Reasoning About Distributed Systems: WYSIWYG*

Aiswarya Cyriac[1] and Paul Gastin[2]

1 **Uppsala University, Sweden**
`aiswarya.cyriac@it.uu.se`
2 **LSV, ENS Cachan, CNRS, INRIA, France**
`paul.gastin@lsv.ens-cachan.fr`

Abstract

There are two schools of thought on reasoning about distributed systems: one following interleaving-based semantics, and one following partial-order/graph-based semantics. This paper compares these two approaches and argues in favour of the latter. An introductory treatment of the split-width technique is also provided.

1998 ACM Subject Classification F.1.1 [Computation by Abstract Devices]: Models of Computation, F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs

Keywords and phrases Verification of distributed systems, Communicating recursive programs, Partial order/graph semantics, Split-width and tree interpretation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.11

Category Invited Talk

1 Introduction

Distributed systems form a crucially important but particularly challenging domain. Designing correct distributed systems is demanding, and verifying its correctness is even more so. The main cause of difficulty here is concurrency and interaction (or communication) between various distributed components. Hence it is important to provide a framework that makes easy the design of systems as well as their analysis. In this paper we argue in favour of (visual) graph-based techniques towards this end.

The behaviour of a distributed system is often understood by means of an interleaving-based semantics. People are guided by this understanding when designing a system, and also when formally expressing properties for system verification. But interleavings obfuscate the interactions between components. This inherent complication of interleaving-based semantics makes the design and verification vulnerable to many (human) errors. Moreover, expressing distributed properties on interleavings is non-trivial and sometimes also impossible to achieve. In contrast, a visual understanding of behaviours of distributed systems would make it less prone to errors – in the understanding of the semantics, in the statement of properties and in verification algorithms. Not only does the visual approach help in providing right intuitions, but, we will demonstrate that, it is also very powerful and efficient.

* Supported by LIA InForMel.

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 11–30

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

What you see is what you understand. A good example of the visual representation of behaviours is the ITU standard message sequence charts (MSCs) [20] which describe protocols involving message exchanges. The events executed by a local process are linearly ordered, see e. g., Figure 3. The transmission of the messages is also depicted. This visual description reveals not only the interactions between components but also the concurrency and the causality relations. The causality relation corresponds to the transitive closure of the linear orders on processes and of the message relation. Events that are not causally ordered are concurrent.

Another example which illustrates the power of visual representations is nested words [5] for the behaviours of recursive programs. The binary relation matching push-pop pairs, which is very fundamental for reasoning about recursive (or pushdown) systems, is explicitly provided in a nested word.

What you see is what you state. One main advantage of such a visual representation is the ease and power of specification. The underlying graph structure provides a richer framework for formal specification. For example, monadic second order logic (MSO) may have causality relation, concurrency relation, process ordering, message transmissions, push-pop matching relation etc. as basic predicates. Many of these fundamental relations are very difficult (or even impossible) to recover if we settle for an interleaving-based understanding. For example, the monadic second order logic over words cannot express a matching push-pop relation even when we assume a visible alphabet (one in which letter dictates whether it is a push position or a pop position). This is because such a relation requires some implicit counting for which MSO over words is too weak.

What about LTL? Temporal logics and navigation logics have been studied over the intuitive visual descriptions, for instance over nested words [4, 3], MSCs [8], nested traces [7], multiply nested words [24], etc. Such logics allow us to express the fundamental relations (causality, concurrency, message matching, push-pop matching, etc.) as basic modalities. Thus, properties of distributed systems can be easily and naturally expressed. On the other hand, if the behaviour of a distributed system is understood in terms of linear sequences of events, one is tempted to use LTL over words for specifications. But the classical modalities of LTL are not suited to the distributed setting. For example, the temporal next modality of LTL over words is nonsensical in the actual distributed behaviour since concurrent/independent events can be ordered arbitrarily by the operational semantics. So, LTL is sometimes deformed by removing the next modality [28].

When does a specification over linearizations make sense? In fact, a specification of distributed systems given over linearisations (by means of MSO or LTL) is meaningful only if it is satisfied by *all* or *none* of the equivalent linearisations of any given distributed behaviour. We say a distributed specification is *closed* if it satisfies this inevitable semantic closure condition. In some particular cases, it is decidable to check whether a given specification, e. g., in LTL over words, is closed [29, 27]. But this does not provide a convenient specification language, which would syntactically ensure that all specifications are closed. On the other hand, logics over graph-based representations naturally eliminate this problem since the semantics is independent of the linearisations.

Why care beyond reachability? Very often people care only about reachability, and not beyond it. One of the main reasons is that, in the case of sequential non-recursive systems,

the model-checking and satisfiability problems reduce to reachability. Most of the decision procedures proceed by building a machine which accepts the models of the specification. This is then followed by boolean operations on the machine model and finally performing an emptiness checking on the resulting machine which is nothing but a reachability test. However, for distributed systems, such translations from specifications to machine models do not exist. Closure under boolean operations also does not hold in general. Hence the model checking and satisfiability problems in the distributed setting cannot be reduced to reachability. Thus, while it is necessary and important to study the basic reachability problem, it is not sufficient. We need to devise techniques / verification procedures for specifications given in logical languages.

But we have techniques and tools available for words. What about graphs? In fact, graph theory is a very well-studied and mature discipline. We may use the insights and results from graph theory to our advantage. For example, generic logics on graphs serve as a good specification formalism. Graph measures such as tree-width (or clique-width or split-width) could offer good under-approximation parameters towards regaining decidability of our Turing powerful systems. The generic proof technique via tree interpretations helps us in obtaining efficient algorithms. We explore these directions in this paper with the help of split-width.

Split-width? Tree-interpretations? Split-width [16, 15, 2] offers an intuitive visual technique to decompose our behaviour graphs such as MSCs and nested words. The decomposition is mainly a divide-and-conquer technique which naturally results in a tree decomposition. Every behaviour can now be interpreted over its decomposition tree. Properties over the behaviour naturally transfer into properties over the decomposition tree. This allows us to use tree-automata techniques to obtain uniform and efficient decision procedures for a range of problems such as reachability, model checking against logical formalisms etc. Furthermore, the simple visual mechanism of split-width is as powerful as yardstick graph measures such as tree-width or clique-width. Hence it captures any class of distributed behaviours with a decidable MSO theory.

How efficient are the decision preocedures? Since graphs have a richer structure, and allow richer specifications, the verification problems are more challenging in the case of graphs as compared to words. However, our decision procedures for visual behaviours via split-width match the same time complexity as the decision procedures based on the interleaving semantics. In short, the visual technique solves more for the same price.

What you see in Table 1 is what you get. The rest of this article illustrates this.

2 Communicating Recursive Programs

We aim at analysing complex distributed systems consisting of several multithreaded recursive programs communicating via channels. In this section, we introduce the abstract model for such systems and we give its operational semantics resulting in linear behaviours. We also recall some undecidability results on these systems.

The overall structure of a system is given by its architecture, consisting of a finite set of processes, and a finite set of data structures. We are mainly interested in stack and queue data structures, though we also handle bags.

Table 1 Comparing interleavings and graphs: WYSIWYG.

WYS \ WYG	Understanding (Behaviours)	Expressiveness (Specifications)	Efficiency (Complexity of algorithms)
Words	– interleaved sequence of events. Interactions are obfuscated and very difficult to recover. – combinatorial explosion (single distributed behaviour results in a huge number of interleaved traces)	– too weak for many natural specifications – requires semantical closure to be meaningful: equivalent linear traces should agree on a specification	– undecidable in general – decidable under restrictions – reductions to sequential word automata – many tools available on the shelf – good space complexity
Graphs	– visual description of events – interactions are visible / self-explained – no combinatorial explosion	– powerful specifications. trivial to express interactions – independent of particular linearisation (i. e., naturally meaningful)	– undecidable in general – decidable under (more lenient) restrictions – reductions to tree automata via tree-interpretations – good time complexity

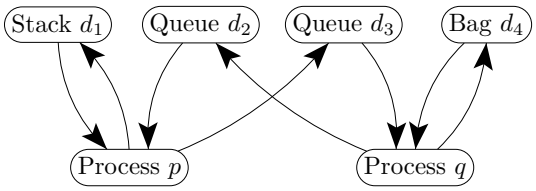

Figure 1 An Architecture. It has four data structures and two processes. Writer and Reader of the data structures are depicted by the incoming and outgoing arrows respectively.

An architecture. $\mathfrak{A}$ is a tuple $(\mathsf{Procs}, \mathsf{DS}, \mathsf{Writer}, \mathsf{Reader})$ consisting of a finite set Procs of processes, a finite set $\mathsf{DS} = \mathsf{Bags} \uplus \mathsf{Stacks} \uplus \mathsf{Queues}$ of data structures and functions Writer and Reader which assign to each data structure the process that will write into it and the process that will read from it respectively. In the special case of communicating recursive programs, we use stacks for recursion and queues for FIFO message passing. Bags are useful when no specific order is imposed on the message delivery. Since stacks are used to model recursion, we assume that $\mathsf{Writer}(s) = \mathsf{Reader}(s)$ for all $s \in \mathsf{Stacks}$. An architecture is depicted in Figure 1.

Each process is described as a finite state machine in which transitions may either be internal, only modifying the local state of the machine, or access a data structure. In the latter case, it is executing either a write event, adding a value to the data structure, or a read event, removing a value from the data structure.

Since these data structures permit only destructive reads, they induce a binary matching relation between write events and read events of a behaviour.

▶ **Definition 1.** A system of concurrent processes with data structures (CPDS) over an architecture $\mathfrak{A}$ and an alphabet Σ of action names is a tuple $\mathcal{S} = (\mathsf{Locs}, \mathsf{Val}, (\mathsf{Trans}_p)_{p\in\mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin})$ where Locs is a finite set of locations, Val is a finite set of values that can be stored in the data structures, $\ell_{\text{in}} \in \mathsf{Locs}$ is the initial location, $\mathsf{Fin} \subseteq \mathsf{Locs}^{\mathsf{Procs}}$ is the set of global final locations, and Trans_p is the set of transitions of process p. Trans_p may have write (resp. read) transitions on data structure d only if $\mathsf{Writer}(d) = p$ (resp. $\mathsf{Reader}(p) = d$). For $\ell, \ell' \in \mathsf{Locs}$, $a \in \Sigma$, $d \in \mathsf{DS}$ and $v \in \mathsf{Val}$, Trans_p has

- internal transitions of the form $\ell \xrightarrow{a} \ell'$,
- write transitions of the form $\ell \xrightarrow{a,d!v} \ell'$ with $\mathsf{Writer}(d) = p$, and
- read transitions of the form $\ell \xrightarrow{a,d?v} \ell'$ with $\mathsf{Reader}(d) = p$.

The operational semantics of a CPDS $\mathcal{S}$ may be given as an infinite state transition system $\mathcal{TS}$. The infinite set of states of $\mathcal{TS}$ is $\mathsf{Locs}^{\mathsf{Procs}} \times (\mathsf{Val}^*)^{\mathsf{DS}}$. In the following, a state of $\mathcal{TS}$ is a pair $(\overline{\ell}, \overline{z})$ where $\overline{\ell} = (\ell_p)_{p\in\mathsf{Procs}}$ and $\overline{z} = (z_d)_{d\in\mathsf{DS}}$. Such a state is initial if $\ell_p = \ell_{\text{in}}$ for all $p \in \mathsf{Procs}$ and $z_d = \varepsilon$ for all $d \in \mathsf{DS}$. It is final if $\overline{\ell} \in \mathsf{Fin}$ and $z_d = \varepsilon$ for all $d \in \mathsf{DS}$. The transitions of the CPDS $\mathcal{S}$ induce the transitions of $\mathcal{TS}$ as follows.

- $(\overline{\ell}, \overline{z}) \xrightarrow{p,a} (\overline{\ell'}, \overline{z})$ if $\ell_p \xrightarrow{a} \ell'_p$ in Trans_p and $\ell'_q = \ell_q$ for all $q \neq p$,
- $(\overline{\ell}, \overline{z}) \xrightarrow{p,a,d!} (\overline{\ell'}, \overline{z'})$ if $\ell_p \xrightarrow{a,d!v} \ell'_p$ in Trans_p for some value $v \in \mathsf{Val}$ and $\ell'_q = \ell_q$ for all $q \neq p$, and $z'_d = z_d v$, and $z'_c = z_c$ for all $c \neq d$,
- $(\overline{\ell}, \overline{z}) \xrightarrow{p,a,d?} (\overline{\ell'}, \overline{z'})$ if $\ell_p \xrightarrow{a,d?v} \ell'_p$ in Trans_p for some value $v \in \mathsf{Val}$ and $\ell'_q = \ell_q$ for all $q \neq p$, and $z'_c = z_c$ for all $c \neq d$, and $z_d = uvw$ and $z'_d = uw$ for some $u, w \in \mathsf{Val}^*$. If $d \in \mathsf{Queues}$ (resp. $d \in \mathsf{Stacks}$) we require in addition that $u = \varepsilon$ (resp. $w = \varepsilon$).

A run of $\mathcal{TS}$ is a sequence of consecutive transitions, it is accepting if it starts in the initial state and ends in some final state of $\mathcal{TS}$. The linear trace of the run is the word obtained by concatenating the sequence of transition labels. It is a word over the alphabet $\Gamma = (\mathsf{Procs} \times \Sigma) \cup (\mathsf{Procs} \times \Sigma \times \mathsf{DS} \times \{!, ?\})$. We denote by $\mathcal{L}_{\mathsf{lin}}(\mathcal{S})$ the set of linear traces accepted by the operational semantics of $\mathcal{S}$.

Keeping the data structure access in the linear traces allows us to recover the matching relation between write actions and corresponding read actions for stacks or queues (but not for bags). This requires some counting, hence it is not a regular relation (an MSO definable relation) on the linear traces.

Undecidability. Since the operational semantics is an infinite state system, we cannot analyse it directly. The most basic problem, reachability, is already undecidable. This is in particular the case for a single process with a self queue, or a single process with two stacks, or two processes with two queues between them (the direction of the queues does not matter), or two processes having a stack each and linked with a queue (see, e. g., [15]). Characterizations of decidable topologies have been studied in the case of reliable and lossy channels [12] or in the case of FIFO channels and bags [13].

Since decidable architectures are much too restrictive, under-approximation techniques have been developped. Decidability is recovered by putting some restrictions on the possible behaviours of the system. For instance, a very natural restriction is to put a bound on the data structure capacities. The operational semantics described above becomes a finite state transition system. The analysis in this case is restricted to so-called existentially bounded behaviours [18]. Many other under-approximation classes have been studied, e. g., bounded context [30], bounded phase [21], bounded scope [23], etc.

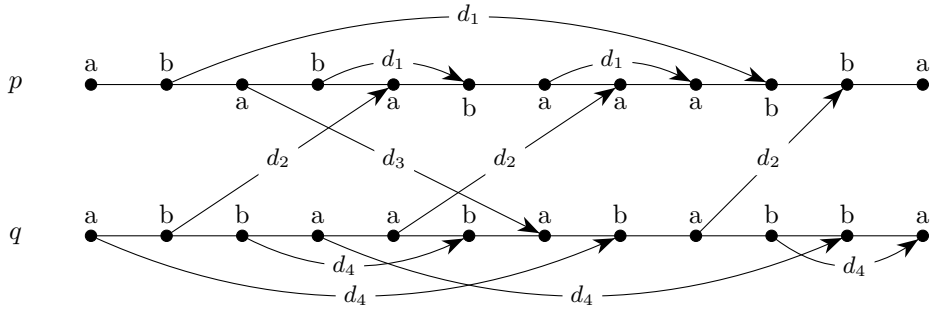

Figure 2 A CBM over the architecture given in Figure 1 and alphabet $\{a, b\}$.

3 Distributed Behaviours

In this section, we introduce the distributed semantics of CPDSs. We relate the distributed semantics and the operational semantics by showing that the linear traces arising from the latter are exactly the linearisations of the graphs defined by the former. We illustrate the benefits of considering the distributed semantics in the rest of this paper.

As a first example, the graph on Figure 2 provides a visual description of the behaviour of a CPDS. On such a graph, called concurrent behaviour with matching (CBM), the horizontal lines describe the linear behaviour of each process of the system, the other edges describe the matching relations between writes and corresponding reads. The CBM in Figure 2, is over the architecture of Figure 1. The curved arrows on process p and process q form the matching relations of stack d_1 and bag d_4 respectively. The matching relations induced by queues are shown by the arrows between the processes. As a comparison, the operational semantics generates linearisations of this behaviour such as

$$\begin{aligned}
\mathrm{Lin}_1 = {} & (p,a)(p,b,d_1,!)(p,a,d_3,!)(p,b,d_1,!)(q,a,d_4,!)(q,b,d_2,!)(q,b,d_4,!)\\
& (q,a,d_4,!)(q,a,d_2,!)(q,b,d_4,?)(q,a,d_3,?)(q,b,d_4,?)(q,a,d_2,!)\\
& (q,b,d_4,!)(q,b,d_4,?)(q,a,d_4,?)(p,a,d_2,?)(p,b,d_1,?)(p,a,d_1,!)\\
& (p,a,d_2,?)(p,a,d_1,?)(p,b,d_1,?)(p,b,d_2,?)(p,a)\\
\mathrm{Lin}_2 = {} & (p,a)(q,a,d_4,!)(p,b,d_1,!)(q,b,d_2,!)(p,a,d_3,!)(q,b,d_4,!)(p,b,d_1,!)\\
& (q,a,d_4,!)(p,b,d_1,?)(q,a,d_2,!)(p,a,d_2,?)(q,b,d_4,?)(p,a,d_1,!)\\
& (q,a,d_3,?)(p,a,d_2,?)(q,b,d_4,?)(p,a,d_1,?)(q,a,d_2,!)\\
& (p,b,d_1,?)(q,b,d_4,!)(p,b,d_2,?)(q,b,d_4,?)(p,a)(q,a,d_4,?)
\end{aligned}$$

from which it is much harder to get an intuition of the interactions going on in this behaviour. Actually, when we have bags, we cannot uniquely reconstruct a CBM from a linearisation. Hence, for distributed systems, graphs provide a visual and intuitive description of behaviours well-suited for human beings.

In the next sections, we discuss further the benefits of describing behaviours as directed graphs. Here, we define these CBMs. The intuition is easy, we have one linear trace for each process and in addition binary matching relations relating write events to corresponding reads. The formal definition has to state additional properties so that the matching relations comply with the access policies of data structures.

▶ **Definition 2.** A concurrent behaviour with matching (CBM) over architecture $\mathfrak{A}$ and alphabet Σ is a tuple $\mathcal{M} = ((w_p)_{p\in\mathsf{Procs}}, (\rhd^d)_{d\in\mathsf{DS}})$ where $w_p \in \Sigma^*$ is the sequence of events

on process p and $\rhd^d$ is the binary relation matching write events on data structure d with their corresponding read events. We let $\mathcal{E}_p = \{(p,i) \mid 1 \leq i \leq |w_p|\}$ be the set of events on process $p \in \mathsf{Procs}$ and $\mathcal{E} = \bigcup_{p\in\mathsf{Procs}} \mathcal{E}_p$. For an event $e = (p,i) \in \mathcal{E}_p$, we set $\mathsf{pid}(e) = p$ and $\lambda(e)$ be the ith letter of w_p. We write $\to$ for the successor relation on processes: $(p,i) \to (p,i+1)$ if $1 \leq i < |w_p|$.

The matching relations should comply with the architecture: $\rhd^d \subseteq \mathcal{E}_{\mathsf{Writer}(d)} \times \mathcal{E}_{\mathsf{Reader}(d)}$ for all $d \in \mathsf{DS}$ and data structure accesses are disjoint: if $e_1 \rhd^d e_2$ and $e_3 \rhd^{d'} e_4$ are different edges ($d \neq d'$ or $(e_1,e_2) \neq (e_3,e_4)$) then they are disjoint ($|\{e_1,e_2,e_3,e_4\}| = 4$). Finally, writes should precede reads, so we require the relation $< = (\to \cup \rhd)^+$ to be a strict partial order on the set $\mathcal{E}$ of events, where $\rhd = \bigcup_{d\in\mathsf{DS}} \rhd^d$ is the set of all matching edges. There are no additional constraints for bags, but for stacks or queues we have to impose in addition

- $\forall d \in \mathsf{Stacks}$, $\rhd^d$ conforms to LIFO: if $e_1 \rhd^d f_1$, $e_2 \rhd^d f_2$ and $e_1 < e_2 < f_1$ then $f_2 < f_1$,
- $\forall d \in \mathsf{Queues}$, $\rhd^d$ conforms to FIFO: if $e_1 \rhd^d f_1$, $e_2 \rhd^d f_2$ and $e_1 < e_2$ then $f_1 < f_2$.

We let $\mathsf{CBM}(\mathfrak{A},\Sigma)$ be the set of CBMs over $\mathfrak{A}$ and Σ.

A run of a CPDS $\mathcal{S}$ on a CBM $\mathcal{M}$ is simply a labelling $\rho : \mathcal{E} \to \mathsf{Locs}$ of events by states which is compatible with the transition relation. We denote by $\rho^- : \mathcal{E} \to \mathsf{Locs}$ the map that associates with each event the state which labels its predecessor: $\rho^-(e) = \rho(e')$ if $e' \to e$ and $\rho^-(e) = \ell_{\text{in}}$ if e is minimal on its process. Then, the map ρ is a run if

($\mathtt{T}_1$) for all $e \rhd^d f$, there exists some value $v \in \mathsf{Val}$ such that $\rho^-(e) \xrightarrow{\lambda(e),d!v} \rho(e)$ in $\mathsf{Trans}_{\mathsf{pid}(e)}$ and $\rho^-(f) \xrightarrow{\lambda(f),d?v} \rho(f)$ in $\mathsf{Trans}_{\mathsf{pid}(f)}$,

($\mathtt{T}_2$) for all internal events e we have $\rho^-(e) \xrightarrow{\lambda(e)} \rho(e)$ in $\mathsf{Trans}_{\mathsf{pid}(e)}$.

A run ρ is accepting if $\mathsf{last}^\rho \in \mathsf{Fin}$ where $\mathsf{last}^\rho \in \mathsf{Locs}^{\mathsf{Procs}}$ gives the final location of the run for each process: $\mathsf{last}^\rho_p = \ell_{\text{in}}$ if $\mathcal{E}_p = \emptyset$ and $\mathsf{last}^\rho_p = \rho(\max(\mathcal{E}_p))$ otherwise. We denote by $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$ the set of CBMs accepted by $\mathcal{S}$.

We explain now the relationship between the *distributed* semantics and the sequential operational semantics. Intuitively, the linear traces of the operational semantics are precisely the linearisations of the CBMs accepted by the distributed semantics. Let us make this statement more precise. Consider a CBM $\mathcal{M}$ and define the labelling $\gamma\colon \mathcal{E} \to \Gamma$ where $\Gamma = (\mathsf{Procs} \times \Sigma) \cup (\mathsf{Procs} \times \Sigma \times \mathsf{DS} \times \{!,?\})$ by $\gamma(e) = (\mathsf{pid}(e), \lambda(e))$ if e is an internal event, and if $e \rhd^d f$ then $\gamma(e) = (\mathsf{pid}(e), \lambda(e), d!)$ and $\gamma(f) = (\mathsf{pid}(f), \lambda(f), d?)$. A linearisation of $\mathcal{M}$ is given by a total order $\sqsubseteq_{\mathsf{lin}}$ on the set $\mathcal{E}$ of events which is compatible with the causality relation: $< \subseteq \sqsubseteq_{\mathsf{lin}}$. It defines the word $\gamma(e_1)\gamma(e_2)\cdots\gamma(e_n) \in \Gamma^*$ if the linear order on $\mathcal{E}$ is $e_1 \sqsubset_{\mathsf{lin}} e_2 \sqsubset_{\mathsf{lin}} \cdots \sqsubset_{\mathsf{lin}} e_n$. We denote by $\mathsf{Lin}(\mathcal{M}) \subseteq \Gamma^*$ the set of linearisations of $\mathcal{M}$.

▶ **Theorem 3.** *We have* $\mathsf{Lin}(\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})) = \mathcal{L}_{\mathsf{lin}}(\mathcal{S})$.

However, there are also subtle differences between these two semantics. As seen above, it is possible to derive the linear traces from the CBMs, but the converse is not possible in general. For instance, if the data structure d is a bag, then the linear trace $(p,a_1,d!)(p,a_2,d!)(p,a_3,d?)(p,a_4,d?)$ is both a linearisation of the CBM $\mathcal{M}_1$ which matches a_1 with a_3 and the CBM $\mathcal{M}_2$ which matches a_1 with a_4. Hence, some specifications involving the matching relations may not be expressible on the linearisations.

If we only have stacks and queues, then we can unambiguously reconstruct a CBM from a linear trace $w \in \mathcal{L}_{\mathsf{lin}}(\mathcal{S})$. This is achieved as follows. For each $p \in \mathsf{Procs}$, the sequence of actions executed by process p is the word $w_p \in \Sigma^*$ obtained as the projection on Σ of the subword of w consisting of the letters whose first component is process p. This yields the

first component $(w_p)_{p\in\mathsf{Procs}}$ of the CBM $\mathcal{M}$ associated with w. Notice that there is a natural bijection between the set of positions of w and the set $\mathcal{E}$ of events of $\mathcal{M}$. To define the matching relations, consider two events e, f associated with positions i, j of w. Then, $e \triangleright^d f$ iff the letters of w at positions i and j are $w(i) \in \mathsf{Procs}\times\Sigma\times\{d!\}$ and $w(j) \in \mathsf{Procs}\times\Sigma\times\{d?\}$ and

- if $d \in \mathsf{Queues}$ then the number of writes to d before i equals the number of reads from d before j,
- if $d \in \mathsf{Stacks}$ then j is the minimal position after i such that between i and j, the number of writes to d equals the number of reads from d.

Notice that, even in the case of stacks and queues for which the matching relations can be unambiguously recovered from linear traces, these relations are not MSO definable on the linear traces. Hence, even with the powerful MSO logic, we cannot specify properties involving matching relations on linear traces. We will discuss this more precisely in the next section.

4 Specifications

The simplest specifications consist of local state reachability or global state reachability: is there a run of $\mathcal{S}$ which reaches a given local state $\ell \in \mathsf{Locs}$ on some process $p \in \mathsf{Procs}$ or a given global state $\overline{\ell} \in \mathsf{Locs}^{\mathsf{Procs}}$. To express more elaborate properties, we need some specification languages, such as first-order logic, monadic second-order logic, temporal logic, propositional dynamic logic, etc. Here it makes a big difference whether we work with the sequential operational semantics or the distributed semantics. In the first case, traces are words and the logics will refer to the linear order $\sqsubseteq_{\mathsf{lin}}$, whereas in the latter case, behaviours are graphs and the logic will have direct access to the causal ordering $<$ as well as to the process successor relation $\rightarrow$ and the matching relations $\triangleright^d$. The process successor relation $\rightarrow$ can be easily recovered from the linear order $\sqsubseteq_{\mathsf{lin}}$. This is not the case for $\triangleright^d$, hence also for the causal ordering $<$, in the logics mentioned above. Actually, recovering a relation $\triangleright^d$ is possible if d is a stack or a queue but it requires some counting as explained above. This counting is not possible, even in the powerful MSO logic, unless the capacity of the data structure d is bounded by some fixed value B. In this case, it is possible to express $\triangleright^d$ in MSO, though the formula is non-trivial and depends on the bound B. Hence, we favour specification logics on CBMs rather than on linear traces.

The partial order $< = (\rightarrow \cup \triangleright)^+$ that comes with a CBM $\mathcal{M} = ((w_p)_{p\in\mathsf{Procs}}, (\triangleright^d)_{d\in\mathsf{DS}})$ describes the causality relation between events. Some specifications rely on this causality relation. For instance, a distributed system may receive requests on some process p, do some internal computation involving several other processes, and finally deliver an answer on some other process q. A natural specification is that every request should be answered. Indeed, the answer to a request should be in its causal future. Such a specification is easy to write on CBMs where the causal ordering is available. For instance, it corresponds to the first order formula $\varphi = \forall x\,(\mathrm{request}(x) \implies \exists y\,(x < y \wedge \mathrm{response}(y)))$ or to the local LTL formula $\mathsf{G}(\mathrm{request} \implies \mathsf{F}\,\mathrm{response})$ (where G means *for all events in the causal future* and F means *for some event in the causal future*). The CBM $\mathcal{M}$ depicted in Figure 3 does not satisfy this specification as Request 2 is not responded. Request 1 has Response 2 in its future, though not Response 1. Recall that, by Theorem 3, linear traces are linear extensions of CBMs and events that are concurent in $\mathcal{M}$ may be ordered arbitrarily in a linear trace. Therefore, $\mathsf{Lin}(\mathcal{M})$ includes many linearisations in which the responses follow the requests and from which it is not easy to see whether the specification is satisfied or not.

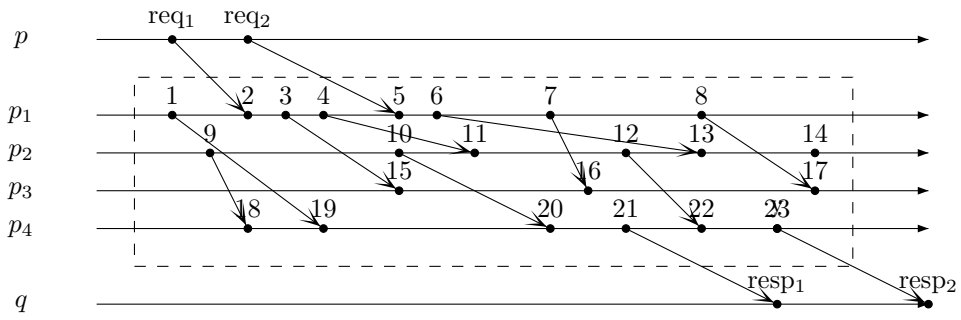

Figure 3 A request/response scenario.

As explained above, recovering the causal order $<$ from the total order $\sqsubseteq_{\mathsf{lin}}$ is not possible, even with very expressive specification languages such as MSO over words. As a conclusion, a simple and natural specification such as the request/response property, cannot be reduced to a reachability problem on the operational semantics in general. Such a reduction is possible when the data structures are restricted to bounded stacks and bounded queues (no bags), but it is non-trivial.

The same argument holds for specifications that involve the matching relation associated with a stack. For instance, we may specify that after receiving a request, the process calls a recursive procedure and when *this* call returns it *immediately* delivers the response. Again, matching a call with the corresponding return requires counting which is not a regular property unless the call depth is bounded.

As another example, a specification may require that when an access to some critical section is denied, there is a good reason for that, say some concurrent event is accessing the critical section. Again, concurrency – which is the absence of causal ordering – cannot be expressed on the linear traces in general.

We introduce below two powerful specification languages on CBMs. First, monadic second-order logic over $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ is denoted $\mathsf{MSO}(\mathfrak{A}, \Sigma)$. It follows the syntax:

$$\varphi ::= \mathsf{false} \mid a(x) \mid p(x) \mid x \leq y \mid x \rhd^d y \mid x \to y \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x\, \varphi \mid \exists X\, \varphi$$

where $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$ and $a \in \Sigma$. The semantics is as expected. Every sentence φ in MSO defines a language $\mathcal{L}_{\mathsf{cbm}}(\varphi) \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ consisting of all CBMs that satisfy that sentence. A language $L \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ is MSO definable if $L = \mathcal{L}_{\mathsf{cbm}}(\varphi)$ for some sentence $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$.

▶ Remark. The set $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ is MSO definable in the class of graphs over the signature associated with $(\mathfrak{A}, \Sigma)$. More precisely, there is an $\mathsf{MSO}(\mathfrak{A}, \Sigma)$ sentence Φ_{cbm} such that a graph $G = (\mathcal{E}, \to, (\rhd^d)_{d\in\mathsf{DS}}, \mathsf{pid}, \lambda)$ satisfies Φ_{cbm} iff it is a CBM over $(\mathfrak{A}, \Sigma)$. It is easy to obtain the formula Φ_{cbm} from Definition 2, including the LIFO and FIFO conditions for stacks and queues.

▶ Remark. The language $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$ of a CPDS $\mathcal{S}$ is definable with an existential MSO sentence $\Phi_{\mathcal{S}}$. Intuitively, with an existential prefix $\exists (X_{p,\tau})_{p\in\mathsf{Procs},\tau\in\mathsf{Trans}_p}$ the formula guesses for each transition $\tau \in \mathsf{Trans}_p$ the set of events from process $p \in \mathsf{Procs}$ that will execute this transition, and then checks with a first-order formula that this guess defines an accepting run of $\mathcal{S}$.

▶ Remark. Notice that CBM-graphs have degree bounded by 3 since any event may take part in at most one matching relation. Therefore, on CBMs, the logic MSO_2 in which we may also quantify over edges (individual variables or set variables) has the same expressive power as MSO.

We are interested in two decision problems: satisfiability of a specification and model checking of a system against a specification. Given an architecture $\mathfrak{A}$, an alphabet Σ and an MSO sentence $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$, the satisfiability problem asks whether $\mathcal{M} \models \varphi$ for some $\mathcal{M} \in \mathsf{CBM}(\mathfrak{A}, \Sigma)$. For the model checking problem, we are also given a CPDS $\mathcal{S}$ and we ask whether the specification is satisfied for all (or for some) behaviours of the system: $\mathcal{M} \models \varphi$ for all $\mathcal{M} \in \mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$.

Since reachability (or equivalently emptiness) is undecidable in general for CPDSs, both satisfiability and model checking are undecidable for any specification language that can express reachability, in particular MSO. This is trivial for model checking: the specification false is satisfied iff the language of $\mathcal{S}$ is empty, i. e., if the final states are not reachable. For satisfiability, it follows from the remark above since $\Phi_{\mathcal{S}}$ is satisfiable iff the set of final states is reachable in $\mathcal{S}$.

▶ Remark. We have seen above that reachability reduces to model-checking or to satisfiability. In the case of finite sequential systems, the converse holds since, for any MSO formula φ, we can compute an automaton $\mathcal{A}_\varphi$ which accepts exactly the models of φ [11, 17, 32]. Then, the model-checking problem reduces to the emptiness problem for the intersection of the system and the negation of the formula. But this approach fails for distributed systems because it is not possible in general to compute an automaton equivalent to a given formula. Indeed, we have already seen that the matching relation, hence also the partial order, cannot be computed by an automaton on the linearisations. Even if we stay in the distributed semantics, an MSO formula cannot be translated to a CPDS in general. This is because even the simpler class of message passing automata (i. e., when DS = Queues) is not closed under complementation [9]. Therefore the model-checking problem for CPDSs and MSO does not reduce to reachability in general.

Is MSO the ultimate logic? MSO is a very expressive logic for specifications. Its drawback is that, even when we recover decidability by restricting to some under-approximation class, the complexity of the decision procedure in non-elementary. This is already the case for words or trees. To get better complexity, one should use other formalisms such as Temporal Logics or Propositional Dynamic Logic (PDL). Towards expressing properties of CBMs, the classical LTL over words has been extended to *visible* behaviours such as nested words [4, 3], MSCs [8], nested traces [7], multiply nested words [24], etc. These temporal logics have explicit modalities that allow one to retrieve matching edges or to follow the partial order. Below, we describe propositional dynamic logic which embeds all these logics and provides powerful navigational abilities.

In PDL, there are two types of formulas. State formulas (σ) describe the properties of events in a behaviours, hence they have an implicit first-order free variable assigned to the current event. Atomic propositions such as p or a assert that the current event is on process $p \in \mathsf{Procs}$ or is labelled $a \in \Sigma$. In addition to boolean connectives, we have a path modality $\langle \pi \rangle \sigma$ claiming the existence of a path following π from the current node to an event satisfying σ. A path formula π has two implicit first-order free variables assigned to the end points of the path. They are built from basic moves following edges of the graph (in our case $\rightarrow$ and $\triangleright^d$), either forwards or backwards, using rational expressions that may use intersection in addition to the classical union, concatenation and iteration. In addition, we may check a state formula σ along a path. Formally, the syntax of $\mathsf{ICPDL}(\mathfrak{A}, \Sigma)$ is given by

$$\sigma ::= \mathsf{false} \mid p \mid a \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle \pi \rangle \sigma$$
$$\pi ::= \mathsf{test}(\sigma) \mid \rightarrow \mid \triangleright^d \mid \pi^{-1} \mid \pi + \pi \mid \pi \cap \pi \mid \pi \cdot \pi \mid \pi^*$$

where $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$ and $a \in \Sigma$. If intersection $\pi \cap \pi$ is not allowed, the fragment is PDL with converse (CPDL)[1].

5 Graph theoretic approach to verification

In this section, we show how results from graph theory may help in designing decidable under-approximation techniques for the verification of CPDSs. The distributed semantics defines the behaviours as graphs, hence we are interested in checking properties of the set of graphs $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$ accepted by a CPDS. More precisely, our aim is to solve the model checking problem: given a system $\mathcal{S}$ and a specification $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$, does $\mathcal{S} \models \varphi$, i. e., is the formula φ valid on $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$?

Let us fix some architecture $\mathfrak{A}$ and the set Σ of action labels. Decidability of the model-checking problem is equivalent to decidability of the MSO theory of the set $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ of CBM-graphs. Indeed, we have seen in Remark 4 that from a CPDS $\mathcal{S}$ we can compute a formula $\Phi_{\mathcal{S}}$ which defines $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$. Therefore, $\mathcal{S} \models \varphi$ iff $\neg\Phi_{\mathcal{S}} \vee \varphi$ is valid on $\mathsf{CBM}(\mathfrak{A}, \Sigma)$. Hence, decidability of model-checking reduces to decidability of the MSO theory of $\mathsf{CBM}(\mathfrak{A}, \Sigma)$. For the converse, it suffices to consider a universal CPDS $\mathcal{S}$ with $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) = \mathsf{CBM}(\mathfrak{A}, \Sigma)$.

We have seen in Section 2 that reachability, the most basic model-checking problem, is undecidable for CPDS, even for very simple architectures such as two processes communicating via FIFO channels (with no stacks) or a single process with two stacks. Hence, the MSO theory of $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ is undecidable in general, which can be seen also directly since CBMs have unbounded tree-width (or clique-width) in general. Still, it is extremely challenging to develop correct programs for distributed multi-threaded recursive systems. Hence, techniques for approximate verification have been extensively developed recently. We will not discuss over-approximation techniques in the present paper.

An under-approximation technique restricts the problems (reachability, satisfiability or model-checking) to a decidable subclass $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ of behaviours. Often the subclass $\mathcal{C}_m$ is parametrised with some integer m. We cover more behaviours by increasing the parameter m. The approximation family $(\mathcal{C}_m)_{m \geq 0}$ is *complete* or *exhaustive* if $\mathsf{CBM}(\mathfrak{A}, \Sigma) = \bigcup_m \mathcal{C}_m$. Hence, the aim of under-approximate verification is to define and study *meaningful* classes $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ with a decidable MSO theory.

Decidability of the MSO theory (or equivalently decidability of the MSO satisfiabiliy problem) for classes $\mathcal{C}$ of graphs has been extensively studied. We recall now some important results that will be useful for our purpose (see [14, Chapter 1] for a survey). Recall that CBM-graphs have degree bounded by 3 since any event may take part in at most one matching relation. Hence, we restrict our attention to results for classes $\mathcal{C}$ of bounded degree graphs. The following fact summarizes some of the main results (see Theorem 4).

> An MSO definable class $\mathcal{C}$ of bounded degree graphs has a decidable MSO theory iff it can be interpreted[2] in the class of binary (labelled) trees.

[1] If backward paths π^{-1} are not allowed the fragment is called PDL with intersection (IPDL). In simple PDL neither backwards paths nor intersection is allowed.

[2] There are several equivalent ways to define an interpretation of a graph $G = (V, E)$ in a labelled tree T. We describe the MSO-transductions of Courcelle, but one may, for example, also use the regular path descriptions of Engelfriet and van Oostrom. An MSO-interpretation is given by a tuple of MSO formulas. We will give a concrete example in Section 6. Intuitively, not all labelled trees admit a valid graph interpretation, hence, we use a sentence Φ_{valid} to select the "good" trees. The vertices of the graph are some nodes of the tree, and we use a formula $\Phi_{\mathsf{vertex}}(x)$ to select those nodes of T which should be interpreted as vertices of G: $V = \{u \in T \mid T \models \Phi_{\mathsf{vertex}}(u)\}$. Finally, a formula $\Phi_{\mathsf{edge}}(x, y)$ encodes

In the light of this fact, under-approximation classes are obtained by MSO definitions together with tree interpretations. Then, verification problems are reduced to problems on tree automata, yielding efficient algorithms.

Such tree interpretations can be defined specifically for some class $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$. This is for instance the case for bounded phase behaviours of multi-pushdown automata [21] where multiply nested words of bounded phase are interpreted in binary trees called stack-trees. Another example is given by the interpretation of some classes of multiply nested words in visibly (k-)path trees in order to prove decidability of emptiness and closure under complement of multi-pushdown automata when restricted to some classes of behaviours that can be interpreted in these path-trees [25] .

A higher level approach is to prove some combinatorial property on the class $\mathcal{C}$ which ensures the existence of a tree interpretation. For instance, one may show that the class $\mathcal{C}$ has bounded tree-width (and is MSO definable). This is the approach taken in [26], where decidability of several under-approximation classes is established by proving that they are MSO definable and have bounded tree-width. For most of the classes considered in [26] the decidability had been already proved directly. Hence, [26] provides a unifying approach as well as efficient algorithms based on tree interpretations.

Alternative combinatorial properties may be more convenient, for instance bounded clique-width, which is equivalent to bounded tree-width on classes of bounded degree graphs. In [16, 2] another decomposition technique, called split-width, is defined specifically for CBMs (see Section 6). On classes of CBMs, bounded tree-width, bounded clique-width and bounded split-width are all equivalent. We believe that, for a class of CBMs, establishing a bound on split-width is easier than the other measures. Also, we will see in Section 6 that split-width gives an easy and natural interpretation of CBMs in binary trees. Hence, split-width provides a convenient, necessary and sufficient condition, to establish decidability of the MSO theory of an under-approximation class.

The following theorem summarizes some of the relevant results. For more details, the reader is referred to [14, Chapter 1] and [16, 15].

▶ **Theorem 4.** *Let $\mathcal{C}$ be a class of bounded degree graphs which is* MSO *definable. TFAE*

1. *$\mathcal{C}$ has a decidable* MSO *theory,*
2. *$\mathcal{C}$ can be interpreted in binary trees,*
3. *$\mathcal{C}$ has bounded tree-width,*
4. *$\mathcal{C}$ has bounded clique-width,*
5. *$\mathcal{C}$ has bounded split-width (if $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ is a class of* CBMs*).*

In order to define a good under-approximation class, one may show that it is MSO definable and that it satisfies one of the conditions of Theorem 4. We will introduce split-width in the next section and show that it is a convenient tools for CBMs. Proving MSO definability is often easy. This is the case for many under-approximation classes, like bounded context, bounded phase, bounded scope, ordered etc. for multi-pushdown systems. Also, for distributed systems, it is easy to give an MSO definition for universally bounded MSCs, or the bounded context and well-queuing assumption of [22, 19].

---

the edge relation of G: $T \models \Phi_{\mathsf{edge}}(u, v)$ iff $(u, v) \in E$. We may also interpret vertex-labelled graphs by refining Φ_{vertex} in a tuple of formulæ $\Phi_a(x)$ which selects those vertices/nodes that are labelled a. Finally, in case of edge-labelled graphs, the formula $\Phi_{\mathsf{edge}}(x, y)$ is refined in a tuple of formulæ $\Phi_d(x, y)$ one for each edge-label d.

6 Split-width

In this section, we introduce split-width. We explain the associated tree-interpretations and infer the decidability and complexity of a collection of verification problems when parametrised by split-width. We also discuss how to use split-width in order to obtain similar results for various under-approximation classes.

With K. Narayan Kumar, we introduced split-width in [16] for multiply nested words. The technique was later extended to CBMs in [15, 2].

The idea is to decompose a graph in atomic pieces consisting of matching write/read pairs, see Figure 4. This can be seen as a two-player turn-based game with a fixed budget k which will be the width of the decomposition. The existential player (Eve), trying to prove the existence of a decomposition of width at most k, has to disconnect the CBM graph by splitting at most k process edges. For instance, the root of Figure 4 is labelled with a CBM $\mathcal{M}$ over the architecture $\mathfrak{A}$ of Figure 1. The graph $\mathcal{M}$ cannot be disconnected by splitting only one or two process edges. So Eve splits three process edges that are shown as dashed red edges in the split-CBM $\mathcal{M}'$. The universal player (Adam) will now choose one of the connected components of $\mathcal{M}'$ and the game continues. $\mathcal{M}'$ has two connected components $\mathcal{M}_1$ and $\mathcal{M}_2$, providing two choices for Adam.

If $\mathcal{M}_2$ is chosen, Eve splits the two process edges and the resulting graph $\mathcal{M}'_2$ has now two connected components. Whichever is chosen by Adam is an atomic write/read edge, which is a winning position for Eve.

Assume now that $\mathcal{M}_1$ is chosen. Note that $\mathcal{M}_1$ has two *blocks* of events on process q with one *hole* between them. The first block consists of a single event labelled b and the second one consists of three events labelled cdc. A *block* of events in a split-CBM is a maximal sequence of events on a single process. For instance, $\mathcal{M}'$ has two blocks on process p and three blocks on process q. Clearly, the number of holes on some process is the maximum of zero and the number of blocks minus one on that process. The budget k of Eve is reduced by the number of holes. For instance, $\mathcal{M}_1$ has one hole (on process q) hence Eve is only allowed two $(3-1)$ more splits to disconnect $\mathcal{M}_1$ without exceeding her budget. Her choice is depicted in $\mathcal{M}'_1$. One connected component of $\mathcal{M}'_1$ is a matching edge which is winning for Eve. So Adam should choose $\mathcal{M}_3$ lest he lose immediately. Eve splits the remaining process edge and wins regardless of Adam's choice.

To summarize, Eve wins a play if it ends in an atomic CBM, i. e., a single internal event or a matching write/read edge. She loses if she cannot disconnect a non-atomic graph without introducing more than k split-edges (holes). The split-width of a CBM is the minimum budget k for which Eve has a winning strategy. A winning strategy for Eve with budget 3 is depicted in Figure 4 for the CBM $\mathcal{M}$ at the root. As explained above, $\mathcal{M}$ cannot be disconnected with only two splits, hence its split-width is exactly 3. We denote by $\mathsf{CBM}^{k}_{\mathsf{split}}(\mathfrak{A}, \Sigma)$ the set of CBMs over $\mathfrak{A}$ and Σ with split-width bounded by k.

▶ **Example 5.** Nested words have split-width at most 2. Nested words [5] are CBMs over an architecture with a single process and a single stack. The bound on split-width can be seen easily since a nested word w is (a) either the concatenation of two nested words in which case Eve splits the edge between the two nested words, (b) or is of the form $a \overset{\frown}{\rightarrow} w \rightarrow b$ where w is a nested word, in which case Eve splits the first and last process edges, (c) or is an atomic CBM.

▶ **Example 6.** Existentially k-bounded CBMs have split-width at most $k+1$. A CBM $\mathcal{M}$ is existentially k-bounded if it admits a linearization such that the number of unmatched

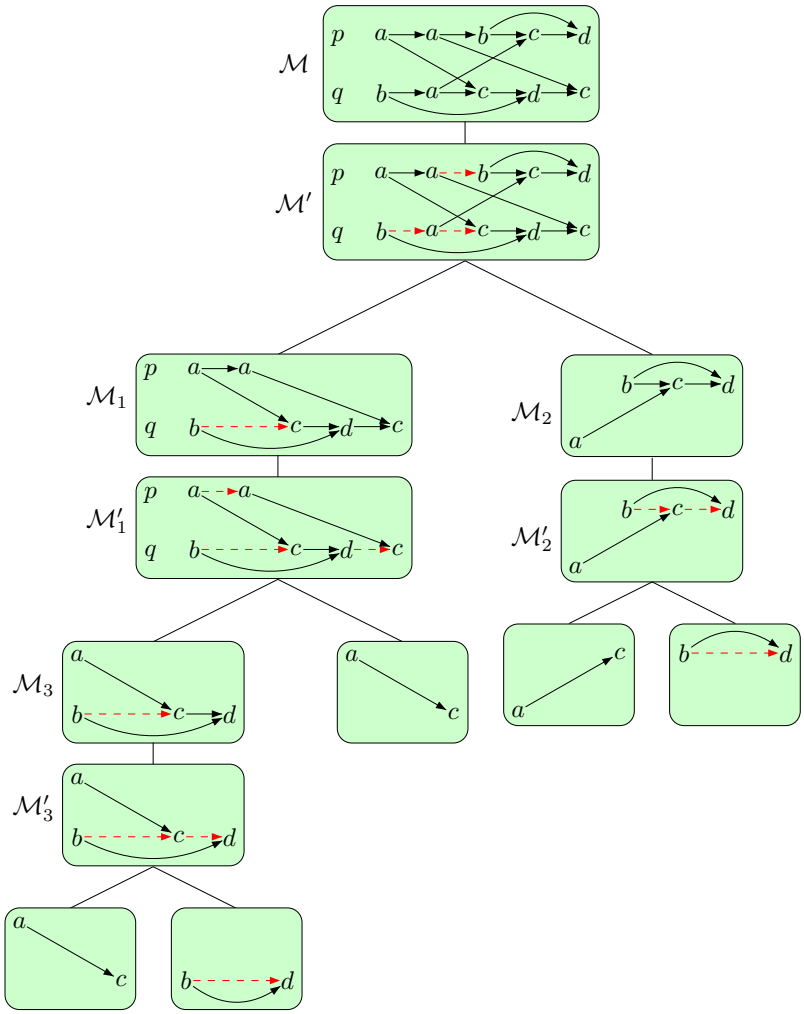

Figure 4 A split decomposition of width 3.

writes at any point is bounded by k. For instance, the CBM $\mathcal{M}$ at the root of Figure 4 is existentially 3-bounded. Let the linear order witnessing the existential bound be $\sqsubseteq_{\mathsf{lin}}$. The strategy of Eve is to detach the first $k+1$ events of $\mathcal{M}$ with respect to $\sqsubseteq_{\mathsf{lin}}$ by splitting the corresponding outgoing process edges. The resulting split-CBM $\mathcal{M}'$ must be disconnected. Indeed, the detached events cannot all be write events, otherwise the bound k is exceeded. If a read event is detached, then the corresponding write is also detached since it must come earlier in any linearization. Therefore, the split-CBM $\mathcal{M}'$ contains some connected components which are atomic CBMs and at most one connected component $\mathcal{M}_1$ which is non-atomic. Adam must choose the component $\mathcal{M}_1$ to avoid losing the game immediately. Then, Eve proceeds by splitting some more process edges until $k+1$ events are detached in the $\sqsubseteq_{\mathsf{lin}}$ order. As above, the resulting split-CBM must be disconnected and at most one of its connected component is non-atomic. Eve applies the same strategy as long as there is a non-atomic connected component.

▶ **Example 7.** Multiply nested words with at most m phases have split-width at most 2^m. Multiply nested words (MNWs) are CBMs over an architecture with a single process and several stacks. A *phase* in a MNW is a factor in which all read events are from the same stack.

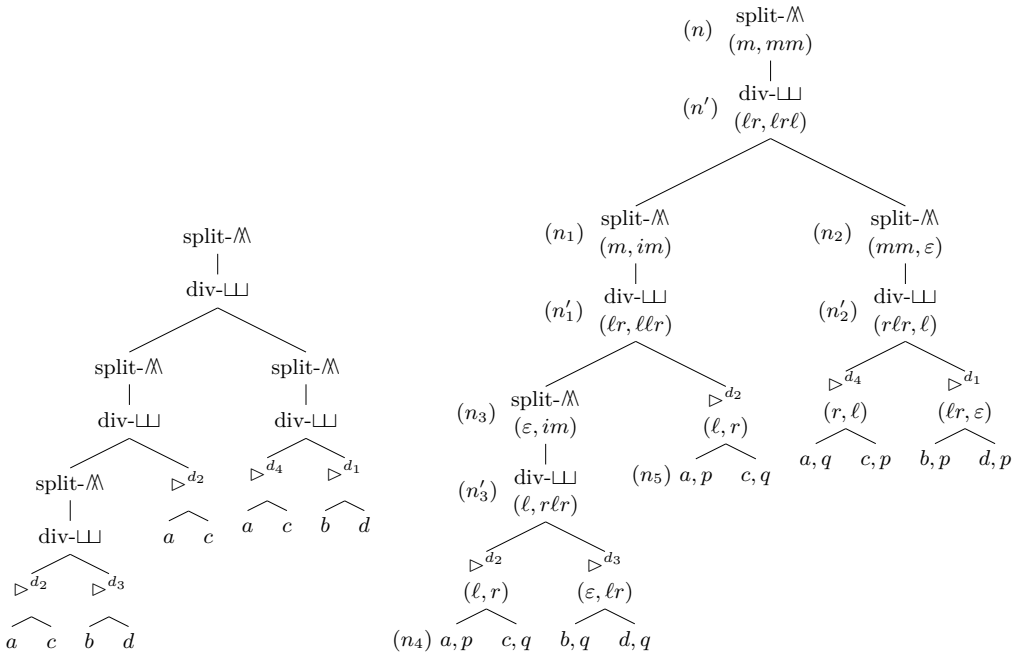

Figure 5 A split term s (left) and a labelled term t (right) corresponding to Figure 4.

Figure 6 Two CBMs that can be decomposed with the split-term s of Figure 4.

A MNW is m-phase bounded if it is the concatenation of at most m phases [21]. All m-phase bounded MNWs have split-width at most 2^m [16]. The bounded phase under-approximation has been extended to distributed systems in [15, 1].

In fact, an upper bound on split-width has been established for many under-approximation classes. See [16] for many classes of multi-pushdown systems such as bounded scope [23], ordered [10, 6], etc. For many classes of communicating (multi-pushdown) systems, see [15, 1, 2].

Split-algebra. The split-game introduced above gives the decomposition view (top-down) which is useful to establish a bound on split-width. A winning strategy of Eve for some CBM $\mathcal{M}$ can be represented with a tree as in Figure 4. Dually, there is also a split-algebra which constructs CBMs in a bottom-up fashion starting from atomic ones using two operations: shuffle (opposite of divide) and merge (opposite of split). The terms of the split-algebra over $\mathfrak{A}$ and Σ follow the syntax:

$$s ::= a \mid a \triangleright^d b \mid ⩚(s) \mid s \sqcup\!\sqcup s$$

with $a, b \in \Sigma$ and $d \in \mathsf{DS}$. The split-term s corresponding to Figure 4 is given on the left of Figure 5 (recall that the architecture is taken from Figure 1).

Several CBMs may admit a decomposition via the same split-term. For instance, the split-term s on the left of Figure 5 allows us to decompose both the CBMs $\mathcal{M}$ and $\mathcal{M}''$ of Figure 6. The main reason is that a shuffle node does not specify how the blocks of the two children are shuffled, and a merge node does not specify which holes of the child are mended into process edges. This ambiguity can be removed with an extra labelling as shown on tree t on the right of Figure 5 corresponding to the decomposition of Figure 4. At a shuffle node, the labelling consists of a tuple of words $(w_p)_{p\in\mathsf{Procs}}$, where $w_p \in \{\ell, r\}^*$ describes how the blocks on process p of the children are shuffled. For instance, the shuffle node (n') of t is labelled $(w_p, w_q) = (\ell r, \ell r \ell)$ which means that the first block of $\mathcal{M}'$ – corresponding to node (n') – on process p comes from the left child $\mathcal{M}_1$ – corresponding to (n_1) – and the second block comes from the right child $\mathcal{M}_2$ – corresponding to (n_2). On process q there are 3 blocks, first and third coming from $\mathcal{M}_1$ and second coming from $\mathcal{M}_2$. The same kind of labelling is used at $\triangleright$-nodes. Now, for a merge node, the labelling is also a tuple of words $(w_p)_{p\in\mathsf{Procs}}$, but now a word $w_p \in \{i, m\}^*$ tells whether the holes (split edges) of the child are kept as such (i for inherited) or are turned into process edges (m for mended). For instance, node (n_1) – corresponding to $\mathcal{M}_1$ – is labelled (m, im) which means that from its child $\mathcal{M}'_1$ – corresponding to (n'_1) – the hole of process p is mended, the first split edge of q is inherited and the second one is mended. Also, each leaf is labelled with its corresponding process.

Note that, the width of a decomposition can be recovered from the labelled split-term (but not from the unlabelled split-term). Indeed, the labelling of a merge node directly gives the number of holes of its child, and the labelling of a shuffle node, or a $\triangleright$-node, gives the number of blocks from which we can infer the number of holes.

Tree-interpretations. Each CBM that can be decomposed with a split-term s admits an MSO-interpretation (cf. Footnote 2) in the tree s, which is defined by a tuple of formulas over binary trees $(\Phi_{\mathsf{valid}}, (\Phi_a)_{a\in\Sigma}, (\Phi_p)_{p\in\mathsf{Procs}}, (\Phi_d)_{d\in\mathsf{DS}}, \Phi_{\to})$. The interpretation guesses (with set variables) a labelling to disambiguate the split term as explained above. Not all labelled split-terms allow an interpretation, so we use a formula Φ_{valid} to check the validity of the labelling. Essentially, we have to check that, for each process p, the number of blocks at a node is compatible with that of the children. For instance, a label $w_p \in \{\ell, r\}^*$ at a shuffle node assumes $|w_p|_\ell$ (resp. $|w_p|_r$) blocks on process p from the left (resp. right) child. A label $w_p \in \{i, m\}^*$ at a merge node assumes $|w_p|$ holes on process p from the child. For a $\triangleright^d$ node with $p = \mathsf{Writer}(d)$ and $q = \mathsf{Reader}(d)$, we request that the children are leaves labelled p (left) and q (right), and if $p = q$ then we request $w_p = \ell r$ and $w_s = \varepsilon$ for $s \neq p$, and if $p \neq q$ then we request $w_p = \ell$, $w_q = r$ and $w_s = \varepsilon$ for $p \neq s \neq q$. In addition, for stack or queue data structures, the formula Φ_{valid} has to check that the LIFO or FIFO conditions are respected. To do so, we need to enrich further the labelling. If $d \in \mathsf{Stacks}$ then we maintain the $\triangleright^d$ relation between blocks of process $p = \mathsf{Reader}(d) = \mathsf{Writer}(d)$: $i \triangleright^d j$ if $e \triangleright^d f$ for some e in the ith block of process p and some f in the jth block of process p. This information can be easily computed by a deterministic bottom-up tree automaton. At a shuffle node, we make sure that the LIFO condition is respected by rejecting shuffles that would result in a $\triangleright^d$ relation between blocks that is not well-nested. Hence we obtain an EMSO formula to check the LIFO condition for data structure d. We proceed similarly for queues.

We denote by $\mathsf{DST}^k_{\mathsf{valid}}$ the set of split-terms *disambiguated* by a *valid* labelling of width at most k. Each tree $t \in \mathsf{DST}^k_{\mathsf{valid}}$ encodes a unique CBM of split-width at most k, denoted $\mathsf{cbm}(t)$. Conversely, every $\mathcal{M} \in \mathsf{CBM}^k_{\mathsf{split}}$ is encoded by some, often many, trees $t \in \mathsf{DST}^k_{\mathsf{valid}}$.

Vertices of $\mathsf{cbm}(t)$ are leaves of t hence we let $\Phi_{\mathsf{vertex}}(x) = \mathsf{leaf}(x)$. The vertex labelling in $\mathsf{cbm}(t)$ is the corresponding leaf labelling in t. Hence, formulæ $\Phi_a(x)$ and $\Phi_p(x)$ state that

leaf x is labelled a and p in t. The matching relation also admits a trivial interpretation: for $d \in \mathsf{DS}$, $\Phi_d(x,y)$ states that x and y are leaves with a comon father labelled $\triangleright^d$.

The process relation is slightly harder to recover. This is where the additional labelling is needed. Intuitively, $\Phi_{\to}(x,y)$ states that from leaf x it is possible to walk up the tree to some merge node m, then walk down the tree to leaf y, and that the split edge from x to y has been mended at node m. It is easy to check this property with a tree automaton and to deduce the (EMSO) formula $\Phi_{\to}(x,y)$. More precisely, the deterministic bottom-up tree automaton keeps in its states the block B_x of which x is the right-most event, and the block B_y of which y is the left-most event. It goes to an accepting state only if the hole between B_x and B_y is mended into a process edge at some merge node. For instance, the process edge from leaf (n_4) to leaf (n_5) is established at merge node (n_1).

Tree-width, clique-width and split-width. On CBMs, split-width is a measure that is very similar to clique-width or tree-width. It is shown in [16, 15] that for CBMs, a bound on split-width implies a (linear) bound on clique-width or tree-width and vice versa. More precisely, if a CBM has split-width k then it has clique-width at most $2(k+|\mathsf{Procs}|)+1$ and tree-width at most $2(k+|\mathsf{Procs}|)-1$. Conversely, if a CBM has clique-width c or tree-width t then it has split-width bounded by $2c-3$ or $120(t+1)$.

Verification procedures for bounded split-width. Most verification problems become decidable with reasonable complexity when parametrised by a bound on split-width. Intuitively, the tree-interpretation provided by split-width allows us to uniformly reduce a collection of problems on CBMs of bounded split-width to problems on trees, which are then solved with tree automata techniques.

More precisely, let $\mathcal{S}$ be a CPDS over $(\mathfrak{A},\Sigma)$ and $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$ be a specification. The model checking problem restricted to the class $\mathsf{CBM}^k_{\mathsf{split}}$ of CBMs with split-width bounded by k asks whether $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) \cap \mathsf{CBM}^k_{\mathsf{split}} \subseteq \mathcal{L}_{\mathsf{cbm}}(\varphi)$. Similarly, the emptiness problem for $\mathcal{S}$ (resp. the satisfiability problem for φ) restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ asks whether $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) \cap \mathsf{CBM}^k_{\mathsf{split}} = \emptyset$ (resp. $\mathcal{L}_{\mathsf{cbm}}(\varphi) \cap \mathsf{CBM}^k_{\mathsf{split}} \neq \emptyset$). We reduce these problems to the emptiness problem for tree automata as follows.

First, we can build a tree automaton $\mathcal{A}^k_{\mathsf{valid}}$ of size $2^{\mathcal{O}(k^2|\mathfrak{A}|)}$ which accepts $\mathsf{DST}^k_{\mathsf{valid}}$. Next, we can build a tree automaton $\mathcal{A}^k_{\mathcal{S}}$ of size $|\mathcal{S}|^{\mathcal{O}(k+|\mathsf{Procs}|)}$ which accepts a tree $t \in \mathsf{DST}^k_{\mathsf{valid}}$ if and only if $\mathsf{cbm}(t) \in \mathcal{L}_{\mathsf{cbm}}(\mathcal{S})$. Therefore, the emptiness problem for the CPDS $\mathcal{S}$ restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ reduces to the emptiness problem of the tree automaton $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{S}}$.

Now, let φ be a sentence in $\mathsf{MSO}(\mathfrak{A},\Sigma)$. Using the MSO interpretation $(\Phi_{\mathsf{valid}}, \Phi_{\mathsf{vertex}}, (\Phi_a)_{a\in\Sigma}, (\Phi_p)_{p\in\mathsf{Procs}}, (\Phi_d)_{d\in\mathsf{DS}}, \Phi_{\to})$ for k-bounded split-width, we can construct a formula $\overline{\varphi}^k$ from φ such that for all trees $t \in \mathsf{DST}^k_{\mathsf{valid}}$, we have $t \models \overline{\varphi}^k$ if and only if $\mathsf{cbm}(t) \models \varphi$. By [31], from the MSO formula $\overline{\varphi}^k$ we can construct an equivalent tree automaton $\mathcal{A}^k_{\varphi}$. Therefore, the satisfiability problem for the MSO formula φ restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ reduces to the emptiness problem of the tree automaton $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\varphi}$.

Finally, we deduce easily that $\mathcal{L}_{\mathsf{cbm}}(\mathcal{S}) \cap \mathsf{CBM}^k_{\mathsf{split}} \subseteq \mathcal{L}_{\mathsf{cbm}}(\varphi)$ if and only if $t \models \overline{\varphi}^k$ for all trees t accepted by $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{S}}$. Therefore, the model checking problem $\mathcal{S} \models \varphi$ restricted to $\mathsf{CBM}^k_{\mathsf{split}}$ reduces to the emptiness problem for the tree automaton $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\neg\varphi}$.

We have described above uniform decision procedures for an array of verification problems. We refer to [16, 15, 2] for more details and we summarise the computational complexities of these procedures in Table 2.

Table 2 Summary of the complexities for bounded split-width verification.

Problem	Complexity	
	Architecture $\mathfrak{A}$, alphabet Σ, bound k on split-width	
	being part of the input (k in unary)	being fixed
CPDS emptiness	ExpTime-Complete	PTime-Complete
CPDS inclusion or universality	2ExpTime	ExpTime-Complete
LTL/CPDL satisfiability, model checking	ExpTime-Complete	
ICPDL satisfiability or model checking	2ExpTime -Complete	
MSO satisfiability or model checking	Non-elementary	

Verification procedures for other under-approximation classes. Our approach is generic in yet another sense. Under-approximation classes which admit a bound on split-width also may benefit from the uniform decision procedures described above, provided these classes correspond to regular sets of split-terms.

More precisely, let $\mathcal{C}_m$ be an under-approximation class with $\mathcal{C}_m \subseteq \mathsf{CBM}^k_{\mathsf{split}}$. For instance, we have seen that existentially m-bounded CBMs have split-width at most $k = m+1$ (Ex. 6) and m-bounded phase MNWs have split-width at most $k = 2^m$ (Ex. 7). Assume that we can construct[3] a tree automaton $\mathcal{A}^k_{\mathcal{C}_m}$ which accepts a tree $t \in \mathsf{DST}^k_{\mathsf{valid}}$ if and only if $\mathsf{cbm}(t) \in \mathcal{C}_m$. Then, the decision procedures can be restricted to the class $\mathcal{C}_m$ with a further intersection with the tree automaton $\mathcal{A}^k_{\mathcal{C}_m}$. For instance, the emptiness problem for $\mathcal{S}$ restricted to $\mathcal{C}_m$ reduces to the emptiness problem of $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{C}_m} \cap \mathcal{A}^k_{\mathcal{S}}$. The model checking problem $\mathcal{S} \models \varphi$ restricted to $\mathcal{C}_m$ reduces to the emptiness problem of $\mathcal{A}^k_{\mathsf{valid}} \cap \mathcal{A}^k_{\mathcal{C}_m} \cap \mathcal{A}^k_{\mathcal{S}} \cap \mathcal{A}^k_{\neg\varphi}$.

Clearly, the bound k on split-width in terms of m as well as the size of $\mathcal{A}^k_{\mathcal{C}_m}$ will impact on the complexity of the decision procedures. We give below several examples.

First, nested words have split-width bounded by a constant 2, and the set of nested words can be recognised by a trivial 1-state CPDS. Hence the complexities of various problems follow the right-most column of Table 2. Notice that already for this simple case, the complexities match the corresponding lower bounds for all problems.

Next, suppose a class $\mathcal{C}_m$ admits a bound on split-width $k = \mathsf{poly}(m)$ and $\mathcal{A}^k_{\mathcal{C}_m}$ is of size[4] bounded by $2^{\mathsf{poly}(m)}$. Then the decision procedures for various problems with respect to the under-approximation class $\mathcal{C}_m$ follow the complexities given in Table 2.

This can be extended as follows. Assume the bound k on split-width of the under-approximation class $\mathcal{C}_m$ is n-fold exponential in m and that the size of the tree automaton $\mathcal{A}^k_{\mathcal{C}_m}$ is bounded by $(n+1)$-fold exponential in m (e. g., if we have a CPDS $\mathcal{S}_m$ of size $2^{\mathsf{poly}(k)}$), then the complexities given in Table 2 (left column) will be augmented by a n-fold exponentiation. For instance, the class $\mathcal{C}_m$ of m-bounded phase MNWs has split-width bounded by 2^m (Ex. 7). Also, it is trivial to get a CPDS $\mathcal{S}_m$ for $\mathcal{C}_m$ of size $\mathsf{poly}(m)$. Hence, the size of $\mathcal{A}^k_{\mathcal{C}_m} = \mathcal{A}^k_{\mathcal{S}_m}$ is $2^{\mathsf{poly}(m)}$. We deduce that the complexities given in Table 2 (left column) are augmented by one exponentiation for m-bounded phase MNWs.

Thus the verification method via split-width is uniform not only for a wide range of

[3] One way to obtain $\mathcal{A}^k_{\mathcal{C}_m}$ is to provide a CPDS $\mathcal{S}_m$ which accepts the class $\mathcal{C}_m$, then the automaton $\mathcal{A}^k_{\mathcal{S}_m}$ serves as $\mathcal{A}^k_{\mathcal{C}_m}$. Similarly, if there is a formula φ_m in $\mathsf{MSO}(\mathfrak{A}, \Sigma)$ characterising the under-approximation then the automaton $\mathcal{A}^k_{\varphi_m}$ serves as $\mathcal{A}^k_{\mathcal{C}_m}$.

[4] If $\mathcal{C}_m$ is recognised by a CPDS $\mathcal{S}_m$ of size $2^{\mathsf{poly}(k)}$, then the automaton $\mathcal{A}^k_{\mathcal{C}_m} = \mathcal{A}^k_{\mathcal{S}_m}$ is of size $2^{\mathsf{poly}(k)}$.

problems but also for a wide range of classes. The complexities stated in Table 2 match the lower-bounds for many known under-approximation classes, thus asserting the optimality of the uniform decision procedures. For details we refer to [15, Section 4.4].

Word-like. A split-decomposition is said to be word-like if for every binary node in the decomposition tree, one of its subtrees has depth bounded by a constant. In this case, we could employ word automata instead of tree automata. All behaviours of some under-approximation classes, like existentially m-bounded, admit a word-like split decomposition. For many problems, the complexity upper bounds fall to the maximal space-complexity classes contained in the time-complexity classes. For example, emptiness checking of a CPDS parametrised by (word-like) split-width k can be done in PSpace instead of ExpTime, and if k is fixed, in NLogSpace instead of PTime.

Acknowledgement. The authors thank Benedikt Bollig for many fruitful discussions and constructive comments.

References

1 C. Aiswarya, P. Gastin, and K. Narayan Kumar. Controllers for the verification of communicating multi-pushdown systems. In *CONCUR'14*, volume 8704 of *LNCS*, pages 297–311. Springer, 2014.

2 C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *ATVA'14*, volume 8837 of *LNCS*. Springer, 2014. To appear.

3 R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Log. Meth. Comput. Sci.*, 4(4:11), 2008.

4 R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.

5 R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3:16), 2009.

6 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-Complete. In *DLT'08*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.

7 B. Bollig, A. Cyriac, P. Gastin, and M. Zeitoun. Temporal logics for concurrent recursive programs: Satisfiability and model checking. *Journal of Applied Logic*, 2014. To appear.

8 B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3:16), 2010.

9 B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2):150–172, 2006.

10 L. Breveglieri, A. Cherubini, Cl. Citrini, and S. Crespi-Reghizzi. Multi-pushdown languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

11 J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

12 P. Chambart and Ph. Schnoebelen. Mixing lossy and perfect FIFO channels. In *CONCUR'08*, volume 5201 of *LNCS*, pages 340–355. Springer, 2008.

13 L. Clemente, F. Herbreteau, and G. Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In *CONCUR'14*, volume 8704 of *LNCS*, pages 281–296. Springer, 2014.

14 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

15 A. Cyriac. *Verification of Communicating Recursive Programs via Split-width.* PhD thesis, ENS Cachan, 2014. `http://www.lsv.ens-cachan.fr/~cyriac/download/Thesis_Aiswarya_Cyriac.pdf`.

16 A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR'12*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.

17 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

18 B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.

19 A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS'10*, volume 6014 of *LNCS*, pages 267–281. Springer, 2010.

20 ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, February 2011.

21 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.

22 S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS'08*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.

23 S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR'11*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

24 S. La Torre and M. Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, volume 7604 of *LNCS*, pages 225–239. Springer, 2012.

25 S. La Torre, M. Napoli, and G. Parlato. A unifying approach for multistack pushdown automata. In *MFCS'14*, volume 8634 of *LNCS*, pages 377–389. Springer, 2014.

26 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 283–294. ACM, 2011.

27 A. Muscholl. *Über die Erkennbarkeit unendlicher Spuren.* Teubner, 1996.

28 D. Peled and Th. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, 1997.

29 D. Peled, Th. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and ω-regular languages. *Theoretical Computer Science*, 195(2):183–203, 1998.

30 Sh. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

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

32 B. A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961.