Form Methods Syst Des (2018) 53:339–362
https://doi.org/10.1007/s10703-017-0282-y

Realizability of concurrent recursive programs

Benedikt Bollig[1] · Manuela-Lidia Grindei[1] · Peter Habermehl[2]

Published online: 24 July 2017

Abstract We study the realizability problem for concurrent recursive programs: given a distributed system architecture and a sequential specification over words, find a distributed automata implementation that is equivalent to the specification. This problem is well-studied as far as finite-state processes are concerned, and it has a solution in terms of Zielonka's Theorem. We lift Zielonka's Theorem to the case where processes are recursive and modeled as visibly pushdown (or, equivalently, nested-word) automata. However, contrarily to the finite-state case, it is undecidable whether a specification is realizable or not. Therefore, we also consider suitable underapproximation techniques from the literature developed for multi-pushdown systems, and we show that they lead to a realizability framework with effective algorithms.

Keywords Concurrent recursive programs · Realizability · Asynchronous automata · Nested-word automata · Mazurkiewicz traces · Zielonka's theorem · Monadic second-order logic

1 Introduction

The realizability problem arises when we are given a specification that we would like to transform into an implementation in terms of an automaton model. So, the first question to

✉ Benedikt Bollig
bollig@lsv.fr

Manuela-Lidia Grindei
manuela.grindei@ens-cachan.org

Peter Habermehl
peter.habermehl@liafa.univ-paris-diderot.fr

1 CNRS, LSV, ENS Paris-Saclay, 61, avenue du Président Wilson, 94235 Cachan Cedex, France

2 Université Paris Diderot (Paris 7), IRIF, Bâtiment Sophie Germain, Bureau 4017, 8, place FM/13, 75013 Paris, France

ask is if there is such an implementation at all. Famous instances of that problem are the Kleene Theorem and the Büchi–Elgot–Trakhtenbrot Theorem saying that, roughly speaking, all regular specifications are realizable. In Fig. 1, three equivalent specifications are given (from top to bottom: an LTL formula, a regular expression, a first-order formula) as well as an implementation in terms of a finite automaton.

While, as far as regular languages and their various representations are concerned, any specification has an implementation, the situation is more difficult when we move to a *concurrent* setting. Let us look at a well-studied model of concurrent shared-memory systems with finite-state processes, which we later refer to as Zielonka automata [46]. Suppose that we have two processes, 1 and 2, where 1 executes action a, and 2 executes action b. Consider Fig. 2. Is it justified to say that the concurrent automaton on the right-hand side, with no communication between processes 1 and 2, is an implementation of the regular expression $\alpha = ab(ab)^*$? There are (at least) two arguments against it. Expression α implies that any execution performs as many a's as b's. But this is not realizable under the architecture that we consider (recall that 1 and 2 do not communicate). Moreover, b's have to be preceded by a's, which is not realizable for the same reason. Note that these problems are inherent to the specifications and will not vanish with a more clever implementation. So, what is the "right" formalism for the specification of such systems? Or, put differently, what are the specifications that, unlike $ab(ab)^*$, are realizable?

Consider Fig. 3, and suppose that the system provides a third action, call it c, that is executed simultaneously by both processes. The concurrent automaton on the right-hand side might then be seen as an implementation of the regular expression on the left. First, the action c allows both processes to synchronize after each a-step (b-step, respectively), so that, in particular, there are as many a's as b's in any execution. Second, the order of

$$G(a \rightarrow Fb)$$
$$((b+c)^*(\varepsilon + a(a+c)^*b))^*$$
$$\forall x.(a(x) \rightarrow \exists y.(x \leq y \wedge b(y)))$$

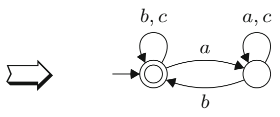

Fig. 1 Realizability of sequential finite-state systems

$$ab(ab)^*$$

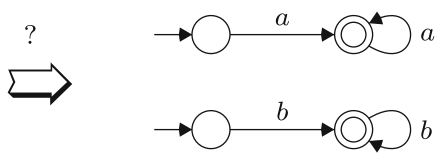

Fig. 2 (Non-)realizability of concurrent finite-state systems

$$((ab+ba)c)^*$$

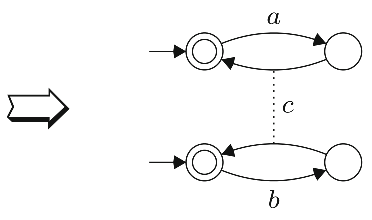

Fig. 3 Realizability of concurrent finite-state systems

executing a and b is not fixed anymore by the specification. Indeed, a classical result of Mazurkiewicz trace theory due to Zielonka states that regular specifications that are *closed* under permutation of such *independent* actions can be effectively translated into a concurrent automaton [46]. Note that the specification from Fig. 2 is actually not closed in that sense (it contains ab, but not ba), while the specification from Fig. 3 is closed. Similar results were obtained in the static message-passing setting [11,17,21,22] for finite-state processes communicating via (existentially or universally) bounded FIFO channels.

In this paper, we obtain a generalization of Zielonka's theorem for concurrent *recursive* (as opposed to finite-state) programs. A system is recursive if a process may suspend its activity and invoke (or, call) a subtask, before proceeding (or, returning). A stack is used to store the current configuration of the process, and to recover it once the subtask has been completed. The programs we consider are *boolean*, i.e., possible variables range over a finite domain. This implies that the set of states and the alphabet of stack symbols are finite. Boolean programs, in turn, may arise as abstractions from programs with infinite-domain variables. Such an extension is interesting from a theoretical but also from a practical point of view. Zielonka's theorem allows for automatic implementations of sequential specifications such as mutual exclusion properties, which are easy to specify in a sequential, linearization-based language, but hard to transfer to a distributed setting. These properties are, of course, not restricted to finite-state processes but may involve recursive procedures. It would be interesting to extend the case-studies from [42] in that respect.

In fact, recursive processes may be defined in several equivalent ways. The classical definition is in terms of a pushdown automaton with explicit stacks. A run is then a sequence of configurations that keep track of the current stack contents. Alternatively, automata may run on words enriched with nesting relations [5], one for each process. A nesting relation connects a call position with the corresponding return position. In that case, a transition of the automaton that performs a return will depend on the state/stack symbol associated with the corresponding call position. It is then sufficient to define a run as a sequence of states, without explicitly mentioning the stack contents. This is the approach adopted in this paper. Note that automata over words (or partial orders) that include one or several nesting relations are equivalent to visibly pushdown automata (cf. again [5]) where each action is associated with a unique stack operation. Generally, nested-word automata/visibly pushdown automata are more robust than classical pushdown automata (they are complementable and closed under intersection), and our results crucially rely on this robustness.

Since the realizability question amounts to asking whether a specification can be translated into an automaton at all, we would like to distinguish, algorithmically, between the specifications from Fig. 2 (not realizable) and Fig. 3 (realizable). The question is decidable for finite-state systems [37,40], but, unfortunately, it is undecidable in the recursive case, which follows from the undecidability of the nonemptiness problem. We will, therefore, identify sufficient decidable criteria that still guarantee realizability.

A fruitful approach to recover decidability in the realm of verification has been to restrict the possible system executions, for example by imposing a bound on the number of *context switches*, a notion introduced in [41]. There, each context allows only one dedicated process to perform calls or returns. This amounts to *underapproximating* the complete system behavior. However, as calls and returns from different processes are possibly independent, the final implementation may still exhibit executions that do not fit into this restriction anymore. The notion of context switches has been relaxed in several orthogonal ways. *Phase-bounded* systems only restrict the number of switches between returns from different processes [25]. *Scope-bounded* systems, on the other hand, restrict the number of contexts that separate a call and its return [28]. Finally, *ordered* systems impose an ordering on the processes and allow only the first process with a pending call to perform a return [6]. All these underapproximations render basic verification and model-checking questions decidable, though with varying complexity. Interestingly, it was later shown that they all induce classes of graphs that have bounded tree-width, which yields a unifying proof technique [31,36]. Relying on the corresponding decidability results, we show that realizability becomes decidable for context- and phase-bounded systems.

We then look at monadic second-order (MSO) logic, a specification formalism that does not distinguish between equivalent *linearizations* (like *ab* and *ba*) but is interpreted directly on the partial order induced by a system execution. The partial order induced by *ab* with *a* and *b* independent has two incomparable elements with labels *a* and *b*, respectively. Though an MSO specification does not necessarily guarantee realizability, it already rules out design errors that can a priori be avoided. Under the above-mentioned restrictions, we provide results in the spirit of the Büchi–Elgot–Trakhtenbrot Theorem, showing an effective equivalence between automata and MSO logic. This actually generalizes a result by Thomas for finite-state processes [44].

Outline The paper is organized as follows: Sect. 2 introduces nested-trace automata (NTAs), our model of concurrent recursive programs. Section 3 focuses on sequential specifications in terms of nested-word languages, and it presents the corresponding automata model of nested-word automata (NWAs). In Sect. 4, we consider realizability, i.e., the task of synthesizing an NTA from a given NWA specification. In doing so, we extend Zielonka's Theorem to concurrent recursive programs. Section 5 first reviews known results on NWAs based on restrictions that render basic decision problems decidable and that will then be used to obtain *decidable* criteria for realizability. In Sect. 6, we provide a logical characterization of NTAs in terms of MSO logic. We conclude with Sect. 7, in which we suggest several directions for future work.

This is a revised and extended version of the FOSSACS'09 paper [9]. The presentation, however, is quite different, presenting the framework in terms of *nested* words and traces rather than classical words. Note that both views are, in a sense, equivalent, and proofs are not immediately affected. However, the current presentation simplifies some definitions and corresponds to nested-word automata as presented in the standard reference [5]. Since structures exhibit explicit nesting relations, it is also closer to the MSO characterization presented in Sect. 6. Finally, we extended some results to context-bounded behaviors, and some others to scope-bounded and ordered structures.

2 Nested traces and their automata

In this section, we present our model of a concurrent recursive program, where a fixed finite set of processes communicate via shared memory. As a single process is recursive and involves function calls and returns, its executions are not just words, but rather *nested* words [5]. A nested word extends a word over a finite alphabet by a nesting relation, connecting function calls with their respective returns. An execution of the whole program, which we call a nested *trace*, involves several processes. It can, thus, be described as a collection of nested words. Some of the word positions are "shared" by several nested words, which models the shared memory. Note that nested traces combine nested words and Mazurkiewicz traces [15,33] in a straightforward manner. An example nested trace is depicted in Fig. 4.

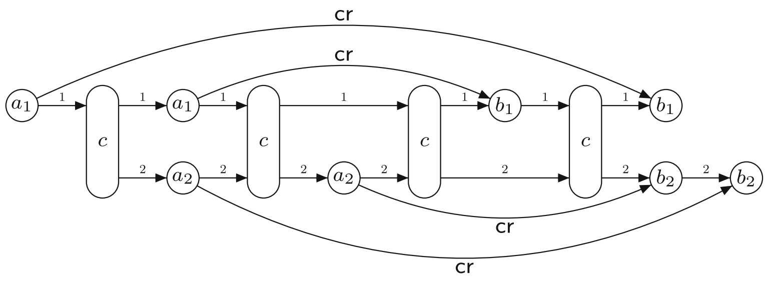

Fig. 4 Nested trace T

Before we can define nested traces formally, we will have to fix an architecture, which determines the actions and processes of a system, the type of an action (call, return, or internal), and the attribution of an action to one or more processes. Fixing the type of an action implies that the nesting relations are uniquely determined by the underlying strings.

Definition 1 (*Distributed call-return alphabet*) A *distributed call-return alphabet* is a tuple $\tilde{\Sigma} = (\Sigma, P, type, dom)$ where

- Σ is the nonempty finite set of *actions*,
- P is the nonempty finite set of *processes*,
- $type : \Sigma \to \{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$ indicates the *type* of an action (for a given type $\tau \in \{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$, we let $\Sigma_\tau \overset{\text{def}}{=} type^{-1}(\tau)$), and
- $dom : \Sigma \to 2^P$ associates with every action its *domain*, i.e., the set of processes that are involved in its execution.

We require that $dom(a) \neq \emptyset$ for all $a \in \Sigma$, and $|dom(a)| = 1$ for all $a \in \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$. The latter condition will imply that synchronization is achieved via internal actions only.

When $\Sigma = \Sigma_{\mathsf{int}}$, then we say *distributed alphabet* instead of distributed call-return alphabet.

Definition 2 (*Nested trace*) A *nested trace* over the distributed call-return alphabet $\tilde{\Sigma}$ is a tuple $(E, (\lhd_p)_{p\in P}, \lhd_{\mathsf{cr}}, \lambda)$ where

1. E is a finite set of *events*,
2. $\lambda : E \to \Sigma$ is the *event-labeling function* (given $\tau \in \{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$ and $p \in P$, we let $E_\tau \overset{\text{def}}{=} \{e \in E \mid \lambda(e) \in \Sigma_\tau\}$ and $E_p \overset{\text{def}}{=} \{e \in E \mid p \in dom(\lambda(e))\}$),
3. for all $p \in P$, $\lhd_p \subseteq E_p \times E_p$ is the direct-sucessor relation of some (unique) total order on E_p, which we denote by $\leqslant_p$ (with strict part $<_p$), and
4. $\lhd_{\mathsf{cr}} \subseteq E \times E$ is the *call-return relation* satisfying the following:

   (a) $\lhd_{\mathsf{cr}}$ induces a bijection between E_{call} and E_{ret},
   (b) for all $(e, f) \in \lhd_{\mathsf{cr}}$, there is $p \in P$ such that $e <_p f$,
   (c) for all $(e_1, f_1), (e_2, f_2) \in \lhd_{\mathsf{cr}}$ and $p \in P$ such that $e_1 \in E_p$, $e_2 \in E_p$, and $e_1 <_p e_2 <_p f_1$, we have $f_2 <_p f_1$.

In addition, we require that $\leq \overset{\text{def}}{=} (\lhd_{\mathsf{cr}} \cup \bigcup_{p\in P} \lhd_p)^*$ is a partial order.

Condition 4(a) implies that there are no pending calls or returns.[1] Condition 4(b) says that a call-return pair belongs to a unique process and that a call takes always place before its return. By Condition 4(c), the call-return relation of each process is well-nested.

[1] In [5], pending calls or returns are possible. We can also include them without affecting any of the results, but do not do so to simplify the presentation.

The set of nested traces over $\tilde{\Sigma}$ is denoted by $\mathsf{NTr}(\tilde{\Sigma})$. We do not distinguish between isomorphic nested traces.

Example 1 Consider the distributed call-return alphabet $\tilde{\Sigma} = (\Sigma, P, \mathit{type}, \mathit{dom})$ given by $\Sigma = \{a_1, b_1, a_2, b_2, c\}$, $P = \{1, 2\}$, $\mathit{type}(a_1) = \mathit{type}(a_2) = \mathsf{call}$, $\mathit{type}(b_1) = \mathit{type}(b_2) = \mathsf{ret}$, and $\mathit{type}(c) = \mathsf{int}$, as well as $\mathit{dom}(a_1) = \mathit{dom}(b_1) = \{1\}$, $\mathit{dom}(a_2) = \mathit{dom}(b_2) = \{2\}$, and $\mathit{dom}(c) = \{1, 2\}$. A nested trace over $\tilde{\Sigma}$ is depicted in Fig. 4. Curved edges represent the relation $\lhd_{\mathsf{cr}}$. Moreover, straight edges with label $p \in P$ represent the process-successor relation $\lhd_p$.

Next, we define a distributed automata model running on nested traces, which is a mix of nested-word automata and Zielonka automata. Here, the term *distributed* reflects the fact that every process has its own local state space and transition relation. Actually, there is a transition relation for every action $a \in \Sigma$. A transition involving a can access the local state of *any* process from $\mathit{dom}(a)$, and modify it. Therefore, since $a \in \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$ implies $|\mathit{dom}(a)| = 1$, executing a call or return action updates the local state of only one single component.

Definition 3 (*NTA*) A *nested-trace automaton (NTA)* over $\tilde{\Sigma}$ is a tuple $\mathcal{C} = ((S_p)_{p\in P}, \Gamma, \Delta, \iota, F)$ where

- the S_p are disjoint finite sets of *local states* (for $P' \subseteq P$, we define $S_{P'} \overset{\text{def}}{=} \prod_{p\in P'} S_p$),
- Γ is the nonempty finite set of *stack symbols*,
- $\iota \in S_P$ is the *global initial state*,
- $F \subseteq S_P$ is the set of *global final states*, and
- $\Delta = \Delta_{\mathsf{call}} \uplus \Delta_{\mathsf{ret}} \uplus \Delta_{\mathsf{int}}$ is the *transition relation*, partitioned into
  - $\Delta_{\mathsf{call}} \subseteq \bigcup_{p\in P}(S_p \times \Sigma_{\mathsf{call}} \times \Gamma \times S_p)$,
  - $\Delta_{\mathsf{ret}} \subseteq \bigcup_{p\in P}(S_p \times \Sigma_{\mathsf{ret}} \times \Gamma \times S_p)$, and
  - $\Delta_{\mathsf{int}} \subseteq \bigcup_{a\in\Sigma_{\mathsf{int}}}(S_{\mathit{dom}(a)} \times \{a\} \times S_{\mathit{dom}(a)})$.

Let $\mathbb{S} = \bigcup_{P'\subseteq P} S_{P'}$. For $s \in \mathbb{S}$ and $p \in P$, we let s_p be the p-th component of s (if it exists).

Suppose the automaton is about to read an event e of a nested trace with label $\lambda(e) = a$. If $a \in \Sigma_{\mathsf{int}}$, a transition of the form $(s, a, s') \in \Delta_{\mathsf{int}}$ lets process p move on from s_p to s'_p, for all $p \in \mathit{dom}(a)$. Only one process is involved when $a \in \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$ and a transition of the form $(s, a, A, s') \in \Delta_{\mathsf{call}} \cup \Delta_{\mathsf{ret}}$ is applied. In addition, if $a \in \Sigma_{\mathsf{call}}$, the call event e will be tagged with stack symbol A. The latter can be retrieved at the corresponding return position. More precisely, we require that, whenever $e \lhd_{\mathsf{cr}} f$, the stack symbol chosen at e is the same as the stack symbol employed by the transition that is taken at position f. In a sense, this is equivalent to reading a stack symbol previously pushed onto a stack.

A *run* of the NTA $\mathcal{C}$ on a nested trace $T = (E, (\lhd_p)_{p\in P}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NTr}(\tilde{\Sigma})$ will include a mapping $\rho : E \to \mathbb{S}$ such that $\rho(e) \in S_{\mathit{dom}(\lambda(e))}$ for all $e \in E$. Intuitively, for process $p \in \mathit{dom}(\lambda(e))$, the component $\rho(e)_p$ is the state that p reaches *after* executing e. Before we specify when ρ is actually part of a run, let us define another mapping $\rho^- : E \to \mathbb{S}$ that also satisfies $\rho^-(e) \in S_{\mathit{dom}(\lambda(e))}$ for all $e \in E$. Here, the intuition is that $\rho^-(e)$ collects the current source states of all processes that are involved in executing e. We let $\rho^-(e) = (s_p)_{p\,\in\,\mathit{dom}(\lambda(e))}$ where

$$s_p = \begin{cases} \iota_p & \text{if } e \text{ is } \lhd_p\text{-minimal} \\ \rho(f)_p & \text{if } f \lhd_p e\,. \end{cases}$$

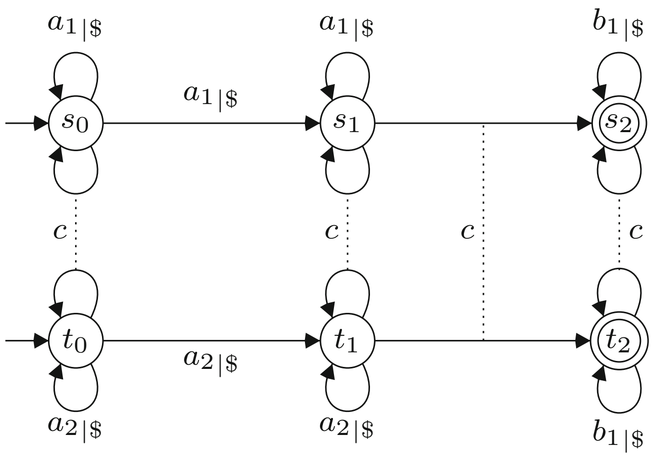

Fig. 5 Nested-trace automaton C

Recall that, hereby, $\rho(f)_p$ denotes the p-th component of $\rho(f) \in S_{dom(\lambda(f))}$. This component indeed exists since $p \in dom(\lambda(f))$. Moreover, e is called $\lhd_p$-minimal ($\lhd_p$-maximal) if there is no $e' \in E_p$ such that $e' \lhd_p e$ (respectively, $e \lhd_p e'$). Now, we call the pair (ρ, σ), with $\sigma : E_{\mathsf{call}} \cup E_{\mathsf{ret}} \to \Gamma$, a *run* of $\mathcal{C}$ on T if,

- for all $(e, f) \in \lhd_{\mathsf{cr}}$, we have $\sigma(e) = \sigma(f)$, and
- for all $e \in E$, it holds[2]

$$\begin{cases} \big(\rho^-(e)_p, \lambda(e), \sigma(e), \rho(e)_p\big) \in \Delta_{\mathsf{call}} & \text{if } e \in E_{\mathsf{call}} \cap E_p \\ \big(\rho^-(e)_p, \lambda(e), \sigma(e), \rho(e)_p\big) \in \Delta_{\mathsf{ret}} & \text{if } e \in E_{\mathsf{ret}} \cap E_p \\ \big(\rho^-(e), \lambda(e), \rho(e)\big) \in \Delta_{\mathsf{int}} & \text{if } e \in E_{\mathsf{int}}\,. \end{cases}$$

To determine if run (ρ, σ) is accepting, we look at the global final state reached at the end of a run. It collects, for all $p \in P$, the local state associated with the $\lhd_p$-maximal event, or ι_p if $E_p = \emptyset$. Formally, let $f = (f_p)_{p \in P} \in S_P$ be given by

$$f_p = \begin{cases} \rho(e)_p & \text{if } e \text{ is } \lhd_p \text{-maximal} \\ \iota_p & \text{if } E_p = \emptyset\,. \end{cases}$$

With this, we call (ρ, σ) *accepting* if $f \in F$. Finally, we denote by $L(\mathcal{C})$ the set of nested traces over $\tilde{\Sigma}$ that have an accepting run of $\mathcal{C}$.

Example 2 Again, we assume the distributed call-return alphabet $\tilde{\Sigma}$ from Example 1, with $P = \{1, 2\}$. Consider the NTA $\mathsf{C} = ((S_p)_{p \in P}, \{\$\}, \Delta, \iota, F)$ over $\tilde{\Sigma}$ depicted in Fig. 5. Its components are given by $S_1 = \{s_0, s_1, s_2\}$, $S_2 = \{t_0, t_1, t_2\}$, $\iota = (s_0, t_0)$, $F = \{(s_2, t_2)\}$. The new feature compared to sequential automata are the synchronizing transitions from Δ_{int}, which include $((s_i, t_i), c, (s_i, t_i))$ for all $i \in \{0, 1, 2\}$ as well as $((s_1, t_1), c, (s_2, t_2))$. Note that C is eventually forced to execute "concurrent" occurrences of a_1 and a_2: when one process moves on to s_1 or t_1, then executing c is no longer possible unless the other process catches

[2] Here, we rather use $\rho(e)_p$ than $\rho(e)$, since, strictly speaking, the latter is a *tuple* consisting of one state, whereas Δ_{call} and Δ_{ret} refer to *states*.

up. The nested trace T from Fig. 4 is accepted by C. Note that there are indeed concurrent occurrences of a_1 and a_2.

A special case of NTAs is given in the case of a distributed alphabet, i.e., when $\Sigma = \Sigma_{\mathsf{int}}$. Then, an NTA is precisely an *asynchronous automaton* (also *Zielonka automaton*) [46], and a nested trace is actually a *Mazurkiewicz trace*. However, following our terminology, we rather call it a *trace automaton*. Since, in that case, all actions from Σ have type int, the language of a trace automaton may indeed be seen as a set of Mazurkiewicz traces.

3 Multiply nested words and their automata

An NTA can be seen as a model of a concurrent system, with physically distributed processes, each of which has a local state space. On the specification side, on the other hand, it is natural to allow for some global view of the system, which facilitates the specification of what the system is supposed to do as a whole. This leads us to define *nested-word automata*. As opposed to NTAs, nested-word automata have one single state space, albeit preserving several stacks. Their behavior can be described as a set of *nested words* with multiple nesting relations. In fact, a nested word can be defined as a total order, or *linearization*, put on top of a given nested trace. Thus, it interleaves concurrent events performed by distinct processes.

Again, we fix a distributed call-return alphabet $\tilde{\Sigma} = (\Sigma, P, \mathit{type}, \mathit{dom})$. However, note that the concrete distribution of internal actions in terms of dom only matters when we consider nested traces.

Definition 4 (*Linearization*) Let $T = (E, (\lhd_p)_{p \in P}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NTr}(\tilde{\Sigma})$. A *linearization* of T is any structure of the form $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$ where $\lhd_{+1}$ is the direct-successor relation of some total order on E such that $\bigcup_{p \in P} \lhd_p \subseteq (\lhd_{+1})^*$. In particular, W and T share the same call-return relation.

By $\mathit{lin}(T)$, we denote the set of all linearizations of T. Extending this to languages $L \subseteq \mathsf{NTr}(\tilde{\Sigma})$, we let $\mathit{lin}(L) \overset{\text{def}}{=} \bigcup_{T \in L} \mathit{lin}(T)$.

Now, the linearizations of nested traces are precisely what we call nested words. Formally, the set of *nested words* is given by $\mathsf{NW}(\tilde{\Sigma}) \overset{\text{def}}{=} \mathit{lin}(\mathsf{NTr}(\tilde{\Sigma}))$. We do not distinguish isomorphic linearizations/nested words.

Example 3 Consider the distributed call-return alphabet $\tilde{\Sigma}$ from Example 1. Figure 6 shows a nested word over $\tilde{\Sigma}$. It is a linearization of the nested trace from Fig. 4. Straight edges represent the relation $\lhd_{+1}$, whereas curved edges represent $\lhd_{\mathsf{cr}}$.

Now, we consider automata that are suitable to represent global specifications of multi-threaded recursive programs. Though these automata use a set of stack symbols, they run, similarly to NTAs, directly on nested words so that stacks are implicitly given and not referred to explicitly (cf. [5]).

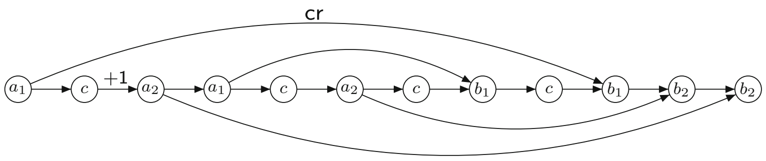

Fig. 6 Nested word W

Definition 5 (*NWA*) A *nested-word automaton (NWA)* over $\tilde{\Sigma}$ is a tuple $\mathcal{A} = (S, \Gamma, \Delta, \iota, F)$ where

- S is the nonempty finite set of *states*,
- Γ is the nonempty finite set of *stack symbols*,
- $\iota \in S$ is the *initial state*,
- $F \subseteq S$ is the set of *final states*, and
- $\Delta = \Delta_{\mathsf{call}} \uplus \Delta_{\mathsf{ret}} \uplus \Delta_{\mathsf{int}}$ is the *transition relation*, partitioned into
  - $\Delta_{\mathsf{call}} \subseteq S \times \Sigma_{\mathsf{call}} \times \Gamma \times S$,
  - $\Delta_{\mathsf{ret}} \subseteq S \times \Sigma_{\mathsf{ret}} \times \Gamma \times S$, and
  - $\Delta_{\mathsf{int}} \subseteq S \times \Sigma_{\mathsf{int}} \times S$.

The behavior of an NWA is defined similarly to the behavior of an NTA. Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$ be a nested word. Assume that $E = \{e_1, \ldots, e_n\}$ with $e_1 \lhd_{+1} e_2 \lhd_{+1} \ldots \lhd_{+1} e_n$. A *run* of $\mathcal{A}$ on W is a pair (ρ, σ) of mappings $\rho : E \to S$ and $\sigma : E_{\mathsf{call}} \cup E_{\mathsf{ret}} \to \Gamma$ such that,

- for all $(e, f) \in \lhd_{\mathsf{cr}}$, we have $\sigma(e) = \sigma(f)$, and
- for every $i \in \{1, \ldots, n\}$ (letting $\rho(e_0) = \iota$),

$$\begin{cases} \big(\rho(e_{i-1}), \lambda(e_i), \rho(e_i)\big) \in \Delta_{\mathsf{int}} & \text{if } e_i \in E_{\mathsf{int}} \\ \big(\rho(e_{i-1}), \lambda(e_i), \sigma(e_i), \rho(e_i)\big) \in \Delta_{\mathsf{call}} \cup \Delta_{\mathsf{ret}} & \text{if } e_i \in E_{\mathsf{call}} \cup E_{\mathsf{ret}}\,. \end{cases}$$

The run (ρ, σ) is accepting if $\rho(e_n) \in F$ (in particular, $\iota \in F$ if $n = 0$). The set of nested words over $\tilde{\Sigma}$ for which there is an accepting run is denoted by $L(\mathcal{A})$.

Example 4 An NWA $\mathsf{A} = (\{s_0, \ldots, s_5\}, \{\$\}, \Delta, s_0, \{s_5\})$ over the distributed call-return alphabet $\tilde{\Sigma}$ from Examples 1 and 3 is given in Fig. 7. The transitions are self-explanatory. For example, the set Δ_{int} contains (s_3, c, s_4), and Δ_{ret} contains $(s_4, b_2, \$, s_5)$, which is indicated by the edge from s_4 to s_5 with label $b_{2|\$}$. Note that A accepts those nested words that (i) contain an infix a_1a_2 or a_2a_1, (ii) have a call and a subsequent return phase, separated by some action c, and (iii) schedule returns of process 1 before those of process 2. The nested word over $\tilde{\Sigma}$ from Fig. 6 is accepted by A.

It is easy to see that NWAs (and NTAs) are closed under union and intersection. This has to be seen in contrast to the fact that the class of context-free languages (CFLs) is not closed under intersection. The intuitive reason for this is that, in CFLs, actions are no longer "visibly". Indeed, augmenting words by the nesting structure makes sure that, while simulating two NWAs, the automaton for intersection performs push and pop operations at the same positions of the input word.

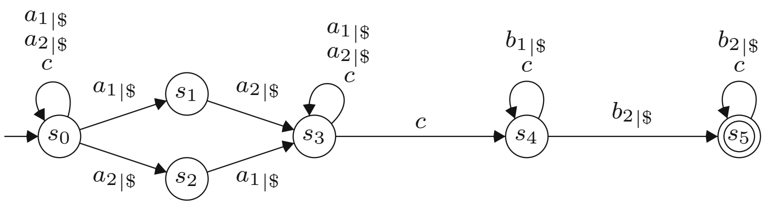

Fig. 7 Nested-word automaton A

Unfortunately, in the presence of at least two processes/stacks, most basic verification problems for NWAs are undecidable, such as nonemptiness. The following result is folklore (and, of course, carries over to NTAs):

Theorem 1 *The following problem is undecidable:*
INSTANCE: A distributed call-return alphabet $\tilde{\Sigma}$; NWA $\mathcal{A}$ over $\tilde{\Sigma}$
QUESTION: $L(\mathcal{A}) \neq \emptyset$?

In fact, one can easily simulate a two-counter machine based on two stacks, choosing $\tilde{\Sigma}$ such that $|P| = 2$. Note that the problem is decidable when $|P| = 1$, as then we deal with (visibly) context-free languages.

4 Realizability of NWA specifications

We will now start studying the realizability problem. Given an NWA, we would like to construct an NTA that represents the language of the NWA, but in a distributed environment. Of course, we have to define what "represents" means here, and we will also have to identify conditions under which a given NWA is realizable as an NTA at all.

Now, NTAs are a truly concurrent model as processes may evolve independently unless they perform synchronizing actions from Σ_{int}. NWAs, on the other hand, possess one single state space, and the global control may enforce an order even on a priori independent actions. Yet, there are tight connections between NTAs and NWAs. In Sect. 4.1, we will formalize that relation. Then, in Sect. 4.2, we consider realizability in a restricted, but well-known setting, namely without calls and returns. Finally, in Sects. 4.3 and 4.4, we study the realizability problem in full generality.

4.1 From traces to linearizations and back

Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$ be a nested word over $\tilde{\Sigma}$. The call-return relation $\lhd_{\mathsf{cr}}$ is uniquely determined by the other ingredients of W. Let us formalize this and assume that $E = \{e_1, \ldots, e_n\}$ with $e_1 \lhd_{+1} e_2 \lhd_{+1} \ldots \lhd_{+1} e_n$. The *string* of W is defined as $\mathit{string}(W) \overset{\text{def}}{=} \lambda(e_1) \ldots \lambda(e_n) \in \Sigma^*$. For example, for the nested word W from Fig. 6, we have $\mathit{string}(\mathsf{W}) = a_1ca_2a_1ca_2cb_1cb_1b_2b_2$. Now, given any word $w \in \Sigma^*$, there is at most one (up to isomorphism) nested word W over $\tilde{\Sigma}$ such that $\mathit{string}(W) = w$. If it exists, then we denote it by $\mathit{nested}(w)$. Note that, for any nested word W, we have $\mathit{nested}(\mathit{string}(W)) = W$. For languages $L \subseteq \mathsf{NW}(\tilde{\Sigma})$, we set $\mathit{string}(L) = \{\mathit{string}(W) \mid W \in L\}$.

Given $W \in \mathsf{NW}(\tilde{\Sigma})$, there is a unique nested trace $T \in \mathsf{NTr}(\tilde{\Sigma})$ such that $W \in \mathit{lin}(T)$. We denote this trace by $\mathit{trace}(W)$. Again, this is extended to languages $L \subseteq \mathsf{NW}(\tilde{\Sigma})$, and we set $\mathit{trace}(L) = \{\mathit{trace}(W) \mid W \in L\}$. The relation between the mappings *string* and *nested* as well as *lin* and *trace* is illustrated in Fig. 8.

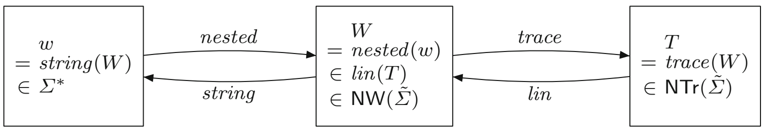
$$\begin{tikzcd} \boxed{\begin{array}{l} w \\ = \mathit{string}(W) \\ \in \Sigma^* \end{array}} \arrow[r, bend left, "\mathit{nested}"] & \boxed{\begin{array}{l} W \\ = \mathit{nested}(w) \\ \in \mathit{lin}(T) \\ \in \mathsf{NW}(\tilde{\Sigma}) \end{array}} \arrow[l, bend left, "\mathit{string}"] \arrow[r, bend left, "\mathit{trace}"] & \boxed{\begin{array}{l} T \\ = \mathit{trace}(W) \\ \in \mathsf{NTr}(\tilde{\Sigma}) \end{array}} \arrow[l, bend left, "\mathit{lin}"] \end{tikzcd}$$

Fig. 8 Relation between strings, nested words, and nested traces

Since $\mathit{nested}(\mathit{string}(W)) = W$ for any nested word W, we may consider the set of linearizations of T as a string language over Σ. This will facilitate some definitions when we consider languages up to a congruence relation taking into account that some actions are independent (those that do not share processes), while others are not (those that have at least one process in common).

Independence relation Given $\tilde{\Sigma}$, we consider the classical *independence relation* $I_{\tilde{\Sigma}} \overset{\text{def}}{=} \{(a, b) \in \Sigma \times \Sigma \mid \mathit{dom}(a) \cap \mathit{dom}(b) = \emptyset\}$. Note that $I_{\tilde{\Sigma}}$ is irreflexive and symmetric. Its complement is the *dependence relation* $D_{\tilde{\Sigma}} \overset{\text{def}}{=} (\Sigma \times \Sigma)\backslash I_{\tilde{\Sigma}}$, which is then reflexive and symmetric. With this, let $\sim_{\tilde{\Sigma}} \subseteq \Sigma^* \times \Sigma^*$ be the least congruence relation that satisfies $ab \sim_{\tilde{\Sigma}} ba$ for all $(a, b) \in I_{\tilde{\Sigma}}$.

For instance, if $\tilde{\Sigma}$ is the distributed call-return alphabet from Example 3, then the set $\{a_1a_2cb_1b_2\, ,\ a_2a_1cb_1b_2\, ,\ a_1a_2cb_2b_1\, ,\ a_2a_1cb_2b_1\}$ is an equivalence class of $\sim_{\tilde{\Sigma}}$.

The equivalence relation $\sim_{\tilde{\Sigma}}$ is lifted in the natural way to nested words: we let $W \sim_{\tilde{\Sigma}} W'$ if $\mathit{string}(W) \sim_{\tilde{\Sigma}} \mathit{string}(W')$. Moreover, we say that $L \subseteq \mathsf{NW}(\tilde{\Sigma})$ is $\sim_{\tilde{\Sigma}}$*-closed* if we have $L = [L]_{\tilde{\Sigma}} \overset{\text{def}}{=} \{W \in \mathsf{NW}(\tilde{\Sigma}) \mid W \sim_{\tilde{\Sigma}} W' \text{ for some } W' \in L\}$.

Realizability We will consider an NWA $\mathcal{A}$ to be a specification of a system, and we are looking for a *realization* or *implementation* of $\mathcal{A}$, which is provided by an NTA $\mathcal{C}$ such that $L(\mathcal{C}) = \mathit{trace}(L(\mathcal{A}))$. Actually, specifications often have a "global" view of the system, and the difficult task is to *distribute* the state space onto the processes, which henceforth communicate in a restricted manner according to the predefined system architecture $\tilde{\Sigma}$. Note that, unlike $\mathit{lin}(L(\mathcal{C}))$, the language $L(\mathcal{A})$ of an NWA $\mathcal{A}$ is not necessarily $\sim_{\tilde{\Sigma}}$-closed. However, $\mathcal{A}$ may yet be considered as an incomplete specification so that we can still ask for an NTA $\mathcal{C}$ such that $L(\mathcal{C}) = \mathit{trace}(L(\mathcal{A}))$.

Observe that it is easy to come up with an NWA recognizing the linearizations of the nested traces recognized by a given NTA. Essentially, the state space of the NWA is the Cartesian product of the local state spaces.

Lemma 1 *Let $\mathcal{C}$ be an NTA over $\tilde{\Sigma}$. There is an NWA $\mathcal{A}$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}) = \mathit{lin}(L(\mathcal{C}))$.*

4.2 Realizability in the absence of stacks: well-known facts

We are, however, interested in the other direction, i.e., to transform a given NWA into an NTA. As a preparation, we now recall two well-known theorems from Mazurkiewicz trace theory. The first one, Zielonka's celebrated theorem, applies to distributed call-return alphabets such that $\Sigma = \Sigma_{\mathsf{int}}$. Below, it will be lifted to general distributed call-return alphabets.

Theorem 2 ([46]) *Suppose $\Sigma = \Sigma_{int}$. Let $\mathcal{A}$ be an NWA over $\tilde{\Sigma}$ such that $L(\mathcal{A})$ is $\sim_{\tilde{\Sigma}}$-closed. Then, there is an NTA $\mathcal{C}$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}) = \mathit{trace}(L(\mathcal{A}))$.*

Note that the theorem actually yields a *deterministic* automaton $\mathcal{C}$ (we omit the definition of "deterministic"). The doubly exponential complexity (in the number of processes) of Zielonka's construction has later been reduced to singly exponential [16,19].

Moreover, again under the assumption $\Sigma = \Sigma_{\mathsf{int}}$, closure of a word language under $\sim_{\tilde{\Sigma}}$ is a decidable criterion:

Theorem 3 ([37,40]) *The following decision problem is* PSPACE*-complete:*

INSTANCE: $\tilde{\Sigma}$ such that $\Sigma = \Sigma_{\mathsf{int}}$; NWA $\mathcal{A}$ over $\tilde{\Sigma}$
QUESTION: Is $L(\mathcal{A}) \sim_{\tilde{\Sigma}}$-closed ?

4.3 Realizability in the presence of stacks

In Theorems 2 and 3, the given NWA and the NTA do not employ any stack so that we actually deal with a finite and a Zielonka automaton, respectively. We can lift Zielonka's theorem to distributed *call-return* alphabets:

Theorem 4 *Let $\mathcal{A}$ be an NWA over $\tilde{\Sigma}$ such that the language $L(\mathcal{A})$ is $\sim_{\tilde{\Sigma}}$-closed. There is an NTA $\mathcal{C}$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}) = trace(L(\mathcal{A}))$.*

Remark 1 The size of $\mathcal{C}$ is at most doubly exponential in $|\mathcal{A}|$ and triply exponential in $|\Sigma|$.

The proof is postponed to the next subsection. Theorem 4 demonstrates that NWAs, though they have a global view of the system in terms of one single state space, are suitable specifications for NTAs provided they recognize a $\sim_{\tilde{\Sigma}}$-closed language. Unfortunately, it is in general undecidable if the language of a given NWA is $\sim_{\tilde{\Sigma}}$-closed.

Theorem 5 *The following problem is undecidable:*

INSTANCE: $\tilde{\Sigma}$; NWA $\mathcal{A}$ over $\tilde{\Sigma}$
QUESTION: Is $L(\mathcal{A})$ $\sim_{\tilde{\Sigma}}$-closed ?

Proof We proceed by a reduction from the undecidable emptiness problem (cf. Theorem 1). Let $\tilde{\Sigma} = (\Sigma, P, type, dom)$ be the given distributed call-return alphabet and $\mathcal{A} = (S, \Gamma, \Delta, \iota, F)$ be the given NWA over $\tilde{\Sigma}$. We assume $P = \{1, 2\}$, as Theorem 1 already holds when we restrict to two processes. Moreover, we assume that $\mathcal{A}$ has a single final state f.

From the given $\tilde{\Sigma}$, we first define a new distributed call-return alphabet $\tilde{\Sigma}' = (\Sigma', P, type', dom')$ where we simply add two independent internal actions a and b that do not occur in Σ. That is, $\Sigma' = \Sigma \uplus \{a, b\}$, $type'(a) = type'(b) = \mathsf{int}$, $dom'(a) = \{1\}$, and $dom'(b) = \{2\}$. On letters from Σ, $type'$ and dom' behave like $type$ and dom, respectively.

Building on $\mathcal{A}$, we then define an NWA $\mathcal{A}'$ over $\tilde{\Sigma}'$. Essentially, $\mathcal{A}'$ coincides with $\mathcal{A}$. However, being in f, we allow $\mathcal{A}'$ to read ab and to then go into a new final state, while f is no longer final. Formally, $\mathcal{A}' = (S \uplus \{s_1, s_2\}, \Gamma, \Delta', \iota, \{s_2\})$ where $\Delta' = \Delta \cup \{(f, a, s_1), (s_1, b, s_2)\}$.

We will show that $L(\mathcal{A}')$ is *not* $\sim_{\tilde{\Sigma}'}$-closed iff $L(\mathcal{A}) \neq \emptyset$. As the latter is undecidable by Theorem 1, this will conclude the proof. Of course, $L(\mathcal{A}) = \emptyset$ implies $L(\mathcal{A}') = \emptyset$ so that $L(\mathcal{A}')$ is trivially $\sim_{\tilde{\Sigma}'}$-closed. If, on the other hand, $L(\mathcal{A}) \neq \emptyset$, then f is reachable in $\mathcal{A}$ (formally, there is a nested word with an accepting run that ends in f). Thus, $\mathcal{A}'$ accepts some nested word whose associated string ends in ab. Since a and b do not occur elsewhere in $\mathcal{A}'$, the language $L(\mathcal{A}')$ cannot be $\sim_{\tilde{\Sigma}'}$-closed. □

Interestingly, closure of a context-free language is undecidable, too [43, Theorem 4.32]. This result is orthogonal to ours, as the problem is again decidable for *visibly* pushdown languages (which can be shown by a simple adaptation of the proof from [37,40]).

4.4 Proof of Theorem 4

We now develop the proof of Theorem 4. The theorem is actually a corollary of the more general Lemma 2 below, which uses the notion of a lexicographic normal form and is also used to prove a different result on NWA realizability based on underapproximations (Theorem 12 in Sect. 5 below).

Let us fix a strict total order $<_{\mathsf{lex}}$ on Σ. Then, $<_{\mathsf{lex}}$ naturally induces a strict lexicographic order on Σ^*, which will also be denoted by $<_{\mathsf{lex}}$.

We say that a nested word $W \in \mathsf{NW}(\tilde{\Sigma})$ is *minimal* if, for all nested words $W' \neq W$ that are equivalent to W wrt. $\sim_{\tilde{\Sigma}}$, we have $string(W) <_{\mathsf{lex}} string(W')$. Given $W \in \mathsf{NW}(\tilde{\Sigma})$, let $nf_{\mathsf{lex}}(W)$ denote its *normal form*, i.e., the unique minimal nested word $W' \in \mathsf{NW}(\tilde{\Sigma})$ such that $W \sim_{\tilde{\Sigma}} W'$. We extend the mapping nf_{lex} to languages $L \subseteq \mathsf{NW}(\tilde{\Sigma})$ as expected and let $nf_{\mathsf{lex}}(L) = \{nf_{\mathsf{lex}}(W) \mid W \in L\}$.

Theorem 6 ([38]) *Suppose $\Sigma = \Sigma_{\mathrm{int}}$ and let $L \subseteq \mathsf{NW}(\tilde{\Sigma})$ such that $string(L) \subseteq \Sigma^*$ is regular and $nf_{\mathrm{lex}}(L) = L$. Then, there is an NWA $\mathcal{A}$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}) = [L]_{\tilde{\Sigma}}$ (in other words, $[L]_{\tilde{\Sigma}}$ is a regular word language over the alphabet Σ).*

We will use Theorems 2 and 6 to show the following crucial lemma:

Lemma 2 *Let $\mathcal{A}$ be an NWA over $\tilde{\Sigma}$ such that $nf_{\mathrm{lex}}(L(\mathcal{A})) \subseteq L(\mathcal{A})$. Then, there is an NTA $\mathcal{C}$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}) = trace(L(\mathcal{A}))$.*

Proof We proceed in several steps. In a nutshell, we first interpret $\mathcal{A}$ as an NWA over a distributed alphabet (without any call or return actions). This allows us to apply the Theorems by Zielonka and Ochmański. Reinterpreting the resulting NTA over the modified alphabet as an NTA over the original alphabet, gives the desired automaton.

Let $\mathcal{A} = (S, \Gamma, \Delta, \iota, F)$ be the given NWA with $nf_{\mathsf{lex}}(L(\mathcal{A})) \subseteq L(\mathcal{A})$. We define the distributed alphabet as $\tilde{\Omega} = (\Omega, P, type', dom')$ where $\Omega = \Sigma \times \Gamma$ and, for all $(a, A) \in \Sigma \times \Gamma$, $type'((a, A)) = \mathsf{int}$ and $dom'((a, A)) = dom(a)$. Note that $\Omega = \Omega_{\mathsf{int}}$. Now, we proceed in several steps.

1. From the NWA $\mathcal{A}$ over $\tilde{\Sigma}$, we build an NWA $\mathcal{B}$ over $\tilde{\Omega}$ as follows. A transition $(s, a, A, s') \in \Delta_{\mathsf{call}} \cup \Delta_{\mathsf{ret}}$ in $\mathcal{A}$ is considered as a transition $(s, (a, A), s')$ in the new NWA $\mathcal{B}$, and a transition $(s, a, s') \in \Delta_{\mathsf{int}}$ is translated to $(s, (a, A_a), s')$ with $A_a \in \Gamma$ arbitrary but fixed. The other components remain unchanged.
   Note that $\sim_{\tilde{\Sigma}}$-closure of $L(\mathcal{A})$ does not imply $\sim_{\tilde{\Omega}}$-closure of $L(\mathcal{B})$, since, in $\mathcal{B}$, two transitions executing the same call/return action do not necessarily use the same stack symbol. To overcome this problem, we will go through lexicographic normal forms.
2. Fix any strict total order $<'_{\mathsf{lex}}$ such that $a <_{\mathsf{lex}} b$ implies $(a, A) <'_{\mathsf{lex}} (b, B)$, for all $A, B \in \Gamma$. We know that $nf_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))$ is recognized by some finite automaton/NWA [38]. Thus, using a product construction, we find an NWA $\mathcal{B}'$ over $\tilde{\Omega}$ such that
$$L(\mathcal{B}') = L(\mathcal{B}) \cap nf_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega})).$$
3. Since $\Omega = \Omega_{\mathsf{int}}$, we can now apply Theorem 6 to obtain, from $\mathcal{B}'$, an NWA $\mathcal{B}''$ over $\tilde{\Omega}$ such that
$$L(\mathcal{B}'') = [L(\mathcal{B}) \cap nf_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))]_{\tilde{\Omega}}.$$
4. Next, we apply Theorem 2 to $\mathcal{B}''$ and get an NTA $\widehat{\mathcal{C}}$ over $\tilde{\Omega}$ such that
$$L(\widehat{\mathcal{C}}) = trace([L(\mathcal{B}) \cap nf_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))]_{\tilde{\Omega}}).$$
5. The final step is to reinterpret $\widehat{\mathcal{C}}$ as an NTA $\mathcal{C}$ over the original alphabet $\tilde{\Sigma}$: Call or return transition $(s, (a, A), s')$ becomes (s, a, A, s'), and an internal transition $(s, (a, A_a), s')$ becomes (s, a, s').

We show that $L(\mathcal{C}) = trace(L(\mathcal{A}))$.

To prove the inclusion from left to right, let $T = (E, (\lhd_p)_{p\in P}, \lhd_{\mathsf{cr}}, \lambda) \in L(\mathcal{C})$. There is an accepting run (ρ, σ) of $\mathcal{C}$ on T. Recall that $\sigma : E_{\mathsf{call}} \cup E_{\mathsf{ret}} \to \Gamma$. Let $\widehat{\lambda} : E \to \Omega$ be given by $\widehat{\lambda}(e) = (\lambda(e), \sigma(e))$ if $e \in E_{\mathsf{call}} \cup E_{\mathsf{ret}}$, and $\widehat{\lambda}(e) = (\lambda(e), A_{\lambda(e)})$ if $e \in E_{\mathsf{int}}$. By the definition of $\mathcal{C}$, the (un)nested trace $\widehat{T} = (E, (\lhd_p)_{p\in P}, \emptyset, \widehat{\lambda})$ over $\tilde{\Omega}$ is contained in $L(\widehat{\mathcal{C}})$.

Since $L(\widehat{\mathcal{C}}) = \mathit{trace}([L(\mathcal{B}) \cap \mathit{nf}_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))]_{\tilde{\Omega}})$, there is a linearization $\widehat{W} = (E, \lhd_{+1}, \emptyset, \widehat{\lambda})$ of $\widehat{T}$ such that $\widehat{W} \in L(\mathcal{B})$. Note that, as $\widehat{T}$ and T share the same underlying partial order, $(E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NW}(\tilde{\Sigma})$ (with $\lhd_{\mathsf{cr}}$ and λ taken from T) is a linearization of T. Let ρ' be an accepting run of the NWA $\mathcal{B}$ on $\widehat{W}$ (as there are only internal actions, a run does not need to map events to stack symbols). Since σ is part of a run of the NTA $\mathcal{C}$ on T, we have $\sigma(e) = \sigma(f)$ for all $(e, f) \in \lhd_{\mathsf{cr}}$. Moreover, $(\rho'(e), \widehat{\lambda}(f), \rho'(f))$ is a transition of $\mathcal{B}$, for all $(e, f) \in \lhd_{+1}$. We deduce that (ρ', σ) is an accepting run of $\mathcal{A}$ on $(E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$. We conclude that $T = (E, (\lhd_p)_{p\in P}, \lhd_{\mathsf{cr}}, \lambda) \in \mathit{trace}(L(\mathcal{A}))$.

For the converse direction, let $T = (E, (\lhd_p)_{p\in P}, \lhd_{\mathsf{cr}}, \lambda) \in \mathit{trace}(L(\mathcal{A}))$. Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda) \in L(\mathcal{A})$ be the minimal linearization of T. Due to $\mathit{nf}_{\mathsf{lex}}(L(\mathcal{A})) \subseteq L(\mathcal{A})$, we have $W \in L(\mathcal{A})$. Let (ρ, σ) be an accepting run of $\mathcal{A}$ on W and suppose σ' is the extension of σ that maps every $e \in E_{\mathsf{int}}$ to $A_{\lambda(e)}$. As W is in lexicographic normal form wrt. $<_{\mathsf{lex}}$ and $<'_{\mathsf{lex}}$ is a conservative extension of $<_{\mathsf{lex}}$, we have that $W' = (E, \lhd_{+1}, \emptyset, (\lambda, \sigma'))$ is in lexicographic normal form wrt. $<'_{\mathsf{lex}}$. By the definition of $\mathcal{B}$, we have $W' \in L(\mathcal{B})$. Thus, we obtain $W' \in L(\mathcal{B}') = L(\mathcal{B}) \cap \mathit{nf}_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))$.

This implies $\mathit{trace}(W') = (E, (\lhd_p)_{p\in P}, \emptyset, (\lambda, \sigma')) \in \mathit{trace}(L(\mathcal{B}')) = L(\widehat{\mathcal{C}})$. Let ρ' be an accepting run of $\widehat{\mathcal{C}}$ on $\mathit{trace}(W')$. As $\sigma'(e) = \sigma'(f)$ for all $(e, f) \in \lhd_{\mathsf{cr}}$ and, moreover, σ and σ' coincide on $E_{\mathsf{call}} \cup E_{\mathsf{ret}}$, (ρ', σ) is an accepting run of $\mathcal{C}$ on $T = (E, (\lhd_p)_{p\in P}, \lhd_{\mathsf{cr}}, \lambda)$. We conclude $T \in L(\mathcal{C})$. □

Remark 2 We will now argue that the size of $\mathcal{C}$ is at most doubly exponential in the size $|\mathcal{A}|$ of $\mathcal{A}$, and triply exponential in $|\Sigma|$. Hereby, we define $|\mathcal{A}|$ as $|S| + |\Gamma|$.

First, according to [23], there is an NWA with at most $(|\Sigma| + 1)!$-many states recognizing $\mathit{nf}_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))$. Thus, $\mathcal{B}'$ is of size $n \overset{\text{def}}{=} |\mathcal{A}| \cdot (|\Sigma| + 1)!$. Note that the size of the construction in [23] actually depends on $|\Sigma|$ rather than $|\Omega|$.

The NWA for $\mathit{nf}_{\mathsf{lex}}(\mathsf{NW}(\tilde{\Omega}))$ and, therefore, $\mathcal{B}'$ have the property of being *loop-connected* [38]. The size of $\mathcal{B}''$ can therefore be bounded by $N \overset{\text{def}}{=} (n^2 \cdot 2^{|\Sigma|})^{(n-1)(|\Sigma|+1)+1}$ [23,35]. From [19], we know that the size of $\widehat{\mathcal{C}}$ can be bounded by $2^{N^2\cdot(|\Sigma|^2+|\Sigma|)+2|\Sigma|^4}$. As $\widehat{\mathcal{C}}$ and $\mathcal{C}$ have the same states, we conclude that the size of $\mathcal{C}$ is doubly exponential in $|\mathcal{A}|$ and triply exponential in $|\Sigma|$.

Note that this upper bound is obtained via a series of reductions that all assume a worst-case complexity without any optimization. It is unlikely that, in practice, each of these worst-cases is encountered at the same time.

Now, Theorem 4 is a corollary from Lemma 2, since we have $\mathit{nf}_{\mathsf{lex}}(L(\mathcal{A})) \subseteq L(\mathcal{A})$ whenever $L(\mathcal{A})$ is $\sim_{\tilde{\Sigma}}$-closed.

5 Realizability of restricted specifications

Despite the extension of Zielonka's theorem that we obtained in terms of Theorem 4, the undecidability result stated in Theorem 5 is unsatisfactory. In this section, we will impose restrictions on NWA specifications that allow us to combine a Zielonka-like theorem with decidable criteria for realizability.

5.1 Contexts, phases, scopes, and ordered stacks

In their seminal paper [41], Qadeer and Rehof exploited the fact that errors of recursive programs typically occur already within a few *contexts* where a context refers to an execution involving only one process. Imposing a bound on the number of contexts indeed renders many verification problems decidable. In other words, instead of looking at all possible executions, we consider an underapproximation of the actual system behavior. This view is particularly suitable when checking *positive specifications*: If a system $\mathcal{A}$ is required to exhibit all behaviors given by a set *Good*, then it is sufficient to show that $Good \subseteq L(\mathcal{U})$ is true for a restricted version $\mathcal{U}$ of $\mathcal{A}$ such that $L(\mathcal{U}) \subseteq L(\mathcal{A})$. Thus, we are interested in restrictions $\mathcal{U}$ of $\mathcal{A}$ such that the inclusion problem $Good \subseteq L(\mathcal{U})$ is decidable and that we may adjust incrementally so as to converge to $L(\mathcal{A})$. On the other hand, given a *negative specification* (or, safety property) *Bad*, we can also draw conclusions from $L(\mathcal{U}) \cap Bad \neq \emptyset$, while showing complete absence of bad behaviors would require an overapproximation of the system behavior.

Next, we recall the notion of a context as well as similar notions that have been defined in the literature. Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NW}(\tilde{\Sigma})$ be a nested word, and let $\leq \overset{\text{def}}{=} (\lhd_{+1})^*$, with strict part $<$. An *interval* of W is a set $I \subseteq E$ such that $I = \emptyset$ or $I = [e, f] \overset{\text{def}}{=} \{g \in E \mid e \leq g \leq f\}$ for some $e, f \in E$. In a *context*, only one designated process is allowed to call or return, while a *phase* only restricts return operations:

– A *context* of W is an interval I such that $I \cap (E_{\mathsf{call}} \cup E_{\mathsf{ret}}) \subseteq E_p$ for some $p \in P$.
– A *phase* of W is an interval I such that $I \cap E_{\mathsf{ret}} \subseteq E_p$ for some $p \in P$.

Definition 6 (*k-context word* [41] *and k-phase word* [25]) Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$ be a nested word and $k \geq 1$. We say that W is a *k-context word* (*k-phase word*) if there are contexts (phases, respectively) $I_1, \ldots, I_k$ of W such that $E = I_1 \cup \ldots \cup I_k$.

Note that every context is a phase, while the converse does not hold in general. Therefore, every k-context word is a k-phase word, but not the other way around. Two *orthogonal* restrictions have been defined in terms of bounded scopes and ordered nested words, which we consider next. A scope-bounded word restricts the number of contexts between a push and the corresponding pop operation.

Definition 7 (*k-scope word* [28]) Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$ be a nested word and $k \geq 1$. We call W a *k-scope word* if, for all $(e, f) \in \lhd_{\mathsf{cr}}$, there exist contexts $I_1, \ldots, I_k$ of W such that $[e, f] = I_1 \cup \ldots \cup I_k$.

Finally, in an *ordered* word, we refer to a fixed total ordering $\preceq$ on P (with irreflexive part $\prec$). Then, a pop operation can only be performed by the process that is minimal among all processes with a nonempty stack.

Definition 8 (*Ordered word* [6]) Let $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda)$ be a nested word. We call W an *ordered word* (wrt. $\preceq$) if, for all $p, p' \in P$, $e, f \in E_p$, and $f' \in E_{\mathsf{ret}} \cap E_{p'}$ such that $e \lhd_{\mathsf{cr}} f$ and $e < f' < f$, we have $p' \preceq p$.

Example 5 (Continues Example 3) Figure 9 illustrates the concepts introduced above by means of the nested word W from Fig. 6. The shadowed areas are dedicated to process 2. Thus, W is a 6-context word, a 5-scope word, and a 2-phase word. All these bounds are optimal. Moreover, W is an ordered word under the assumption $1 \prec 2$ (but not for $2 \prec 1$).

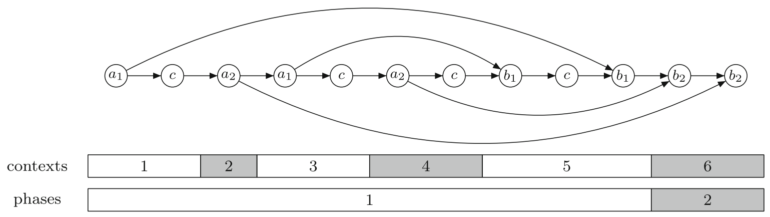

Fig. 9 Illustration of context, phase, and scope

Table 1 Underapproximate decision problems

CONTEXT- NONEMPTINESS		SCOPE- NONEMPTINESS	
INSTANCE:	$\tilde{\Sigma}$; NWA $\mathcal{A}$; $k \geq 1$	INSTANCE:	$\tilde{\Sigma}$; NWA $\mathcal{A}$; $k \geq 1$
QUESTION:	$L_{k\text{-cnt}}(\mathcal{A}) \neq \emptyset$?	QUESTION:	$L_{k\text{-scp}}(\mathcal{A}) \neq \emptyset$?
PHASE- NONEMPTINESS		ORD- NONEMPTINESS	
INSTANCE:	$\tilde{\Sigma}$; NWA $\mathcal{A}$; $k \geq 1$	INSTANCE:	$\tilde{\Sigma}$; NWA $\mathcal{A}$
QUESTION:	$L_{k\text{-ph}}(\mathcal{A}) \neq \emptyset$?	QUESTION:	$L_{\text{ord}}(\mathcal{A}) \neq \emptyset$?

Let $\mathfrak{R} \overset{\text{def}}{=} \{k\text{-cnt}\,,\ k\text{-ph}\,,\ k\text{-scp} \mid k \geq 1\} \cup \{\text{ord}\}$ be the set of possible "restrictions". For $\theta \in \mathfrak{R}$, the set of θ-words is denoted by $\text{NW}_\theta(\tilde{\Sigma})$. In particular, by $\text{NW}_{\text{ord}}(\tilde{\Sigma})$, we denote the set of ordered words, silently assuming a total order on P. Moreover, for an NWA $\mathcal{A}$ over $\tilde{\Sigma}$, we let $L_\theta(\mathcal{A}) \overset{\text{def}}{=} L(\mathcal{A}) \cap \text{NW}_\theta(\tilde{\Sigma})$.

Example 6 Consider the sample NWA A from Fig. 7. We have that $L(\mathsf{A}) = L_{2\text{-ph}}(\mathsf{A}) = L_{\text{ord}}(\mathsf{A})$. On the other hand, $L_{k\text{-cnt}}(\mathsf{A})$ and $L_{k\text{-scp}}(\mathsf{A})$ are strictly included in $L(\mathsf{A})$, for all $k \geq 1$.

Let us consider the respective nonemptiness problems, given in Table 1. In all cases, we assume that the parameter k is given in unary. Indeed, all problems are decidable, with varying complexities. For a comparison between the effects of a unary and a binary encoding of k, see [8].

Theorem 7 ([32,41]) CONTEXT- NONEMPTINESS *is* NP-*complete.*

Theorem 8 ([25,27]) PHASE- NONEMPTINESS *is* 2-EXPTIME-*complete.*

Theorem 9 ([28]) SCOPE- NONEMPTINESS *is* PSPACE-*complete.*

Theorem 10 ([1]) ORD- NONEMPTINESS *is* 2-EXPTIME-*complete.*

In [36], Madhusudan and Parlato give a uniform argument for decidability of the above problems: For $\theta \in \{k\text{-cnt}\,,\ k\text{-ph} \mid k \geq 1\} \cup \{\text{ord}\}$, the class of θ-words has bounded tree-width. By Courcelle's theorem, this implies that nonemptiness of NWAs and even model-checking of NWAs against MSO properties is decidable. It was then shown that scope-bounded words have bounded tree-width, too [31].

An alternative unifying approach is given in [13] in terms of the notion of *split-width*. Originally introduced for multiply nested words, split-width has also produced generalizations of the above-mentioned classes and other existing work on recursive message-passing systems [4,14].

We have seen that restricting the domain of nested words appropriately renders the nonemptiness problem for NWAs decidable. These restrictions produce robust classes of languages. The next theorem states that, under any of the restrictions introduced above, NWAs are complementable:

Theorem 11 ([25,29,30]) *Let $\theta \in \mathfrak{R}$. For all NWAs $\mathcal{A}$ over $\tilde{\Sigma}$, the following hold:*

1. *There exists an NWA $\mathcal{A}'$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}') = \mathsf{NW}_\theta(\tilde{\Sigma}) \backslash L(\mathcal{A})$.*
2. *There exists an NWA $\mathcal{A}'$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}') = \mathsf{NW}(\tilde{\Sigma}) \backslash L_\theta(\mathcal{A})$.*

5.2 Realizability of restricted specifications

In view of the undecidability result stated in Theorem 5, we will now consider realizability modulo restrictions to θ-words, for suitable $\theta \in \mathfrak{R}$. This will allow us to define decidable sufficient criteria for the transformation of an NWA into an NTA. Moreover, there is a Zielonka-like theorem that is tailored to underapproximations. In that theorem, we require that an NWA *represents* the θ-words of a system, while the final implementation can produce executions that do not fit into the θ-restriction.

A θ-representation is a set of nested words that does not distinguish between locally equivalent θ-words. Here, two nested words $W, W' \in \mathsf{NW}(\tilde{\Sigma})$ are said to be *locally equivalent*, written $W \sim^{\mathrm{loc}}_{\tilde{\Sigma}} W'$, if there are $u, v \in \Sigma^*$ and $(a, b) \in I_{\tilde{\Sigma}}$ such that $\mathit{string}(W) = uabv$ and $\mathit{string}(W') = ubav$.

Definition 9 Let $\theta \in \mathfrak{R}$ and $L \subseteq \mathsf{NW}(\tilde{\Sigma})$. We call L a θ-*representation* if $L \subseteq \mathsf{NW}_\theta(\tilde{\Sigma})$ and, for all $W, W' \in \mathsf{NW}_\theta(\tilde{\Sigma})$ such that $W \sim^{\mathrm{loc}}_{\tilde{\Sigma}} W'$, we have $W \in L$ iff $W' \in L$.

We do not know whether the decidability result presented below (Theorem 13) still holds when considering a language L to be a θ-representation if $L = [L]_{\tilde{\Sigma}} \cap \mathsf{NW}_\theta(\tilde{\Sigma})$.

Next, we present our Zielonka theorem suited to θ-representations. Hereby, we require that θ be a member of the set $\mathfrak{R}^- \overset{\mathrm{def}}{=} \{k\text{-}\mathsf{cnt}, k\text{-}\mathsf{ph} \mid k \geq 1\}$. The result relies on the definition of a lexicographic normal form that allows one to apply Ochmański's Theorem [38]. The crux is that reordering independent events in order to obtain the lexicographic normal form shall not affect membership in the given θ-representation. This, however, requires an extension of the underlying distributed call-return alphabet. While we do this for context- and phase-bounded executions, it is unclear whether it is possible for bounded-scope or ordered representations and if the following results holds for these classes as well.

Theorem 12 *Let $\theta \in \mathfrak{R}^-$ be a restriction and let $\mathcal{A}$ be an NWA over $\tilde{\Sigma}$ such that $L_\theta(\mathcal{A})$ is a θ-representation. Then, there is an NTA $\mathcal{C}$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}) = \mathit{trace}(L_\theta(\mathcal{A}))$.*

Proof Like in the proof of Theorem 4, we use Lemma 2. However, we cannot apply it directly, as it is in general not possible to define $<_{\mathsf{lex}}$ in such a way that $\mathit{nf}_{\mathsf{lex}}(L(\mathcal{A})) \subseteq L(\mathcal{A})$ when $L(\mathcal{A})$ is a θ-representation, with $\theta \in \mathfrak{R}^- = \{k\text{-}\mathsf{cnt}, k\text{-}\mathsf{ph} \mid k \geq 1\}$. Our trick is to extend the given distributed call-return alphabet $\tilde{\Sigma}$ by a component that indicates the current context or phase of a word position. An appropriate definition of a lexicographic ordering over this extended alphabet will then allow us to apply Lemma 2.

Suppose $\tilde{\Sigma} = (\Sigma, P, \mathit{type}, \mathit{dom})$. For any restriction $\theta \in \mathfrak{R}^-$, we define a new distributed call-return alphabet $\tilde{\Omega}_\theta = (\Omega, P, \mathit{type}', \mathit{dom}')$ as well as a lexicographic order in terms of a total order $<^\theta_{\mathsf{lex}}$ on Ω.

– $\tilde{\Omega}_{k\text{-}\mathsf{cnt}}$ is given as follows: Let $\Omega = \Sigma_{\mathsf{int}} \cup ((\Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}) \times \{1, \ldots, k\})$. Intuitively, the additional component will keep track of the current context in a nested word. For

$a \in \Sigma_{\mathsf{int}}$, we set $type'(a) = type(a) = \mathsf{int}$ and $dom'(a) = dom(a)$. For $a \in \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$ and $i \in \{1, \ldots, k\}$, we let $type'((a, i)) = type(a)$ and $dom'((a, i)) = dom(a)$. Finally, let $<_{\mathsf{lex}}^{k\text{-}\mathsf{cnt}} \subseteq \Omega \times \Omega$ be any strict total order such that $(a, i) <_{\mathsf{lex}}^{k\text{-}\mathsf{cnt}} (b, j)$ whenever $i < j$.

- $\tilde{\Omega}_{k\text{-}\mathsf{ph}}$ is given as follows: Let $\Omega = \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{int}} \cup (\Sigma_{\mathsf{ret}} \times \{1, \ldots, k\})$. Here, the additional component will keep track of the current *phase*. Therefore, it is only recorded for *return* events. For $a \in \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{int}}$, we set $type'(a) = type(a)$ and $dom'(a) = dom(a)$. For $a \in \Sigma_{\mathsf{ret}}$ and $i \in \{1, \ldots, k\}$, we set $type'((a, i)) = type(a)$ and $dom'((a, i)) = dom(a)$. Again, let $<_{\mathsf{lex}}^{k\text{-}\mathsf{ph}} \subseteq \Omega \times \Omega$ be any strict total order such that $(a, i) <_{\mathsf{lex}}^{k\text{-}\mathsf{ph}} (b, j)$ whenever $i < j$.

For contexts and phases, we will now define the canonical, "greedy" division of a nested word into intervals. Given $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NW}(\tilde{\Sigma})$, we define a mapping $ph_W : E \to \mathbb{N}$ inductively as follows: If e is the minimal event of W (wrt. $\leq$), then we let $ph_W(e) = 1$ and we set $I_e = \{e\}$. Suppose $e \lhd_{+1} f$. If $I_e \cup \{f\}$ is a phase, then let $ph_W(f) = ph_W(e)$ and $I_f = I_e \cup \{f\}$. Otherwise, set $ph_W(f) = ph_W(e) + 1$ and $I_f = \{f\}$. The mapping $ct_W : E \to \mathbb{N}$ for contexts is defined accordingly. Note that W with maximal event e is a k-phase word iff $ph_W(e) \leq k$. The corresponding statement holds for contexts.

Next, we define a mapping $h_\theta : \mathsf{NW}_\theta(\tilde{\Sigma}) \to \mathsf{NW}_\theta(\tilde{\Omega}_\theta)$. This mapping will add to a nested word additional information in terms of the extended alphabet (i.e., the context/phase).

- For a nested word $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NW}_{k\text{-}\mathsf{cnt}}(\tilde{\Sigma})$, we define $h_{k\text{-}\mathsf{cnt}}(W)$ as $(E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda')$ where $\lambda'(e) = \lambda(e)$ for all $e \in E_{\mathsf{int}}$, and $\lambda'(e) = (\lambda(e), ct_W(e))$ for all $e \in E_{\mathsf{call}} \cup E_{\mathsf{ret}}$. Note that $h_{k\text{-}\mathsf{cnt}}(W)$ is indeed a k-context word over $\tilde{\Omega}_{k\text{-}\mathsf{cnt}}$.
- Accordingly, for $W = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NW}_{k\text{-}\mathsf{ph}}(\tilde{\Sigma})$, we let $h_{k\text{-}\mathsf{ph}}(W) = (E, \lhd_{+1}, \lhd_{\mathsf{cr}}, \lambda')$ where $\lambda'(e) = \lambda(e)$ for all $e \in E_{\mathsf{call}} \cup E_{\mathsf{int}}$, and $\lambda'(e) = (\lambda(e), ph_W(e))$ for all $e \in E_{\mathsf{ret}}$. Clearly, $h_{k\text{-}\mathsf{ph}}(W)$ is a k-phase word over $\tilde{\Omega}_{k\text{-}\mathsf{ph}}$.

In the remainder of the proof, we proceed uniformly for all restrictions $\theta \in \mathfrak{R}^-$. Let $\mathcal{A} = (S, \Gamma, \Delta, \iota, F)$ be the given NWA over $\tilde{\Sigma}$ such that $L_\theta(\mathcal{A})$ is a θ-representation. One can easily construct an NWA $\mathcal{B}$ over $\tilde{\Omega}_\theta$ such that $L(\mathcal{B}) = \{h_\theta(W) \mid W \in L_\theta(\mathcal{A})\} \subseteq \mathsf{NW}_\theta(\tilde{\Omega}_\theta)$. To do so, we just remember the current context/phase and its "type". When reading a letter that does not correspond to the current type, the context/phase counter is increased.

We claim that $nf_{\mathsf{lex}}(L(\mathcal{B})) \subseteq L(\mathcal{B})$ (where the mapping nf_{lex} refers to $<_{\mathsf{lex}}^\theta$). In other words, $L(\mathcal{B})$ contains, for every $W \in L(\mathcal{B})$, the normal form of W wrt. $<_{\mathsf{lex}}^\theta$. Let $g_\theta : \Omega \to \Sigma$ define the projection that removes any additional labeling if it exists (i.e., $g_\theta((a, i)) = a$) and which is canonically extended to nested words/traces and languages. Now, $<_{\mathsf{lex}}^\theta$ has been chosen in such a way that the following holds, for all $W \in L(\mathcal{B})$. We obtain the normal form $W' \in \mathsf{NW}(\tilde{\Omega}_\theta)$ of W by successively reordering two independent neighboring events, without swapping the order of events with labels (a, i) and (b, j) such that $i < j$. Note that this implies $W' \in \mathsf{NW}_\theta(\tilde{\Omega}_\theta)$ and $h_\theta(g_\theta(W')) = W'$. Since $L_\theta(\mathcal{A})$ is a θ-representation, the reordering also preserves containment in $L(\mathcal{B})$. Thus, $nf_{\mathsf{lex}}(L(\mathcal{B})) \subseteq L(\mathcal{B})$.

By Lemma 2, there is an NTA $\mathcal{C}$ over $\tilde{\Omega}_\theta$ such that $L(\mathcal{C}) = trace(L(\mathcal{B}))$. It is easily seen that NTAs are closed under projection (which was shown in [25] for NWAs). In particular, applying g_θ to an NTA language over $\tilde{\Omega}_\theta$ yields an NTA language over $\tilde{\Sigma}$. Thus, there is an NTA $\mathcal{C}'$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}') = g_\theta(trace(L(\mathcal{B})))$. As we have $g_\theta(trace(L(\mathcal{B}))) = trace(g_\theta(L(\mathcal{B}))) = trace(L_\theta(\mathcal{A}))$, we are done. □

5.3 Deciding realizability

To complete this section, we show that, given an NWA $\mathcal{A}$ over $\tilde{\Sigma}$ and a restriction $\theta \in \mathfrak{R}$ (now including "bounded scope" and "ordered"), we can decide both whether $L_\theta(\mathcal{A})$ is $\sim_{\tilde{\Sigma}}$-closed and whether $L_\theta(\mathcal{A})$ is a θ-representation.

Theorem 13 *The following problems are decidable in elementary time:*

Instance:	$\tilde{\Sigma}$; $\theta \in \mathfrak{R}$; NWA $\mathcal{A}$ over $\tilde{\Sigma}$
Question 1:	Is $L_\theta(\mathcal{A})$ $\sim_{\tilde{\Sigma}}$-closed ?
Question 2:	Is $L_\theta(\mathcal{A})$ a θ-representation ?

Proof The proof is inspired by [37,40] where analogous problems are addressed in a finite-state setting.

Consider first Question 1. Using Theorem 11, we first transform the given NWA $\mathcal{A} = (S_1, \Gamma_1, \Delta_1, \iota_1, F_1)$ into an NWA $\mathcal{A}' = (S_2, \Gamma_2, \Delta_2, \iota_2, F_2)$ for the complement, i.e., such that $L(\mathcal{A}') = \mathsf{NW}(\tilde{\Sigma}) \setminus L_\theta(\mathcal{A})$. Now, we build an NWA $\mathcal{B}$ recognizing all nested words $W \in L(\mathcal{A})$ where $string(W)$ is of the form $uabv$ such that $(a, b) \in I_{\tilde{\Sigma}}$ and $ubav = string(W')$ for some $W' \in L(\mathcal{A}')$. Then, $L_\theta(\mathcal{A})$ is *not* $\sim_{\tilde{\Sigma}}$-closed iff $L_\theta(\mathcal{B}) \neq \emptyset$. The latter is decidable due to Theorems 7–10. To solve Question 2, the only difference is that we build $\mathcal{A}'$ such that $L(\mathcal{A}') = \mathsf{NW}_\theta(\tilde{\Sigma}) \setminus L(\mathcal{A})$ (again, using Theorem 11). In that case, $L_\theta(\mathcal{A})$ is *not* a θ-representation iff $L_\theta(\mathcal{B}) \neq \emptyset$.

For convenience, we will write an internal transition $(s, a, s') \in \Delta_i$, $i \in \{1, 2\}$, as (s, a, A^i_a, s') where $A^i_a \in \Gamma_i$ is an arbitrary fixed stack symbol. The set of states of $\mathcal{B}$ is $S_1 \times S_2 \times (\{0, 1\} \cup (I_{\tilde{\Sigma}} \times \Gamma_2 \times \Gamma_2))$. The first two components of a state are used to simulate $\mathcal{A}$ and $\mathcal{A}'$, respectively. The third component starts in 0. In states of the form $(s_1, s_2, 0)$, both automata proceed synchronously: Reading $a \in \Sigma$, $\mathcal{B}$ applies a-transitions $(s_1, a, A_1, s'_1) \in \Delta_1$ and $(s_2, a, A_2, s'_2) \in \Delta_2$ to the first and the second component, respectively, resulting in a global step $((s_1, s_2, 0), a, (A_1, A_2), (s'_1, s'_2, 0))$. Note that the stack alphabet is extended to $\Gamma_1 \times \Gamma_2$ to take into account that A_1 and A_2 are in general distinct.

When reading an input word, $\mathcal{A}$ should eventually perform an action sequence ab with $(a, b) \in I_{\tilde{\Sigma}}$, while $\mathcal{A}'$ executes ba. So suppose $\mathcal{B}$ is about to simulate transitions (s_1, a, A_1, s'_1) followed by (s'_1, b, B_1, s''_1) in $\mathcal{A}$ and transition (s_2, b, B_2, s'_2) followed by (s'_2, a, A_2, s''_2) in $\mathcal{A}'$. The global automaton $\mathcal{B}$ will produce this transition sequence "crosswise". It will first read the a and apply the transition involving $A_1 \in \Gamma_1$ to the first component. At the same time, the second component only changes its local state into s'_2. As the stack symbol B_2 cannot be applied directly, it is stored in the third component of the subsequent global state of $\mathcal{B}$, which is of the form $(s'_1, s'_2, ((a, b), B_2, A_2))$. Observe that A_2, which is associated to executing a in $\mathcal{A}'$, must be applied together with reading a so that (A_1, A_2) acts as the stack symbol. In fact, since a corresponding local transition (s'_2, a, A_2, s''_2) has to follow in $\mathcal{A}'$, the stack symbol A_2 needs to be stored as well. The formal description of this step can be found below (2). Now, being in the global state $(s'_1, s'_2, ((a, b), B_2, A_2))$, $\mathcal{B}$ will, according to the local transition (s'_1, b, B_1, s''_1), perform a b and apply (B_1, B_2) to the designated stack. Again, $\mathcal{A}'$ will only change its local state into s''_2. However, the local transition has to conform to action a and the symbol A_2 that had been stored. This step corresponds to rule (3) below. We are now in a global state of the form $(s''_1, s''_2, 1)$. In states with 1 in the third position, $\mathcal{A}$ and $\mathcal{A}'$ are again simulated in sync (rule (1)).

Formally, $\mathcal{B} = (S, \Gamma, \Delta, \iota, F)$ is given by $S = S_1 \times S_2 \times (\{0, 1\} \cup (I_{\tilde{\Sigma}} \times \Gamma_2 \times \Gamma_2))$, $\Gamma = \Gamma_1 \times \Gamma_2$, $\iota = (\iota_1, \iota_2, 0)$, and $F = F_1 \times F_2 \times \{1\}$.

Towards the transition relation Δ, we first define a relation $\Delta' \subseteq S \times \Sigma \times (\Gamma_1 \times \Gamma_2) \times S$. To obtain Δ from Δ', we then simply replace every transition of the form $(s, a, (A_1, A_2), s')$, $a \in \Sigma_{\mathsf{int}}$, by (s, a, s').

(1) For $(s_1, a, A_1, s_1') \in \Delta_1$, $(s_2, a, A_2, s_2') \in \Delta_2$, and $\beta \in \{0, 1\}$, the relation Δ' contains

$$((s_1, s_2, \beta), a, (A_1, A_2), (s_1', s_2', \beta)) .$$

(2) For $(a, b) \in I_{\tilde{\Sigma}}$, $(s_1, a, A_1, s_1') \in \Delta_1$, $(s_2, b, B_2, s_2') \in \Delta_2$, and $A_2 \in \Gamma_2$, the relation Δ' contains

$$((s_1, s_2, 0), a, (A_1, A_2), (s_1', s_2', ((a, b), B_2, A_2))) .$$

(3) For $(a, b) \in I_{\tilde{\Sigma}}$, $(s_1, b, B_1, s_1') \in \Delta_1$, $(s_2, a, A_2, s_2') \in \Delta_2$, and $B_2 \in \Gamma_2$, the relation Δ' contains

$$((s_1, s_2, ((a, b), B_2, A_2)), b, (B_1, B_2), (s_1', s_2', 1)) .$$

Finally, we obtain Δ from Δ' as described above.

Since the constructions from Theorem 11 can be done in at most doubly exponential time, and since the decision problems from Theorems 7–10 are at most doubly exponential, too, we obtain elementary procedures for both Question 1 and Question 2. □

6 Realizability of MSO specifications

In this section, we give a short introduction into monadic second-order (MSO) logic over nested traces. The logic is built over countably infinite supplies $\{x, y, x_1, x_2, \ldots\}$ of *first-order* and $\{X, Y, X_1, X_2, \ldots\}$ of *second-order variables*. First-order variables are interpreted as events, second-order variables as sets of events. As usual, the predicates available in MSO logic depend on the signature of a structure, which is given in terms of the distributed call-return alphabet $\tilde{\Sigma} = (\Sigma, P, \mathit{type}, \mathit{dom})$.

Definition 10 The formulas from ntMSO($\tilde{\Sigma}$) are built according to the following grammar:

$$\begin{aligned}\varphi ::= {} & a(x) \mid x \lhd_p y \mid x \lhd_{\mathsf{cr}} y \mid x = y \mid x \in X \mid \\ & \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi \mid \exists X.\varphi\end{aligned}$$

where $a \in \Sigma$, $p \in P$, x, y are first-order variables, and X is a second-order variable.

A formula φ from the logic ntMSO($\tilde{\Sigma}$) is interpreted over a nested trace $T = (E, (\lhd_p)_{p \in P}, \lhd_{\mathsf{cr}}, \lambda) \in \mathsf{NTr}(\tilde{\Sigma})$ wrt. a mapping $\mathcal{I}$. The purpose of the latter is to interpret free variables. It maps any first-order variable x to an event $\mathcal{I}(x) \in E$ and any second-order variable X to a set of events $\mathcal{I}(X) \subseteq E$. We write $T \models_{\mathcal{I}} \varphi$ if formula φ is evaluated to true when the free variables of φ are interpreted according to $\mathcal{I}$. In particular, we have $T \models_{\mathcal{I}} a(x)$ if $\lambda(\mathcal{I}(x)) = a$, $T \models_{\mathcal{I}} x \lhd_p y$ if $\mathcal{I}(x) \lhd_p \mathcal{I}(y)$, and $T \models_{\mathcal{I}} x \lhd_{\mathsf{cr}} y$ if $\mathcal{I}(x) \lhd_{\mathsf{cr}} \mathcal{I}(y)$. The remaining operators are interpreted as usual. When φ is a *sentence*, i.e., a formula without free variables, we omit the index $\mathcal{I}$ and simply write $T \models \varphi$. The language $L(\varphi)$ is then defined as $\{T \in \mathsf{NTr}(\tilde{\Sigma}) \mid T \models \varphi\}$.

For a thorough introduction to MSO logic on (classical) words and its semantics, we refer the reader to [45].

Again, we will recall a well-known result for concurrent programs without stacks, and then lift it to the recursive setting.

Theorem 14 ([44]) *Suppose $\tilde{\Sigma}$ satisfies $\Sigma = \Sigma_{\mathit{int}}$, and let $L \subseteq \mathsf{NTr}(\tilde{\Sigma})$. The following statements are effectively equivalent:*

1. *There is an NTA $\mathcal{C}$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}) = L$.*
2. *There is a sentence $\varphi \in \text{ntMSO}(\tilde{\Sigma})$ such that $L(\varphi) = L$.*

Note that this theorem cannot be lifted to *nested* traces without imposing any restriction. This is due to the fact that NWAs (and, therefore, NTAs) are not complementable [12], while MSO logic is closed under negation. However, it will turn out that restricting to k-context/k-phase traces also helps in this case.

Of course, given a restriction $\theta \in \mathfrak{R}$, we first have to clarify what we mean by a θ*-trace*. There are at least two reasonable possibilities. For example, we may call a nested trace T a θ-trace if *all* linearizations of T are θ-words. Alternatively, we may require that *some* linearization of T is a θ-word. We will choose the latter, existential, definition, as it captures more nested traces. Note that there are similar options and definitions in the setting of *message sequence charts* under channel bounds [18,24].

Formally, we call $T \in \mathsf{NTr}(\tilde{\Sigma})$ a θ*-trace* if $\mathit{lin}(T) \cap \mathsf{NW}_\theta(\tilde{\Sigma}) \neq \emptyset$. We adopt other notations from nested words and let $\mathsf{NTr}_\theta(\tilde{\Sigma})$ be the set of θ-traces. Moreover, for an NTA $\mathcal{C}$, we let $L_\theta(\mathcal{C}) \overset{\text{def}}{=} L(\mathcal{C}) \cap \mathsf{NTr}_\theta(\tilde{\Sigma})$. For a sentence $\varphi \in \text{ntMSO}(\tilde{\Sigma})$, the set $L_\theta(\varphi)$ is defined accordingly.

Example 7 The nested trace T from Fig. 4 is a 2-phase trace (it admits the 2-phase linearization W from Fig. 6), a 4-context trace, a 3-scope trace, and an ordered trace (since its linearization W is ordered).

As a preparation of a logical characterization of NTAs, we show that they are complementable for some restrictions of nested traces. This is an analogue of Theorem 11 for NWAs. As we rely on Theorem 12, we have to restrict to the set $\mathfrak{R}^- = \{k\text{-}\mathsf{cnt}\,,\ k\text{-}\mathsf{ph} \mid k \geq 1\}$, i.e., to boundedly many contexts or phases.

Lemma 3 *Let $\theta \in \mathfrak{R}^-$ and let $\mathcal{C}$ be an NTA over $\tilde{\Sigma}$. Then, there is an NTA $\mathcal{C}'$ such that $L(\mathcal{C}') = \mathsf{NTr}_\theta(\tilde{\Sigma}) \backslash L(\mathcal{C})$.*

Proof From the given NTA $\mathcal{C}$, we get, using Lemma 1 and Theorem 11, an NWA $\mathcal{A}$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}) = \mathsf{NW}_\theta(\tilde{\Sigma}) \backslash \mathit{lin}(L(\mathcal{C}))$. Observe that $L(\mathcal{A})$ is a θ-representation. By Theorem 12, there is an NTA $\mathcal{C}'$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}') = \mathit{trace}(L(\mathcal{A}))$. One easily verifies that $L(\mathcal{C}') = \mathsf{NTr}_\theta(\tilde{\Sigma}) \backslash L(\mathcal{C})$. □

In particular, Lemma 3 implies that there is an NTA recognizing the set $\mathsf{NTr}_\theta(\tilde{\Sigma})$. The following theorem constitutes a generalization of Theorem 14 adapted to θ-traces, where $\theta \in \mathfrak{R}^-$.

Theorem 15 *The following implications are effective:*

1. *For every NTA $\mathcal{C}$ over $\tilde{\Sigma}$, there is a sentence $\varphi \in \text{ntMSO}(\tilde{\Sigma})$ such that $L(\varphi) = L(\mathcal{C})$.*
2. *Let $\theta \in \mathfrak{R}^-$. For every sentence $\varphi \in \text{ntMSO}(\tilde{\Sigma})$, there is an NTA $\mathcal{C}$ over $\tilde{\Sigma}$ such that $L(\mathcal{C}) = L_\theta(\varphi)$.*

Proof The proof of 1. follows the standard construction for the translation of automata into logic. One guesses an assignment of states and stack symbols to events in terms of existentially quantified second-order variables. Then, a first-order kernel checks if the assignments actually correspond to an accepting run.

For the proof of 2., we proceed by structural induction. Hereby, the only critical case is negation, which will be taken care of by Lemma 3. For simplicity, we suppose that there are no free variables. To get an automaton for $\neg\varphi$, suppose that we already have an NTA $\mathcal{C}$ such that $L(\mathcal{C}) = L_\theta(\varphi)$. By Lemma 3, there is an NTA $\mathcal{C}'$ such that $L(\mathcal{C}') = \mathsf{NTr}_\theta(\tilde{\Sigma})\backslash L(\mathcal{C})$. We have $L(\mathcal{C}') = L_\theta(\neg\varphi)$ so that we are done. □

We conclude this section stating that model checking NTAs against MSO properties is decidable (albeit of inherently nonelementary complexity):

Theorem 16 *The following problem is decidable:*

INSTANCE: $\tilde{\Sigma}$; NTA $\mathcal{C}$ over $\tilde{\Sigma}$; sentence $\varphi \in \text{ntMSO}(\tilde{\Sigma})$; $\theta \in \mathfrak{R}$

QUESTION: $L_\theta(\mathcal{C}) \subseteq L(\varphi)$?

Proof The proof does not rely on Theorem 15, since this would only allows us to show the result for restrictions from $\mathfrak{R}^-$. Instead, we give a direct reduction to linearizations, i.e., to the case of nested words, where underapproximate model checking is decidable.

By Lemma 1, we can construct an NWA $\mathcal{A}$ over $\tilde{\Sigma}$ such that $L(\mathcal{A}) = \mathit{lin}(L(\mathcal{C}))$. Secondly, we translate the sentence $\varphi \in \text{ntMSO}(\tilde{\Sigma})$, inductively, into an MSO formula $\tilde{\varphi}$ over nested words such that $L(\tilde{\varphi}) = \mathit{lin}(L(\varphi))$. We omit the formal definition of MSO over nested words. The only interesting case is $x \lhd_p y$, which is translated to $p(x) \wedge p(y) \wedge x < y \wedge \neg\exists z(p(z) \wedge x < z < y)$ where $p(x) \overset{\text{def}}{=} \bigvee_{a\in\Sigma_p} a(x)$ and $<$ refers to the (MSO-definable) total order induced by a nested word.

Now, we have $L_\theta(\mathcal{C}) \subseteq L(\varphi)$ iff $L_\theta(\mathcal{A}) \subseteq L(\tilde{\varphi})$. The latter problem is decidable, which follows from Theorems 7–10 and the MSO characterizations of NWAs given in [25,29,30]. □

The model-checking problem has been addressed in [7] for the phase-restriction and propositional dynamic logic (PDL) as specification language. Its complexity drops to EXPTIME when the bound on the number of phases is fixed. MSO-definable temporal logics have been considered in [10,34] for various restrictions in a sequential setting. It is shown that the model-checking problem is still elementary even when the restriction (e.g., the bound on the number of phases) is part of the input.

7 Conclusion

Most results presented in this paper rely on specific restrictions of the domain of nested words. It will be worthwhile to study a more generic setting by bounding the tree-width or split-width. The model-checking question has been well-studied in [4,13,34], but not much is known about realizability.

Note that some realizable specifications will inevitably yield implementations in terms of NWAs that are nondeterministic and suffer from deadlocks. One should, therefore, study classes of NWAs that are arguably more "realistic" meaning, in particular, that they are deterministic and/or deadlock-free [2,11,42]. A natural question is then to ask for a specification formalism that guarantees such realistic implementations.

It also remains to study realizability in the realm of recursive processes *communicating through FIFO channels* [20,26]. Here, the model-checking question is by now well understood, in particular thanks to the split-width technique [4,14]. To the best of our knowledge, realizability questions for such communicating recursive processes have not been considered yet. The quest for *controllers*, whose study has been initiated in [3], seems to be closely related.

Recently, visibly pushdown trace languages have been characterized in terms of certain cooperating distributed systems (CD-systems) [39]. There, the dependence relation is independent of any partitioning into call, return, and internal actions. It would actually be interesting to see to which extent more general distributed call-return alphabets can be adopted in our setting.

Acknowledgements We are grateful to the reviewers for the careful reading and for the many pertinent suggestions, which helped to improve the presentation of the paper considerably.

References

1. Atig MF, Bollig B, Habermehl P (2008) Emptiness of multi-pushdown automata is 2ETIME-complete. In: DLT'08, volume 5257 of LNCS. Springer, pp 121–133
2. Akshay S, Dinca I, Genest B, Stefanescu A (2013) Implementing realistic asynchronous automata. In: FSTTCS'13, volume 24 of Leibniz international proceedings in informatics. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp 213–224
3. Aiswarya C, Gastin P, Narayan Kumar K (2014) Controllers for the verification of communicating multi-pushdown systems. In: CONCUR'14, volume 8704 of LNCS. Springer, pp 297–311
4. Aiswarya C, Gastin P, Narayan Kumar K (2014) Verifying communicating multi-pushdown systems via split-width. In: ATVA'14, volume 8837 of LNCS. Springer, 1–17
5. Alur R, Madhusudan P (2009) Adding nesting structure to words. J ACM 56(3):1–43
6. Breveglieri L, Cherubini A, Citrini C, Crespi Reghizzi S (1996) Multi-push-down languages and grammars. Int J Found Comput Sci 7(3):253–292
7. Bollig B, Cyriac A, Gastin P, Zeitoun M (2014) Temporal logics for concurrent recursive programs: satisfiability and model checking. J Appl Logic 12(4):395–416
8. Bansal K, Demri S (2013) Model-checking bounded multi-pushdown systems. In: CSR'13, volume 7913 of LNCS. Springer, pp 405–417
9. Bollig B, Grindei M-L, Habermehl P (2009) Realizability of concurrent recursive programs. In: FoSSaCS'09, volume 5504 of LNCS. Springer, pp 410–424
10. Bollig B, Kuske D, Mennicke R (2013) The complexity of model checking multi-stack systems. In: LICS'13. IEEE Computer Society Press, pp 163–170
11. Baudru N, Morin R (2007) Synthesis of safe message-passing systems. In: FSTTCS'07, volume 4855 of LNCS. Springer, 277–289
12. Bollig B (2008) On the expressive power of 2-stack visibly pushdown automata. Log Methods Comput Sci 4(4:16):1–35
13. Cyriac A, Gastin P, Narayan Kumar K (2012) MSO decidability of multi-pushdown systems via split-width. In: Proceedings of CONCUR'12, volume 7454 of Lecture Notes in Computer Science. Springer, pp 547–561
14. Cyriac A (2014) Verification of communicating recursive programs via split-width. Ph.D. thesis, Laboratoire Spécification et Vérification, ENS Cachan
15. Diekert V, Rozenberg G (eds) (1995) The book of traces. World Scientific, Singapore
16. Genest B, Gimbert H, Muscholl A, Walukiewicz I (2010) Optimal Zielonka-type construction of deterministic asynchronous automata. In: ICALP'10, volume 6199 of LNCS. Springer, pp 52–63
17. Genest B, Kuske D, Muscholl A (2006) A Kleene theorem and model checking algorithms for existentially bounded communicating automata. Inf Comput 204(6):920–956
18. Genest B, Kuske D, Muscholl A (2007) On communicating automata with bounded channels. Fundamenta Informaticae 80(1–3):147–167
19. Genest B, Muscholl A (2006) Constructing exponential-size deterministic Zielonka automata. In: ICALP'06, volume 4052 of LNCS. Springer, pp 565–576
20. Heußner A, Leroux J, Muscholl A, Sutre G (2012) Reachability analysis of communicating pushdown systems. Log Methods Comput Sci 8(3:23):1–20
21. Henriksen JG, Mukund M, Narayan Kumar K, Sohoni M, Thiagarajan PS (2005) A theory of regular MSC languages. Inf Comput 202(1):1–38
22. Kuske D (2003) Regular sets of infinite message sequence charts. Inf Comput 187:80–109
23. Kuske D (2007) Weighted asynchronous cellular automata. Theor Comput Sci 374(1–3):127–148
24. Lohrey M, Muscholl A (2004) Bounded MSC communication. Inf Comput 189(2):160–181

25. La Torre S, Madhusudan P, Parlato G (2007) A robust class of context-sensitive languages. In: LICS'07. IEEE Computer Society Press, pp 161–170
26. La Torre S, Madhusudan P, Parlato G (2008) Context-bounded analysis of concurrent queue systems. In: Proceedings of TACAS'08, volume 4963 of LNCS. Springer, pp 299–314
27. La Torre S, Madhusudan P, Parlato G (2008) An infinite automaton characterization of double exponential time. In: CSL'08, volume 5213 of LNCS. Springer, 33–48
28. La Torre S, Napoli M (2011) Reachability of multistack pushdown systems with scope-bounded matching relations. In: CONCUR'11, volume 6901 of LNCS. Springer, pp 203–218
29. La Torre S, Napoli M, Parlato G (2014) Scope-bounded pushdown languages. In DLT'14, volume 8633 of LNCS. Springer, pp 116–128
30. La Torre S, Napoli M, Parlato G (2014) A unifying approach for multistack pushdown automata. In: MFCS'14, volume 8634 of LNCS, pages . Springer, 377–389
31. La Torre S, Parlato G (2012) Scope-bounded multistack pushdown systems: fixed-point, sequentialization, and tree-width. In: FSTTCS'12, volume 18 of Leibniz international proceedings in informatics. Leibniz-Zentrum für Informatik, pp 173–184
32. Lal A, Touili T, Kidd N, Reps TW (2008) Interprocedural analysis of concurrent programs under a context bound. In: TACAS'08, volume 4963 of LNCS. Springer, pp 282–298
33. Mazurkiewicz A (1977) Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University
34. Mennicke R (2014) Model checking concurrent recursive programs using temporal logics. In: MFCS'14, volume 8634 of LNCS. Springer, pp 438–450
35. Muscholl A, Peled D (1999) Message sequence graphs and decision problems on Mazurkiewicz traces. In: MFCS'99, volume 1672 of LNCS. Springer, pp 81–91
36. Madhusudan P, Parlato G (2011) The tree width of auxiliary storage. In: POPL'11. ACM, pp 283–294
37. Muscholl A (1994) Über die Erkennbarkeit unendlicher Spuren. Ph.D. thesis, Institut für Informatik, Universität Stuttgart
38. Ochmański E (1995) Recognizable trace languages. In: Diekert V, Rozenberg G (eds) The book of traces, chapter 6. World Scientific, Singapore, pp 167–204
39. Otto F (2015) On visibly pushdown trace languages. In: SOFSEM'15, volume 8939 of Lecture Notes in Computer Science. Springer, pp 389–400
40. Peled D, Wilke Th, Wolper P (1998) An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. Theor Comput Sci 195(2):183–203
41. Qadeer S, Rehof J (2005) Context-bounded model checking of concurrent software. In: TACAS'05, volume 3440 of LNCS. Springer, 93–107
42. Stefanescu A, Esparza J, Muscholl A (2003) Synthesis of distributed algorithms using asynchronous automata. In: CONCUR'03, volume 2761 of LNCS. Springer, pp 27–41
43. Stefanescu A (2006) Automatic synthesis of distributed transition systems. Ph.D. thesis, University of Stuttgart
44. Thomas W (1990) On logical definability of trace languages. In: Proceedings of algebraic and syntactic methods in computer science (ASMICS), Report TUM-I9002, Technical University of Munich, pp 172–182
45. Thomas W (1997) Languages, automata and logic. In: Salomaa A, Rozenberg G (eds) Handbook of formal languages, vol 3. Springer, Berlin, pp 389–455
46. Zielonka W (1987) Notes on finite asynchronous automata. RAIRO Informatique Théorique et Applications 21:99–135