrightarrow{v_2,c}_g (q, \bot) \xrightarrow{v_3,c}_g \ldots$$

Intuitively, the first arrow states that there is a run of $\mathcal{A}$ on u contained in f that leads from the configuration $(q_I, \bot)$ to the configuration $(q', \bot)$ and sees c' as the maximal priority on this part. The other arrows are interpreted similarly. Note that the maximal priority occurring infinitely often in this sequence is, clearly, $c = g(q, q)$.

Next we turn this sequence into a run preserving a local and a global invariant. Globally, c remains the maximal priority occurring infinitely often in these sequences. Locally, on any part $(q, \gamma) \xrightarrow{v,c}_h (q', \gamma')$ in this sequence we have: $h = f_v$, and if $|v| = 1$ then there is a transition that transforms the configuration (q, γ) into (q', γ') whilst reading v; and if $|v| > 1$ then h can be decomposed in of three ways, according to Def. 5.

1. $v = ab$ for some $a \in \Sigma_{\mathsf{call}}$ and $b \in \mathsf{ret}$, and $h = f_a \circ f_b$.

2. $v = aub$ for some $a \in \Sigma_{\mathsf{call}}$, some well-matched $u \in \Sigma^+$ and $b \in \Sigma_{\mathsf{ret}}$, and $h = f_a \circ f_u \circ f_b$.
3. $v = v'v''$ for some well-matched $v', v'' \in \Sigma^+$, and $h = f_{v'} \circ f_{v''}$.

Here we only consider case (2). The other cases are handled in the same way and are actually simpler. So suppose that $v = aub$ and $h = f_a \circ f_u \circ f_b$. Then there must be $p, p' \in Q$, $B \in \Gamma$ and $h' \in \mathfrak{T}$ such that $h(q, q') = c' \sqcup c'' \sqcup c'''$ where $c' = f_a(q, B, p)$, $c'' = h'(p, p')$ and $c''' = f_b(p', B, q')$. Then replace $(q, \gamma) \xrightarrow{v,c}_h (q', \gamma')$ in this sequence with

$$(q, \gamma) \xrightarrow{a,c'} (p, B\gamma) \xrightarrow{u,c''}_{h'} (p', B\gamma) \xrightarrow{b,c'''} (q', \gamma)\,.$$

It is not hard to see that these transformation steps can be iterated independently of each other, and the result is of the form

$$(q_I, \bot) \xrightarrow{w_0,c_0} (q_1, \gamma_1) \xrightarrow{w_1,c_1} (q_2, \gamma_2) \xrightarrow{w_2,c_2} \dots$$

According to the local invariant, it forms a run of $\mathcal{A}$ on w. According to the global invariant we have $\bigsqcup \inf_{i \to \infty} c_i = c$. Now remember that above c was assumed to be even. Hence, this is an accepting run of $\mathcal{A}$ on w which contradicts the assumption that $w \in NW^\omega(\Sigma) \setminus L^\omega(\mathcal{A})$.

B Additional Proof Details for Theorem 10

We assume the following time complexities of the following operations. Checking whether two int-TPs are equal is in $\mathcal{O}(n^2)$. Note that for int-TPs f and g, we need to check for all tuples $(q, q') \in Q \times Q$, if the equality $f(q, q') = g(q, q')$ holds. It follows that adding an int-TP to T or N costs $\mathcal{O}(n^2)$ time, since we need to compute the TP's hash value and make a lookup if the TP is already stored in the table. TP composition is carried out in $\mathcal{O}(n^3 \cdot m)$ time. Checking whether an int-TP is idempotent is in $\mathcal{O}(n^3)$ and checking for badness in $\mathcal{O}(n)$.

We observe that the number of int-TPs is bounded by $(k+1)^{n^2}$. Thus, N and T never store more than $(k+1)^{n^2}$ elements. It is easy to see that an int-TP is stored at most once in N at the beginning of the while loop starting at line 3 of the algorithm. It follows that the lines 4 to 6 of the algorithm are executed at most once for a pair of int-TPs. In summary, the lines 4 to 6 take at most $2^{\mathcal{O}(n^2 \cdot \log k)}$ time.

It remains to analyze the time complexity of updating N and T in line 7 and line 8 of the algorithm. The number of carried out composition operations in an iteration is bounded by $\mathcal{O}(|N| \cdot |T| + |T_{\mathsf{call}}| \cdot |N| \cdot |T_{\mathsf{ret}}|)$. Since each int-TP appears at most once in N, the number composition operations in total is bounded by $\mathcal{O}(|T|^2 + |T| \cdot |\Sigma|^2)$. Note that $|T_{\mathsf{call}}|, |T_{\mathsf{ret}}| \leq |\Sigma|$. Since $|T| \leq (k+1)^{n^2}$ and the $\mathcal{O}(n^3 \cdot m)$ time complexity of TP composition, it follows that line 7 (without removing the elements that are also in T) takes in total at most $m^3 \cdot 2^{\mathcal{O}(n^2 \cdot \log k)}$ time. Removing the elements that are also in T in line 7 and T's update in line 8 take in one iteration at most $2^{\mathcal{O}(n^2 \cdot \log k)}$ time. Since the algorithm never removes elements from T, the number of iterations of the algorithm is bounded by $2^{\mathcal{O}(n^2 \cdot \log k)}$.

Overall, we obtain the time complexity $m^3 \cdot 2^{\mathcal{O}(n^2 \cdot \log k)}$.

C Relation to Determinization and Complementation

In this section, we derive complementation constructions for VPAs from the machinery developed in Sect. 3. We start with a complementation construction for VPAs over finite well-matched words and extend it afterwards to infinite well-matched words.

With the machinery developed in Sect. 3, a complementation construction is straightforward. For a VPA $\mathcal{A} = (Q, \Gamma, \Sigma, \delta, q_I, \Omega)$, we define the VPA $\mathcal{C}$ as the tuple $(Q', \Gamma'_\perp, \Sigma, \delta', q'_I, \Omega')$, where its components are as follows:
- $Q' := \{f \mid f \text{ int-TP}\}$,
- $\Gamma' := \{f \mid f \text{ call-TP}\}$,
- for $f, g \in Q$' and $a \in \Sigma$, the transitions are $\delta'_{\mathsf{int}}(f, a) := \big\{f \circ f_a\big\}$ if $a \in \Sigma_{\mathsf{int}}$, $\delta'_{\mathsf{call}}(f, a) := \big\{(q'_I, f \circ f_a)\big\}$ if $a \in \Sigma_{\mathsf{call}}$, and $\delta'_{\mathsf{ret}}(f, g, a) := \big\{(g \circ f) \circ f_a\big\}$ and $\delta'_{\mathsf{ret}}(f, \perp, a) := \emptyset$ if $a \in \Sigma_{\mathsf{ret}}$,
- $q'_I(q, q) := 0$ and $q'_I(q, q') := \dagger$, for $q, q' \in Q$ with $q \neq q'$, and
- for $f \in Q'$, we define $\Omega'(f) := 1$ if $f(q_I, q) \neq \dagger$ and $\Omega(q)$ is even, for some $q \in Q$, and $\Omega'(f) := 0$, otherwise.

The value of $\delta'_{\mathsf{ret}}(f, \perp, a)$ is actually irrelevant, since we assume inputs to be well-matched words.

Note that $\mathcal{C}$ is deterministic. This construction is similar to Alur and Madhusudan's determinization construction for nested-word automata over finite inputs [4], with some minor differences.[7] One difference is due to our objective to obtain a VPA that accepts the complement of $\mathcal{A}$'s language. This is reflected in the definition of the priority function Ω'. The state set Q' and the stack alphabet Γ' are also different from Alur and Madhusudan's construction. We use sets of TPs and Alur and Madhusudan use the sets $2^{Q \times Q}$ and $2^{Q \times Q} \times \Sigma_{\mathsf{call}}$, respectively. Elements from these sets represent similar information about the runs of $\mathcal{A}$. Finally, Alur and Madhusudan's determinization construction deals with pending positions in inputs and produces VPAs with slightly fewer states.

Proposition 15. *$L^*(\mathcal{C}) = NW^*(\Sigma) \setminus L^*(\mathcal{A})$.*

The detailed proof proceeds along the lines of the one in [4]. Here we give some intuition about the correctness of this construction. The VPA $\mathcal{C}$ uses the int-TPs to keep track of the essential behavior of all of $\mathcal{A}$'s runs on the input processed so far. In addition to the classical subset construction, an int-TP f stores the information about the existence of a run from a state q to a state q', i.e., $f(q, q') \neq \dagger$. This information is needed when returning from a call, where the information f about $\mathcal{A}$'s runs on the subword is combined with (1) the information g about $\mathcal{A}$'s runs on the prefix up to the matching call position and (2) the information f_a about $\mathcal{A}$'s runs on the current letter $a \in \Sigma_{\mathsf{ret}}$. $\mathcal{C}$ has pushed the call-TP g on its stack when reading the letter at the corresponding call position. Now, it pops it from the stack and puts the information of the different parts correctly together, i.e., $\mathcal{C}$'s new state h after reading the return

[7] The determinization construction of the printed version of the article is flawed. The error has been corrected, see `www.cis.upenn.edu/~alur/Jacm09.pdf`.

letter a is the composition of the TPs $h := (g \circ f) \circ f_a$. Note that this composition is always defined, since g is a call-TP, f an int-TP, and f_a a ret-TP. Furthermore, note that composing a TP with the int-TP q'_I does not alter the TP.

In the following, we sketch a complementation construction for VPAs on infinite well-matched words. In fact, our construction is a direct extension of the Ramsey-based complementation construction for Büchi automata, see, e.g., [5,7]. It is different from the one given by Alur and Madhusudan in [4]. There, the complementation construction is split into three construction steps. One automaton flattens the hierarchical structure of the inputs by transforming nested words into so-called pseudo-runs. Another automaton reads such pseudo-runs and decides whether to accept or reject the input. The final construction step combines both automata to yield an automaton that accepts the complemented language.

Here we construct a VPA $\mathcal{C}'$ for the complement of $L^\omega(\mathcal{A})$ directly. It consists of several, slightly modified, copies of the VPA $\mathcal{C}$ defined above. One component $\mathcal{C}_*$ takes care of finite prefixes of the inputs. The initial state of $\mathcal{C}'$ is the initial state of $\mathcal{C}_*$. All states in $\mathcal{C}_*$ have the odd priority 1. For each int-TP g, we have another modified copy $\mathcal{C}_g$ of $\mathcal{C}$. This component $\mathcal{C}_g$ takes care of infinite suffixes of the inputs on which $\mathcal{A}$'s runs are looping with respect to the TP g. Without loss of generality, we assume that $\mathcal{C}_g$'s initial state has only outgoing transitions. $\mathcal{C}_g$'s initial state has the even priority 2, all its other states have the odd priority 1. In terms of the Büchi acceptance condition, all states of $\mathcal{C}'$ are rejecting except the initial states of the components $\mathcal{C}_g$ are accepting. Furthermore, we add an ε-transition from $\mathcal{C}_g$'s state g to its initial state. The VPA $\mathcal{C}_*$ is connected to the other VPAs as follows. For each pair (f, g) of int-TPs with $f \circ g = f$, and g idempotent and bad for $f(q_I)$, we connect the state f in $\mathcal{C}_*$ with $\mathcal{C}_g$'s initial state by an ε-transition. Note that ε-transitions can be eliminated in the standard way.

Proposition 16. *$L^\omega(\mathcal{C}') = NW^\omega(\Sigma) \setminus L^\omega(\mathcal{A})$.*

The proof proceeds along the lines of the proof of Büchi's complementation construction and uses similar arguments as in the proof of Thm. 9. In particular, showing that the language on the right-hand side is a subset of the language on the left-hand side relies on Ramsey's theorem. Details are omitted.

We conclude this section by commenting on the differences and similarities of the complementation construction and algorithm UNIV for checking universality of a given VPA. TPs are the basic building blocks of the complementation construction above and they are also at the core of UNIV. This is actually not surprising, since for both problems, one needs to investigate all runs on any input. TPs are an appropriate entity for this purpose. However, the search space for UNIV is more concise and explored with less overhead. The complementation construction involves more book-keeping. In order to build the complement automaton we must determine and store the transitions between its states, which are essentially int-TPs. First, we need to store multiple copies of an int-TP (or pointers to it) for the states in the different copies of the VPA $\mathcal{C}$. Similarly, a call-TP might occur as a stack symbol in several transitions for call and return letters. Second, in the complementation construction we keep track of how exactly a state corresponding to an int-TP f is reachable, which might be different for well-matched words $u, v \in \Sigma^+$ with $f_u = f_v = f$ but $u \neq v$. In contrast, the universality check UNIV only stores the TPs and iteratively composes them. Finally, in the complementation construction we only combine TPs f with atomic TPs f_a, for determining the successor states of states for the letters $a \in \Sigma$. The universality check UNIV constructs the TPs less stringently in the sense that in each iteration already constructed TPs f_u and f_v, with $u, v \in \Sigma^+$, are composed whenever their composition $f_u \circ f_v$ is defined.

D Nested Words with Pending Positions

Let $\sharp \in \{*, \omega\}$. In addition to $NW^\sharp(\Sigma)$, we consider here the three sets of finite words and four corresponding sets of infinite words.

– $NW^\sharp_{\mathsf{call}}(\Sigma)$ is the set of words in $\Sigma^\sharp$ that may contain pending call positions but must not contain pending return positions.
– $NW^\sharp_{\mathsf{ret}}(\Sigma)$ is the set of words in $\Sigma^\sharp$ that may contain pending return positions but must not contain pending call positions.
– $NW^\sharp_{\mathsf{any}}(\Sigma)$ is the set of words in $\Sigma^\sharp$ that may contain pending call positions and pending return positions.

Note that $NW^\sharp_{\mathsf{any}}(\Sigma) = \Sigma^\sharp$. We also call the words in $NW^\sharp(\Sigma)$ *well-matched.*

For program-verification purposes, the four sets $NW^\sharp(\Sigma)$ and $NW^\sharp_{\mathsf{call}}(\Sigma)$ with $\sharp \in \{*, \omega\}$ are certainly of most interest. For instance, $NW^\sharp(\Sigma)$ can be used to describe traces of recursive imperative programs in which every call eventually terminates and there is a top-most procedure which runs forever, when $\sharp = \omega$. Similarly, the set $NW^\omega_{\mathsf{call}}(\Sigma)$ can be used to describe program traces in which subprocedures may not terminate. The sets $NW^\sharp_{\mathsf{ret}}(\Sigma)$ and $NW^\sharp_{\mathsf{any}}(\Sigma)$ are included here because they are not any more difficult to handle, and $NW^\sharp_{\mathsf{any}}(\Sigma)$ may well be useful in specifications about correct call-and-return behavior, i.e., when one wants to *assert* rather than *assume* that no return is possible without a corresponding call beforehand. Furthermore, the call-return dualism need not only be used to describe recursive imperative programs but also programs using data structures like stacks or lists. In that case, a pending return position may correspond to a faulty access to the data structure, and it may therefore well be reasonable to allow pending returns in such specifications.

For a VPA $\mathcal{A}$, we define the additional languages

$$L^\sharp_t(\mathcal{A}) := \{w \in NW^\sharp_t(\Sigma) \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}\,,$$

where $\sharp \in \{*, \omega\}$ and $t \in \{\mathsf{call}, \mathsf{ret}, \mathsf{any}\}$.

E Extended Universality Check

For extending our universality check UNIV to account for infinite words that are not well-matched, we introduce a new operation on TPs, the so-called *collapse* operation $(\cdot){\downarrow}$. It turns call-TPs and ret-TPs into int-TPs. For a call-TP f, we define

$$f{\downarrow}(q, q') \;:=\; \bigvee \bigl\{ f(q, B, q') \bigm| B \in \Gamma \bigr\}$$

and

$$f{\downarrow}(q,q') \;:=\; f(q,\bot,q')\,,$$

when f is a ret-TP. For int-TPs, $(\cdot){\downarrow}$ is the identity. Thus, for call-TPs, the collapse operation ignores the stack symbols and chooses the best that is available with respect to the reward ordering. For ret-TPs, the collapse operation takes the value for $\bot$, which occurs at the top of the stack when reading a symbol in Σ_{ret} iff the position is pending. The collapse operation extends in the natural way to sets, i.e., $F{\downarrow} := \{f{\downarrow} \mid f \in F\}$.

Definition 17. In addition to the sets $\mathfrak{T}$ we define the sets $\mathfrak{T}^{\mathsf{call}*}$ and $\mathfrak{T}^{\mathsf{ret}*}$ of TPs as the least solution of the following system of equations.

$$\begin{aligned}
\mathfrak{T} &= T_{\mathsf{int}} \,\cup\, T_{\mathsf{call}} \circ T_{\mathsf{ret}} \,\cup\, T_{\mathsf{call}} \circ \mathfrak{T} \circ T_{\mathsf{ret}} \,\cup\, \mathfrak{T} \circ \mathfrak{T} \\
\mathfrak{T}^{\mathsf{call}*} &= T_{\mathsf{call}}{\downarrow} \,\cup\, \mathfrak{T} \,\cup\, \mathfrak{T}^{\mathsf{call}*} \circ \mathfrak{T}^{\mathsf{call}*} \\
\mathfrak{T}^{\mathsf{ret}} &= T_{\mathsf{ret}}{\downarrow} \,\cup\, \mathfrak{T} \,\cup\, \mathfrak{T}^{\mathsf{ret}*} \circ \mathfrak{T}^{\mathsf{ret}*}
\end{aligned}$$

The set $\mathfrak{T}$ is the same as before. $\mathfrak{T}^{\mathsf{call}*}$ and $\mathfrak{T}^{\mathsf{ret}*}$ subsume $\mathfrak{T}$. Additionally, they contain the collapsed atomic call-TPs and ret-TPs, respectively, and they are closed under the composition operation $\circ$. The intuition is the following. $\mathfrak{T}^{\mathsf{call}*}$ contains TPs that describe runs like TPs from $\mathfrak{T}$. In particular, the stack content is ignored. The positions where the runs of the TPs in $\mathfrak{T}^{\mathsf{call}*}$ are connected by the composition are pending call positions. Although the VPA pushes at these positions in each run a symbol on the stack, it never pops these symbols later. Thus, the actual symbols are irrelevant. The intuition for $\mathfrak{T}^{\mathsf{ret}*}$ is similar. Here, the positions are pending return positions and the stack symbol is always $\bot$.

The following theorem characterizes (non-)universality of the VPA $\mathcal{A}$ with respect to the sets $NW^{\omega}_{\mathsf{call}}(\Sigma)$, $NW^{\omega}_{\mathsf{ret}}(\Sigma)$, and $NW^{\omega}_{\mathsf{any}}(\Sigma)$. Its proof proceeds very much along the same lines as the proof of Thm. 9 and is therefore omitted.

Theorem 18. *(a)* $L^{\omega}_{\mathsf{call}}(\mathcal{A}) \neq NW^{\omega}_{\mathsf{call}}(\Sigma)$ *iff there are TPs* $f,g \in \mathfrak{T}^{\mathsf{call}*}$ *such that* g *is idempotent and bad for* $f(q_I)$*.*

(b) $L^{\omega}_{\mathsf{ret}}(\mathcal{A}) \neq NW^{\omega}_{\mathsf{ret}}(\Sigma)$ *iff there are TPs* $f,g \in \mathfrak{T}^{\mathsf{ret}*}$ *such that* g *is idempotent and bad for* $f(q_I)$*.*

(c) $L^{\omega}_{\mathsf{any}}(\mathcal{A}) \neq NW^{\omega}_{\mathsf{any}}(\Sigma)$ *iff there are TPs* $f,g \in \mathfrak{T}^{\mathsf{ret}*}$ *or* $f \in \mathfrak{T}^{\mathsf{call}*} \cup \mathfrak{T}^{\mathsf{ret}*}$ *and* $g \in \mathfrak{T}^{\mathsf{call}*}$ *such that* g *is idempotent and bad for* $f(q_I)$*.*

Note that a word in $NW^{\omega}_{\mathsf{any}}(\Sigma)$ might contain pending call and pending return positions. However, all pending return positions must occur before the pending call positions. In this case, the pending positions must occur in a finite prefix of the infinite word.

With Def. 17 and Thm. 18 it is straightforward to adapt algorithm UNIV from Fig. 3 so that it checks universality with respect to the sets $NW^{\omega}_{\mathsf{call}}(\Sigma)$, $NW^{\omega}_{\mathsf{ret}}(\Sigma)$, and $NW^{\omega}_{\mathsf{any}}(\Sigma)$. The asymptotic time complexity does not alter.

F Universality Check for Finite Nested Words

Checking universality of the VPA $\mathcal{A}$ with respect to finite words is now straightforward. Without loss of generality, we assume that $\varepsilon \in L^*(\mathcal{A})$. This special case can be checked separately.

Theorem 19. *$L^*(\mathcal{A}) \neq NW^*(\Sigma)$ iff there is a TP $f \in \mathfrak{T}$ such that $f(q_I, q) = \dagger$ or $\Omega(q)$ is odd, for all states $q \in Q$.*

The characterizations of $\mathcal{A}$'s (non-)universality with respect to the other sets $NW^*_{\mathsf{call}}(\Sigma)$, $NW^*_{\mathsf{ret}}(\Sigma)$, and $NW^*_{\mathsf{any}}(\Sigma)$ can easily be derived in a similar manner.

The algorithmic realization is a straightforward adaption of algorithm UNIV in Fig. 3. Note that further optimizations are possible. For instance, the TPs do not need to keep track of the maximal occurred priorities in the runs that they represents. The resulting asymptotic complexity is $m^3 \cdot 2^{\mathcal{O}(n^2)}$, where n and m are as in Thm. 10. The number of iterations is bounded by $2^{\mathcal{O}(n^2)}$.

G Additional Proof Details for Theorem 11

"$\Leftarrow$" Suppose there are TTPs $f^{\langle q_I^1, c, p\rangle}$ and $g^{\langle p, d, p\rangle}$ with the properties (1) and (2). Assume that $f = f_u$ and $g = f_v$, for some well-matched $u, v \in \Sigma^+$. It is easy to see that there is a run $(q_0^1, \gamma_0^1)(q_1^1, \gamma_1^1)\dots$ of $\mathcal{A}_1$ on uv^ω with $(q_0^1, \gamma_0^1) = (q_I^1, \bot)$ and $(q^1_{|u|+i|v|}, \gamma^1_{|u|+i|v|}) = (p, \bot)$, for all $i \in \mathbb{N}$. In particular, the stack content is $\bot$ after reading uv^i since u and v are well-matched. Furthermore, $d = \bigsqcup\{\Omega(q) \mid q \in \inf(q_0^1 q_1^1 \dots)\}$. It follows that $uv^\omega \in L^\omega(\mathcal{A}_1)$. The fact that $uv^\omega \notin L^\omega(\mathcal{A}_2)$ is a simple consequence of Thm. 9. Note that property (2) above is exactly the condition that is sufficient for $\mathcal{A}_2$ not to accept this uv^ω according to that theorem.

"$\Rightarrow$" Suppose there is a well-matched word $w = a_0 a_1 \dots \in L^\omega(\mathcal{A}_1) \cap NW^\omega(\Sigma) \setminus L^\omega(\mathcal{A}_2)$. Let $(q_0^1, \gamma_0^1)(q_1^1, \gamma_1^1)\dots$ be an accepting run of $\mathcal{A}_1$ on w. Thus, there is an even priority d and a $j_0 \in \mathbb{N}$ such that d is the maximal priority occurring infinitely often in this run, and no greater priority occurs after position j_0. Let c be the maximal priority occurring before position j_0.

As in the proof of Thm. 9, we define an infinite sequence $i_0 < i_1 < \dots$ with $i_0 = 0$ such that $a_{i_j} \dots a_{i_{j+1}-1}$ is a well-matched word, for each $j \in \mathbb{N}$. However, we additionally require $i_1 \geq j_0$. Now, consider the coloring $\chi(i_j, i_{j'}) := f_v$ with $v = a_{i_j} \dots a_{i_{j'}-1}$. As in the proof of the direction from left to right of Thm. 9, Ramsey's theorem yields TPs f' and g' in $\mathfrak{T}$ such that g' is idempotent and bad for $f'(q_I^2)$. By the pidgeon hole principle, there must be some $j, j' \in \mathbb{N}$ such that $j < j'$ and $q_{i_j} = q_{i_{j'}} = p$ for some $p \in Q$. Define TPs

$$f := f' \circ \underbrace{g' \circ \dots \circ g'}_{j \text{ times}} \quad \text{and} \quad g := \underbrace{g' \circ \dots \circ g'}_{j'-j \text{ times}} .$$

It is not hard to see, that g is also idempotent because g' is, and it is bad for $f(q_I^2)$ because g' is bad for $f'(q_I^2)$. The atomic TPs that compose f and g can now be tagged with single transitions of $\mathcal{A}_1$'s accepting run such that their compositions become the TTPs $f^{\langle q_I^1, c, p\rangle}$ and $g^{\langle p, d, p\rangle}$, which finishes the proof.

```
1 $N \leftarrow T_{\mathsf{int}} \ \cup \ T_{\mathsf{call}} \circ T_{\mathsf{ret}}$
2 $T \leftarrow N$
3 while $N \neq \emptyset$ do
4     forall $(f_u^{\langle p,c,p' \rangle}, f_v^{\langle q,d,q' \rangle}) \in N \times T \ \cup \ T \times N$ do
5         if $p = q_I^1$, $p' = q = q'$, $d$ even, $f_v$ idempotent, and $f_v$ bad for $f_u(q_I^2)$ then
6             return inclusion does not hold, witnessed by $uv^\omega$
7     $N \leftarrow \big(N \circ T \ \cup \ T \circ N \ \cup \ T_{\mathsf{call}} \circ N \circ T_{\mathsf{ret}}\big) \setminus T$
8     $T \leftarrow T \cup N$
9 return inclusion holds
```

Fig. 4. Inclusion check INCL for VPAs with respect to well-matched words.

H Additional Proof Details for Theorem 12

The algorithm INCL for checking inclusion is given in Fig. 4. An extension to check inclusion to nested words with pending positions are along the same lines as the corresponding extensions of the universality check from App. E. We omit the details.

In the remainder of this section, we analyze the complexity of INCL. We observe that there are at most $n_1^2 \cdot k_1 \cdot (k_2 + 1)^{n_2^2}$ int-TTPs. Similar to the algorithm UNIV, the total time of the check in the lines 4 to 6 of the algorithm INCL is dominated by the number of int-TTP pairs, which is bounded by $n_1^4 \cdot k_1^2 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$.

For the update N in line 7, we need to compose TTPs in N with TTPs in T, T_{call}, and T_{ret}. Note that a TTP consist of a TP and information about $\mathcal{A}_1$'s behavior. A requirement for composing TTPs is that their information on $\mathcal{A}_1$'s behavior fits together. For instance, the composition of the int-TTPs $f^{\langle p,c,p' \rangle}$ and $g^{\langle q,d,q' \rangle}$ is only defined when $p' = q$. We can reduce the number TTP compositions by grouping TTPs in T, N, T_{call}, and T_{ret} with respect to their information about $\mathcal{A}$'s behavior. For example, when int-TTPs $f^{\langle p,c,p' \rangle}$ and $g^{\langle q,d,q' \rangle}$ are both in T, we group them together in case $p = q$ and $p' = q'$. It follows that the total number of TTP compositions is bounded by $n_1^4 \cdot k_1 \cdot m_1 \cdot m_2^2 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$. Note that each int-TTP in T is at most once in N and since the states determine the priority in the information about $\mathcal{A}_1$'s behavior in the TTPs in T_{call} and T_{ret} we have that $|T_{\mathsf{call}}|, |T_{\mathsf{ret}}| \in \mathcal{O}(n_1^2 \cdot m_1 \cdot |\Sigma|)$. Since equality between two int-TTPs can be checked in $\mathcal{O}(n_2^2)$ time and TTP composition can be carried out in $\mathcal{O}(n_2^3 \cdot m_2)$ time, the updates of N (without removing elements that are also in T) take $n_1^4 \cdot k_1 \cdot m_1 \cdot m_2^3 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$ time in total. Removing the elements that are also in T in line 8 and updating T in line 8 take $n_1^2 \cdot k_1 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$ time in one iteration and $n_1^4 \cdot k_1^2 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$ in total, since the number of iterations is bounded $n_1^2 \cdot k_1 \cdot 2^{\mathcal{O}(n_2^2 \cdot \log k_2)}$

By putting these upper bounds together, we obtain the claimed upper bound on the time complexity of the algorithm INCL.

Tab. 3. Experimental results for the language-inclusion checks with respect to the sets $NW^*_{\mathsf{any}}(\Sigma)$ and $NW^\omega_{\mathsf{any}}(\Sigma)$. The row "FADecider" lists the running times for the tool FADecider for checking $L^*_{\mathsf{any}}(\mathcal{A}) \subseteq L^*_{\mathsf{any}}(\mathcal{B})$ and $L^\omega_{\mathsf{any}}(\mathcal{A}) \subseteq L^\omega_{\mathsf{any}}(\mathcal{B})$. The row "#TTPs" lists the number of encountered TTPs. For comparison, the row "OpenNWA" lists again the running times for the implementation based on the OpenNWA library for checking $L^*_{\mathsf{any}}(\mathcal{A}) \subseteq L^*_{\mathsf{any}}(\mathcal{B})$ and the VPA's size obtained by complementing $\mathcal{B}$.

	ex	*ex-§2.5*	*gzip*	*gzip-fix*	*png2ico*
FADecider	0.00s / 0.00s	0.00s / 0.00s	36s / ‡	230s / 2,134s	0.10s / 0.12s
#TTPs	17 / 17	18 / 66	694 / ‡	9,178 / 21,731	586 / 609
OpenNWA	0.16s / 27	0.04s / 11	49s / 27	1,104s / 176	74.70s / 543

Tab. 4. Experimental results for the language-inclusion checks of the VPAs $\mathcal{A}$ and $\mathcal{B}$ with themselves. The row NW^* lists the running times for the checks $L^*(\mathcal{A}) \subseteq L^*(\mathcal{A})$ and $L^*(\mathcal{B}) \subseteq L^*(\mathcal{B})$ performed by the tool FADecider, and the encountered number of TTPs. The rows NW^ω, NW^*_{any}, and NW^ω_{any} are similar. The rows "OpenNWA" and "determinization" list the running times for the checks $L^*_{\mathsf{any}}(\mathcal{A}) \subseteq L^*_{\mathsf{any}}(\mathcal{A})$ and $L^*_{\mathsf{any}}(\mathcal{B}) \subseteq L^*_{\mathsf{any}}(\mathcal{B})$ performed by the implementation based on the OpenNWA library, and the size of the respective VPA resulting from determinization.

		ex	*ex-§2.5*	*gzip*	*gzip-fix*	*png2ico*
NW^*	FADecider	0.00s / 0.00s	0.00s / 0.00s	0.14s / ‡	0.14s / ‡	0.07s / 1.12s
	# TTPs	6 / 22	13 / 61	704 / ‡	704 / ‡	519 / 1,173
NW^ω	FADecider	0.00s / 0.00s	0.00s / 0.00s	0.14s / ‡	0.14s / ‡	0.08s / 1.14s
	# TTPs	6 / 22	13 / 67	704 / ‡	704 / ‡	542 / 1,223
NW^*_{any}	FADecider	0.00s / 0.01s	0.00s / 0.00s	5.52s / ‡	5.52s / ‡	0.07s / 1.14s
	# TTPs	17 / 159	40 / 61	6,167 / ‡	6,167 / ‡	519 / 1,173
NW^ω_{any}	FADecider	0.00s / 0.01s	0.00s / 0.00s	5.52s / ‡	5.52s / ‡	0.08s / 1.14s
	# TTPs	17 / 171	40 / 67	6,167 / ‡	6,167 / ‡	542 / 1,223
	OpenNWA	0.04s / 1.11s	0.03s / 0.21s	0.98s / †	0.98s / †	0.08s / †
	determinization	9 / 135	10 / 55	174 / †	174 / †	22 / †

I Additional Experiments

In this section, we report on additional experiments for evaluating the performance of FADecider, in particular, when checking language inclusion on all nested words instead of restricting oneselves to well-matched nested words. Tab. 3 shows the results of these exeriments. FADecider still outperforms OpenNWA. However, for the instance *gzip-fix*, the FADecider's running time and the number of encounterted TTPs increase significantly. In Tab. 4, we list the results of further experiments of checking language inclusion. Recall that timeouts (2 hours) are indicated by the symbol ‡. The symbol † indicates when a tool ran out of memory (4 GB).