MSO decidability of Multi-Pushdown Systems via Split-Width

Aiswarya Cyriac, Paul Gastin and K. Narayan Kumar

August 2012

Research report LSV-12-11

MSO decidability of Multi-Pushdown Systems via Split-Width*

Aiswarya Cyriac[1], Paul Gastin[1], and K. Narayan Kumar[2]

[1] LSV, ENS Cachan, CNRS & INRIA, France
`{cyriac,gastin}@lsv.ens-cachan.fr`
[2] Chennai Mathematical Institute, India
`kumar@cmi.ac.in`

Abstract. Multi-threaded programs with recursion are naturally modeled as multi-pushdown systems. The behaviors are represented as multiply nested words (MNWs), which are words enriched with additional binary relations for each stack matching a push operation with the corresponding pop operation. Any MNW can be decomposed by two basic and natural operations: shuffle of two sequences of factors and merge of consecutive factors of a sequence. We say that the split-width of an MNW is k if it admits a decomposition where the number of factors in each sequence is at most k. The MSO theory of MNWs with split-width k is decidable. We introduce two very general classes of MNWs that strictly generalize known decidable classes and prove their MSO decidability via their split-width and obtain comparable or better bounds of tree-width of known classes.

1 Introduction

In the recent years, the analysis of multi-threaded programs [12, 15, 18, 19, 21] has been an important topic of study in formal verification. Sequential recursive programs may be abstracted as pushdown systems and analyzed using a wide variety of techniques and results available in literature. A corresponding abstraction of multi-threaded programs yields multi-pushdown systems (MPDS), with one pushdown (or stack) per thread, where, each computational step accesses only one stack, corresponding to one thread executing one step. Such a system with just two pushdowns can simulate a turing machine. Thus, even the control state reachability problem is undecidable for these abstractions.

The main strategy to get around this undecidability is to restrict the behaviors of MPDSs in terms of how they access the different stacks. A number of such restrictions have been proposed including, bounded context switch, bounded phase, ordered multi-pushdown and bounded scope. Madhusudan and Parlato [20] provide a unifying reason for why these restrictions work, relating it to the *tree-width* of the graph underlying these runs. We propose a similar notion that arises naturally in the setting of behaviors of MPDSs.

* Supported by LIA InForMel, and DIGITEO LoCoReP.

Qadeer and Rehof [21] showed that bounding the number of switches from using one stack to another (i.e., the number of times switches between threads) yields decidability. This restriction, called bounded context switching, has been well studied [15,18,19]. A generalization of this is the bounded-phase restriction [14] — where a limit is placed on the number of phases, a phase refers to sequence of steps in which values are popped only from one designated stack while pushing is permitted on all of the stacks. A different restriction (ordered multi-pushdown) [4,5] imposes an order on the stacks and permits popping only from the non-empty stack with highest priority. In [16] a new model called bounded-scope MPDSs is proposed where number of context-switches between a push and the corresponding pop is bounded. In all these cases, the control state reachability problem is known to be decidable. Further, a stronger version known as the *global reachability problem* is also decidable — given a *regular* set of configurations compute the pre^* of this set [2,3,16,24].

Alur and Madhusudan [1] show that runs of pushdown systems can be modeled as nested words, i.e., a word with a linear order as well as an additional binary matching relation that relates pushes with corresponding pops (or calls with returns) and that the MSO theory of such words is decidable. Subsequently, this idea has been [14,20] extended to define multiply nested words (MNWs), where the linear order is augmented with a nesting relation for each stack. The MSO theory of the class of multiply nested words arising from bounded-phase behaviors is shown to be decidable. The decidability argument shows how to interpret a k-phase multiply nested word as a tree. In a latter paper [20], they refine this technique to a more general theorem. If a class of MNWs is MSO definable and in addition has bounded tree-width (*tree-width* is a measure of the complexity or connectivity of a graph) then any MSO expressible property is decidable over such a class (As a matter of fact, this is true for any class of graphs. See [7,13,23]). They show that for bounded context, bounded phase and ordered multi-pushdown restrictions of MPDSs, the class of permissible MNWs is MSO definable and has bounded tree-width. As a consequence, for instance, we get a uniform proof of the decidability of control state reachability for all these classes. This technique is also applicable to related settings described in [12,15].

As more general classes are more desirable in under-approximate verification, we propose a bigger and natural class of MPDS which is a generalization of ordered and scope bounded MPDS. We freely allow pops of both kinds in this restriction. This can be thought of as the fair runs which comply to the following scheduling policy. There is no restriction on pushes. But the corresponding pop a) has to be within fixed number of context switches from then (analogous to time-out) or b) if a) fails, then all such pop events will be ordered on a priority basis (assuming a total order on the priorities of different stacks). This class is called scope bounded or ordered return (SBO) in the paper. Thus under-approximate verification wrt. SBO is a kind of fair model checking, in which at least those runs which comply to the fair scheduling policy can be verified against some specification. A similar generalization can be thought of when the ordering policy is replaced by a bounded phase restriction. These two general classes are shown to be decidable. Note that, however, a joint generalization of ordered and phase bounded yields undecidability.

The decidability proofs for the above classes are done by showing that these classes have bounded split-width. Split-width is a measure on MNWs which is comparable to tree-width (or *clique-width*) [9, 13]. We explain the notion of split-width in the following paragraphs.

We describe a simple mechanism to construct MNWs. To do this we consider a generalization of MNWs called *split MNWs* . A split MNW may be thought of as MNWs with holes. A k-split MNW contains $k-1$ holes dividing the word into k components (there may be nesting edges that connect one part to other) which are ordered from left to right. The nesting relations must be well-nested. We define two basic and natural operations on split MNWs. The *shuffle* operation takes two split MNWs, with say m and n components and results in a $m+n$ split MNW shuffling the components of the two split MNWs while obeying the well nesting criterion for each nesting relation. The *merge* operation converts an m split MNW into a k-split MNW, $k < m$ by removing some of the holes, i.e., merging adjacent components. Each of these operations results in a set of split MNWs.

The atomic split MNWs are: the 1-split consisting of a single vertex and no edges (corresponding to an internal action) and the 2-split consisting of 2 components, each with one vertex and a nesting edge connecting them. Every MNW can be obtained from atomic split MNWs by a sequence of merge and shuffle operations. If we also insist every intermediary split MNW used in this construction to have at most k components we get the class of MNWs with split-width at most k.

We consider for any fixed k, the class of MNWs with split-width bounded by k. We show that the MSO theory of such a class can be interpreted using the MSO theory of trees and hence obtain the decidability of the MSO theory of such a class. Since MNWs are MSO definable as graphs, MSO decidability for any MSO definable class of bounded split-width MNWs follows. This applies in particular to the new classes we introduce.

As already mentioned, tree-width was used in [20] for showing decidability of emptiness. Indeed, split-width is a measure on MNWs which is similar to tree-width (or *clique-width*) [9, 13]. This, calls for a comparison of split-width to tree-width.

Split-width has a simple definition. It is defined in terms of two basic and natural operations — *shuffle* of two sequences of factors and *merge* of consecutive factors in a sequence. Thus split-width is easier to handle as these are well-tuned for MNWs, where as tree-width is defined for general graphs. This gives easier and simpler proofs.

Bound on split-width can be translated (up to a constant factor) to bound on tree-width (or clique-width). MNWs with split-width at most k have tree-width at most $2k-1$ and clique-width at most $2k+1$. For the other direction, MNWs with clique-width at most k have split-width at most $2k$. Thus we do not yet know whether we have an “equivalence” between split-width and tree-width (or clique-width).

Even though the class of bounded split-width MNWs is not known to be MSO definable, they enjoy a decidable MSO theory. Furthermore, split-width is general enough to *capture all classes of MNWs with a decidable MSO theory*, thanks to the translation from clique-width to split-width.

Thus split-width should be seen as a complementary approach which gives more insight into the structure of the MNWs which have bounded tree-width (or clique-width). The advantages of split-width are reflected in the fact that it helped in improving bounds of tree-width of known classes, and lifting up proofs from different classes to get proofs for joint generalizations.

To summarize, the contributions of this paper are manyfold. On one hand it introduces more general classes of MNWs for more accurate under-approximate verification of MPDS. It introduces the notion of split-width, a measure of complexity of MNWs, which is easier than, yet as general as tree-width or clique-width. It significantly improves the known bounds on tree-width for ordered MPDS and scope bounded MPDS.

This report is organized as follows. Section 2 recalls some preliminary notions. Section 3 gives the definition of split-width. In Section 4 various decidable classes of MNWs are formally defined, and bounds on the split-widths of these classes are obtained. Section 5 compares split-width to tree-width and clique-width. In Section 6 the MSO decidability of bounded split-width is shown.

2 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers. For $n \in \mathbb{N}$, by $[n]$ we denote the set $\{1, \ldots, n\}$. Let S be a set. For a binary relation $\mathcal{R} \subseteq S \times S$, we define *support* of $\mathcal{R}$, denoted $\mathsf{supp}(\mathcal{R})$, to be $\{x \in S \mid \text{there is some } y \in S \text{ such that } (x, y) \in \mathcal{R} \text{ or } (y, x) \in \mathcal{R}\}$.

Multiply-Nested Words (MNWs) A multiply-nested word (MNW) w over Σ is a structure $w = (\mathsf{dom}(w), \lambda, <, \curvearrowright^{1}, \ldots, \curvearrowright^{s})$ where

- $\mathsf{dom}(w)$ is the set of positions
- $\lambda : \mathsf{dom}(w) \mapsto \Sigma$ is the node labelling function
- $<$ is a total order on $\mathsf{dom}(w)$.
- For each $i \in [s]$, $\curvearrowright^{i} \subseteq <$ is a binary relation such that
  1. For $i \neq j$, $\mathsf{supp}(\curvearrowright^{i}) \cap \mathsf{supp}(\curvearrowright^{j}) = \emptyset$
  2. For all $i \in [s]$, $x \curvearrowright^{i} y \implies (\forall z\, (x \curvearrowright^{i} z \implies z = y) \wedge (z \curvearrowright^{i} y \rightarrow z = x))$
  3. For all $i \in [s]$, there do not exist $x < x' < y < y'$ such that $x \curvearrowright^{i} y$ and $x' \curvearrowright^{i} y'$

An MNW on two stacks is shown in Figure 1. We may think of an MNW as a graph whose vertices are labelled by the function λ and edges are labelled using the symbols in $\Gamma = \{<, \curvearrowright^{1}, \curvearrowright^{2}, \ldots, \curvearrowright^{s}\}$. We write $G_w^<$ to denote this (Σ, Γ)-labelled graph. The underlying word is the linear order $(\mathsf{dom}(w), \lambda, <)$ and the

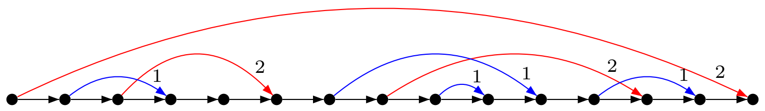

Fig. 1. An MNW

relations $\curvearrowright^i$ defines nesting edges. The conditions above imply well-nesting for each relation $\curvearrowright^i$. If $s = 1$, an MNW is simply called a *nested word* in the literature.

A different, but equivalent presentation of an MNW is to omit the ordering relation $<$ in favor of the immediate successor relation $\lessdot$. The resulting graph is denoted by $G_w^\lessdot$, with $G_w^\lessdot = (\mathsf{dom}(w), \lambda, \lessdot, \curvearrowright^1, \ldots, \curvearrowright^s)$. Here, $\lessdot$ is the immediate successor of the total order $<$ on $\mathsf{dom}(w)$. In the graph $G_w^\lessdot$ we refer to the edges labelled by $\lessdot$ as *linear edges*.

Multi-pushdown systems (MPDS) are finite state systems with a finite number of stacks. A transition may push onto a stack (push transitions), pop from a stack (pop transitions) or leave the stacks untouched. However, in one transition an MPDS can touch at most one stack. Moreover the push transitions and pop transitions are disjoint. Let Σ be the finite alphabet and $s \in \mathbb{N}$ be the number of stacks. We fix the finite alphabet Σ and the set of stacks $[s]$ for the rest of this paper.

Formally, an MPDS $\mathcal{M}$ with s stacks and alphabet Σ is a tuple $(Q, \Sigma, \iota, \Delta, F)$ where

- Q is the finite set of states,
- $\iota \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states,
- Δ is partitioned into $\Delta_{int} \uplus \Delta_{push} \uplus \Delta_{pop}$ such that
  - $\Delta_{int} \subseteq Q \times \Sigma \times Q$
  - $\Delta_{push} \subseteq Q \times \Sigma \times Q \times Q \times [s]$
  - $\Delta_{pop} \subseteq Q \times Q \times \Sigma \times Q \times [s]$

The set of configurations of $\mathcal{M}$ is $C_\mathcal{M} = Q \times (Q^*)^s$. A configuration $(q, w_1, \ldots, w_s)$ in $C_\mathcal{M}$ indicates that the current state is q and the current contents of stack i is w_i. The only initial configuration is $(\iota, \varepsilon, \ldots, \varepsilon)$. The set of final configurations are given by $(q, \varepsilon, \ldots, \varepsilon)$ where $q \in F$. We define the relation $\rightarrow \subseteq C_\mathcal{M} \times \Sigma \times C_\mathcal{M}$ as follows:

- $(q, w_1, \ldots, w_s) \xrightarrow{a} (q', w_1, \ldots, w_s)$ if $(q, a, q') \in \Delta_{int}$
- $(q, w_1, \ldots, w_i, \ldots w_s) \xrightarrow{a} (q', w_1, \ldots, q''w_i, \ldots, w_s)$ if $(q, a, q', q'', i) \in \Delta_{push}$
- $(q, w_1, \ldots, q''w_i, \ldots w_s) \xrightarrow{a} (q', w_1, \ldots, w_i, \ldots, w_s)$ if $(q, q'', a, q', i) \in \Delta_{pop}$

A run ρ of $\mathcal{M}$ on a word $a_1 a_2 \ldots a_n$ is a sequence $c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} c_2 \ldots \xrightarrow{a_n} c_n$. A run ρ is accepting if c_0 is initial and c_n is final. Note that acceptance require that all the stacks are empty. Thus every push in an accepting run has a corresponding pop.

Indeed we can add the information on when-and-how-the-stacks-were-accessed onto the word. These enriched words are multiply-nested words (MNWs). Suppose the transition taken while moving from c_{m-1} to c_m is a push on stack i and the corresponding pop is while moving from $c_{m'-1}$ to $c_{m'}$. Then we relate the positions m and m' of the word by the nesting relation $\curvearrowright^{i}$. Note that the contents of stack i at c_{m-1} is exactly the same as that at $c_{m'}$. The contents of stack i at c_{m-1} is strictly shorter than that at c_j for all $m \leq j < m'$. For all other stacks k, the contents of stack k at c_{m-1} is exactly that same as that at c_m and the contents of stack k at $c_{m'-1}$ is exactly that same as that at $c_{m'}$.

Note that MNWs take information from an accepting run of a word to add the nesting relations. Since a word can have several accepting runs, it is possible to have several MNWs corresponding to the same underlying word. However, there is only one MNW corresponding to an accepting run. The language of an MPDS $\mathcal{M}$ is defined to be the set of all MNWS such that there is a corresponding accepting run of $\mathcal{M}$.

MSO over MNWs We assume that we have an infinite supply of first-order variables $x, y, \ldots$ and second-order variables $X, Y, \ldots$. First order variables vary over positions of an MNW while second order variables vary over subsets of positions. The syntax of the monadic second order logic over MNWs is as follows:

$$\varphi ::= a(x) \mid x \in X \mid x \curvearrowright^{i} y \mid x < y \mid x = y \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $a \in \Sigma$ and $i \in [s]$. We assume familiarity with logic and hence omit the obvious semantics associated with this logic.

Remark 1. The language of a Multi-pushdown system as a set of MNWs can be described in MSO. Indeed, since the well-nesting of relations $\curvearrowright^{i}$ are enforced by the definition of an MNW, it is sufficient to guess the labelling of positions by states and to check locally that the transitions functions are respected.

3 Split-width of MNWs

We will now consider MNWs with a few linear edges missing. An MNW will be split into m components if $m-1$ linear edges are missing. Such a structure is called an m-split MNW, or simply an m-split. We denote the missing edges by '$\dashrightarrow$' and non-missing edges by '$\rightarrow$'.

Given an MNW $w = (\mathsf{dom}(w), \lambda, \lessdot, \curvearrowright^{1}, \ldots, \curvearrowright^{s})$, an *$m$-split of w* is a structure $\overline{w} = (\mathsf{dom}(w), \lambda, \rightarrow, \dashrightarrow, \curvearrowright^{1}, \ldots, \curvearrowright^{s})$ where $\rightarrow \cap \dashrightarrow = \emptyset$, $\rightarrow \cup \dashrightarrow = \lessdot$ and $|\dashrightarrow| = m-1$. The intuition is that the $\dashrightarrow$-edges are *missing* and these missing edges divide the linear order into m linear components (though there may be nesting edges connecting these different components).

A *split multiply nested word (SMNW)* is an m-split $\overline{w}$ of some MNW w and some m. We say that $\overline{w}$ is an m-SMNW. An entire multiply nested word is always a 1-SMNW. Notice that SMNWs continue to have the well nesting property for each $\curvearrowright^{i}$ w.r.t. the linear order generated by $\rightarrow \cup \dashrightarrow$.

Let $\overline{u} = (\mathsf{dom}(u), \lambda_u, \rightarrow_u, \dashrightarrow_u, \overset{1}{\curvearrowright}_u, \ldots, \overset{s}{\curvearrowright}_u)$ be an m-SMNW and let $\overline{v} = (\mathsf{dom}(v), \lambda_v, \rightarrow_v, \dashrightarrow_v, \overset{1}{\curvearrowright}_v, \ldots, \overset{s}{\curvearrowright}_v)$ be an n-SMNW. The *shuffle* of $\overline{u}$ and $\overline{v}$, denoted $\overline{u} \sqcup\!\sqcup \overline{v}$ is a set of $(m+n)$-SMNWs. A $(m+n)$-SMNW $\overline{w} = (\mathsf{dom}(w), \lambda_w, \rightarrow_w, \dashrightarrow_w, \overset{1}{\curvearrowright}_w, \ldots, \overset{s}{\curvearrowright}_w) \in \overline{u} \sqcup\!\sqcup \overline{v}$ if and only if:

- $\mathsf{dom}(w) = \mathsf{dom}(u) \uplus \mathsf{dom}(v)$
- $\lambda_w = \lambda_u \uplus \lambda_v$
- $\rightarrow_w = (\rightarrow_u \cup \rightarrow_v)$
- $\overset{i}{\curvearrowright}_w = \overset{i}{\curvearrowright}_u \cup \overset{i}{\curvearrowright}_v$

An example of a shuffle operation is shown in Figure 2. Note that, by explicitly stating that $\overline{w}$ is an $(m+n)$-SMNW, we have ensured that the nesting edges in $\overline{w}$ are well nested w.r.t. the linear order generated by $\dashrightarrow_w \cup \rightarrow_w$. Note also that, $\dashrightarrow_w \not\supseteq \dashrightarrow_u \cup \dashrightarrow_v$. In fact, by alternately choosing components from $\overline{u}$ and $\overline{v}$, we can have $\dashrightarrow_w \cap (\dashrightarrow_u \cup \dashrightarrow_v) = \emptyset$.

Let $\overline{u} = (\mathsf{dom}(u), \lambda_u, \rightarrow_u, \dashrightarrow_u, \overset{1}{\curvearrowright}_u, \ldots, \overset{s}{\curvearrowright}_u)$ be an m-SMNW. The *merge* of $\overline{u}$, denoted $\mathsf{merge}(\overline{u})$, is a set of n-SMNWs for $1 \leq n < m$, obtained by replacing some $\dashrightarrow$ by $\rightarrow$ in $\overline{u}$.

Let $k \geq 2$. We define the class k-BS (for *k-bounded splits*) to be the smallest set of SMNWs closed under the following operations

- $a \in k$-BS. That is, a single node labelled a is in k-BS,
- $\begin{tikzcd}[cramped] a \arrow[r, dashed] \arrow[r, bend left, "i"] & b \end{tikzcd} \in k$-$\mathsf{BS}$. That is, two nodes labelled a and b, connected by a $\overset{i}{\curvearrowright}$-edge is a 2-SMNW which is in k-BS,
- if $\overline{u}$ is an m-SMNW in k-BS, $\overline{v}$ is an n-SMNW in k-BS and if $m + n \leq k$, then $\overline{u} \sqcup\!\sqcup \overline{v} \subseteq k$-$\mathsf{BS}$,
- if $\overline{u}$ is in k-BS, then $\mathsf{merge}(\overline{u}) \subseteq k$-$\mathsf{BS}$.

For any SMNW $\overline{w}$, if $\overline{w} \in k$-BS we say that the *split-width* of $\overline{w}$ is at most k.

Example 2. Split-width of nested words is at most three. Notice that the nonempty nested words can be generated by the grammar

$$\begin{tikzcd}[cramped] S ::= a \mid a \arrow[r] \arrow[r, bend left] & b \mid a \arrow[r] \arrow[rr, bend left] & S \arrow[r] & b \mid S \arrow[r] & S\,. \end{tikzcd}$$

a is an atomic split in 3-BS. $\begin{tikzcd}[cramped] a \arrow[r] \arrow[r, bend left] & b \end{tikzcd} \in \mathsf{merge}(\begin{tikzcd}[cramped] a \arrow[r, dashed] \arrow[r, bend left] & b \end{tikzcd})$ is also a 1-split in 3-BS. Assume any word w generated by S is a 1-split in 3-BS. Then $\begin{tikzcd}[cramped] a \arrow[r, dashed] \arrow[rr, bend left] & w \arrow[r, dashed] & b \end{tikzcd} \in \begin{tikzcd}[cramped] a \arrow[r, dashed] \arrow[r, bend left] & b \end{tikzcd} \sqcup\!\sqcup\, w$ is a 3-split in 3-BS and $\begin{tikzcd}[cramped] a \arrow[r] \arrow[rr, bend left] & w \arrow[r] & b \end{tikzcd} \in \mathsf{merge}(\begin{tikzcd}[cramped] a \arrow[r, dashed] \arrow[r, bend left] & b \end{tikzcd} \sqcup\!\sqcup\, w)$ is a 1-split in 3-BS. If u and v are 1-splits in 3-BS generated by S, then $u \longrightarrow v \in \mathsf{merge}(u \sqcup\!\sqcup v)$ is also a 1-split in 3-BS generated by S. $\square$

Further examples of classes with bounded split-width are given in the following section.

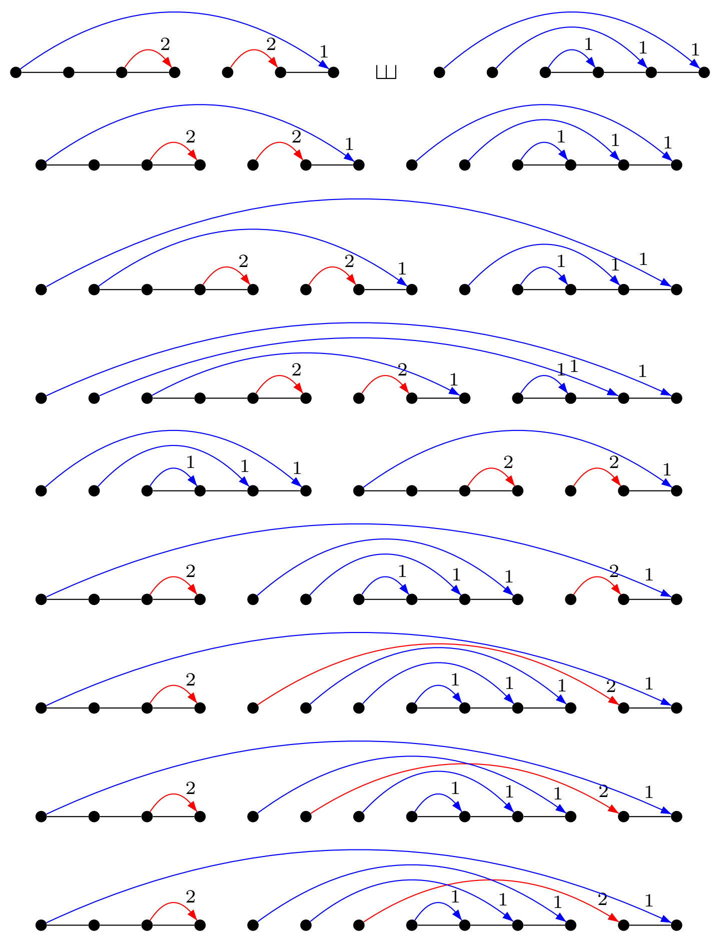

Fig. 2. There are 8 SMNWs in the shuffle defined in the first line.

4 Classes of MNWs

In this section we will see some classes of MNWs that admit bounded split-width. These classes emerge as the behaviors of MPDS conforming to some natural restrictions. These restrictions disallow arbitrary interleaving between the stacks. Most of these restrictions can be thought of as imposed by a global fair scheduler. We need the following preliminary definitions before defining the different classes.

Let w be an MNW. A *factor* u of w is defined to be a sequence of consecutive positions of w. We say that a position $x \in \mathsf{dom}(u)$ is an *i-pending call* in u if there exists $y \in \mathsf{dom}(w) \setminus \mathsf{dom}(u)$ such that $x \overset{i}{\curvearrowright} y$. Similarly, x is an *i-pending return* in u if there exists $y \in \mathsf{dom}(w) \setminus \mathsf{dom}(u)$ such that $y \overset{i}{\curvearrowright} x$. We say that u is complete for i if there are no i-pending calls or i-pending returns in u. This notion is lifted naturally to sequences of factors as well. A *context* is a set of consecutive positions which involves at most one stack.

We recall the definitions of three classes of MNWs for which MSO theory is known to be decidable and follow it with definitions of two new classes we propose.

Bounded Scope MNWs [16] We fix a parameter $m \in \mathbb{N}$. We say that an MNW is m-scope bounded if for all nesting edges, there are *no* more than m different contexts between its source and target.

Bounded Phase MNWs [14] A *phase* is a factor of an MNW in which at most one stack is allowed to return. We fix a parameter $p \in \mathbb{N}$. We say that an MNW is p-phase bounded if it can be partitioned into p phases.

Ordered MNWs [4, 5] Let $[s]$ be the set of stacks with the natural ordering on them. We say that an MNW is ordered if for all stacks $i \in [s]$, there are no pending calls of any stack $j > i$ at the target of a $\overset{i}{\curvearrowright}$ edge. In other words, if there are many pending calls at any instant, the pending calls of the highest stack will return first, then the second highest and so on. This means that, when stack i is returning, all stacks higher than i are empty.

Scope Bounded or Ordered Returns MNWs (SBO) Let $[s]$ be the set of stacks with the natural ordering on them. We fix a parameter $m \in \mathbb{N}$. Given an MNW and the parameter m, we classify the nesting edges into *long* and *short*. A nesting edge is long if there are more than m different contexts between its source and target. It is short otherwise. We say that an MNW is SBO MNW if for all stacks $i \in [s]$, there are no pending long nesting edges of any stack $j > i$ at the target of a long nesting edge of i. In other words, if there are many pending long nesting edges at any instant, the pending long nesting edges of the highest stack will return first, then the second highest and so on. That is to say that, with respect to the long nesting edges, a SBO MNW behaves exactly like an ordered MNW.

Scope or Phase Bounded Returns MNWs (SPB) Given an MNW and the parameters m and p, as in the case of SBO we classify the nesting edges into *long* and *short* (wrt. the parameter m). We say that an MNW is (m, p)-SPB if it can be partitioned into p phases wrt. the long returns.

Proposition 3. *The classes Bounded Scope, Bounded Phase, Ordered,* SBO*,* SPB *are MSO definable.*

Proof. All the returns of an MNW have to satisfy certain conditions to belong to a class. These conditions are easily MSO-definable. □

All the above classes have bounded split-width.

Theorem 4.

1. *m-Bounded scope MNWs have split-width at most $m + 2$.*
2. *p-Bounded phase MNWs have split-width at most 2^p.*
3. *Ordered MNWs have split-width at most 2^s.*
4. *m-SBO have split-width at most $2^s(2m + 1)$.*
5. *(m, p)-SPB have split-width at most $2^p(2m + 1))$.*

The proof is given in the following subsections.

4.1 Bounded Scope MNWs

Claim. m-bounded scope MNWS have split-width at most $m + 2$.

Proof. Our idea is to split the first $m - 1$ contexts of a bounded scope MNW into different components. Due to the scope bounded restriction, the calls in the first component must be returned either in the first $m-1$ components which are single contexts, or in the very first context of the last component. This allows us to obtain such a SMNW as the result of a merge of a shuffle of simpler SMNWs. It turns out that one of the SMNW which takes part in the shuffle has at most two components. We detail this idea in the following.

We write w_i to denote the ith component of an SMNW $\overline{w}$. Given an m-scope bounded MNW w, we repeatedly decompose it using the shuffle and merge operations till we are left with atomic SMNWs, ensuring that we stay within $(m + 2)$-BS in this process. We maintain the invariant:

Invariant 5 *All but the last component of the SMNWs are single contexts.*

To begin, observe that any m-scope bounded MNW w is the merge of an SMNW $\overline{w}$ with at most m components, where the first $m - 1$ components are the first $m - 1$ contexts of w. We continue by applying the following rules:

1. If some component w_i is a complete MNW, let $\overline{v} = w_i$ and $\overline{u}$ be $\overline{w}$ without w_i. Clearly $\overline{w} \in \overline{u} \sqcup \overline{v}$.
2. If some component w_i has a nonempty prefix or suffix which is a complete MNW, we split w_i into $u_i v_i$ (both nonempty) such that one of them, say v_i is a complete MNW. Let $\overline{v}$ be v_i and $\overline{u}$ be $\overline{w}$ without v_i. Clearly $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup \overline{v})$.

3. If there is a $\overset{i}{\curvearrowright}$-edge e whose source, labelled a, is the first node or last node of w_k and whose target, labelled b, is the first node or last node of w_ℓ, then $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup a \overset{i}{\dashrightarrow} b)$ where $\overline{u}$ is $\overline{w}$ without the edge e and its source and target nodes.
4. If the last component is w_j with $j < m$ and it has more than one context, then we split the first context of the last component into a separate component. Repeated application of this rule yields as many components (but at most m) as possible.

Observe that if the invariant holds for $\overline{w}$ then the same holds for the two SMNWs obtained by the application of any of these four rules. Thus the Invariant 5 is maintained. Observe that the rules preserve another invariant:

Invariant 6 *If there is a position x in ith component and a position y in jth component, then there are at least $|i - j| + 1$ different contexts between x and y in the original MNW we started with.*

We will now argue that the above operations decompose the SMNW to base cases. Suppose, for the sake of contradiction, that a non-atomic SMNW $\overline{u}$ is obtained by the above operations from $\overline{w}$ and none of the above operations are applicable.

If for any stack there is a pending return in the first $m - 1$ components, consider the first pending return which is in w_j. Let the corresponding call be in w_i $(i < j)$. Since we are not in case 2, the component w_i, which is single context, ends with this pending call and similarly w_j begins with this pending return, making case 3 applicable. Thus we may assume that in $\overline{w}$ there are no pending returns in any of the first $m - 1$ components, and there are m components if the last component has at least two contexts. Since the first $m - 1$ components cannot be complete MNWs (case 1 is not applicable) they must involve pending calls. Since they do not have complete MNWs as prefixes or suffices and are single context, each of them must begin and end with pending calls with the corresponding returns in w_m.

Claim: The first node of w_m necessarily has to be a pending return of the stack of w_1. The claim holds since a) the first context of w_m belong to the same stack as that of w_1 and also contains the pending returns called in w_1 (Otherwise there are more than m contexts switches between the first pending call and its corresponding return, thanks to Invariant 6). b) w_m cannot have a complete MNW as a prefix, as case 4 was not applicable. This makes case 3 applicable, contradicting the assumption that none of the above cases are applicable.

Notice that, just before any merge, the SMNW contains at most $m + 2$ components. □

4.2 Ordered MNWs

Claim. Ordered MNWs have split-width at most 2^s.

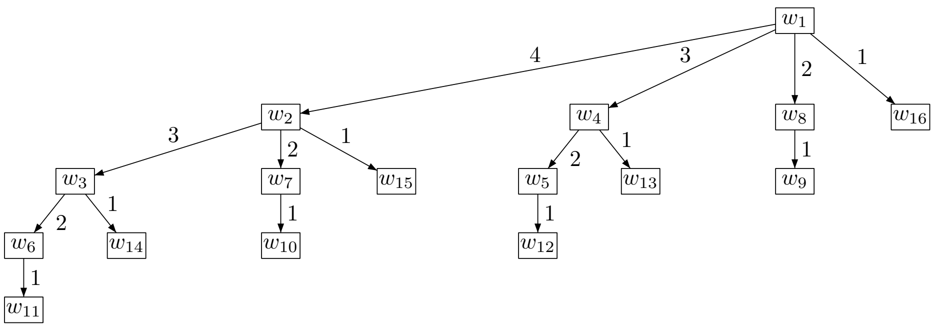

Fig. 3. A binomial tree of rank 5

Proof. We show that any ordered MNW admits a decomposition in which the SMNWs have at most 2^s components. For that, we restrict the number of components of each SMNW to 2^{s-1} before any shuffle operation. A shuffle is followed by a few merge operations so that the bound of 2^{s-1} is maintained before the next shuffle.

Our idea of splitting is as follows. We first split the given MNW into (u_1, v_1), where u_1 is the smallest prefix containing all the returns on the highest priority stack. Thus in v_1, the stack with highest priority is untouched. Then we split v_1 further into two components u_2, v_2 such that u_2 is the smallest prefix of v_1 containing all the returns from the second highest priority stack. Thus the current splitting is (u_1, u_2, v_2) with u_1 containing all the returns of the highest priority stack, u_2 containing all the returns from the second highest priority stack which are returned after the last return of the highest priority stack, and v_2 does not touch the two stacks with highest priority. Note that any pending returns in u_2 are on the second highest priority stack and are called in u_1. Then we split v_2 further into u_3, u_4, v_3 such that u_3u_4 is the smallest prefix of v_2 which contains all the returns of the third highest priority stack, and v_3 does not touch the top three stacks. All the pending returns in u_3u_4 are on stack with third highest priority. u_3u_4 is split into u_3 and u_4 such that the pending returns in u_3 are called in u_2 and the pending returns in u_4 are called in u_1. In the next round we split v_3 into u_5, u_6, u_7, u_8, v_4, and so on. It can be seen that the number of components we get finally is (2^{s-1}).

In fact the splitting we described above is sufficient for a (2^s)-split-width decomposition. The bound on the number of components can be proved by embedding it onto a tree structure which is actually a binomial tree. The details of the procedure for decomposition can also be explained directly on the embedding. This is what we do next.

The (2^{s-1})-SMNWs we obtain in the decomposition have some nice properties which let us embed them in a binomial tree of size 2^{s-1}. Each node in the binomial tree is a single component of the SMNW. The structure of the binomial tree is given in Figure 3 and is defined below.

A binomial tree is an edge labelled tree where each node has a rank. A node of rank i will have $i-1$ outgoing edges labelled with $i-1, \ldots, 1$, and the j-child

(child along the edge labeled j) will be a node of rank j. The rank of a binomial tree is the rank of its root. A binomial tree with rank k has height $k-1$ and has 2^{k-1} nodes. We identify a node by the path to that node from the root. In the figure, root is identified by ε, the leftmost node by 4321 and the rightmost node by 1. The i-child of node x is xi. Note that the rank as well as the labels along any path from the root to a leaf are decreasing.

We say that an SMNW $\overline{w}$ has a *k-binomial embedding* if every component w_i of the SMNW can be assigned a node $\mathsf{node}(i)$ of a binomial tree of rank k such that no two components are assigned to the same node. We will shortly show that an SMNW $\overline{w}$ obtained from the decomposition of an ordered MNW has an s-binomial embedding, satisfying the following invariants. We denote the s-binomial embedding of $\overline{w}$ by W. If $\mathsf{node}(i) = x$ under W, then we denote w_i by W_x in the following.

Invariant 7 *1. There is a $\overset{i}{\curvearrowright}$ edge from a component w_k to another component w_l only if $\mathsf{node}(l)$ is the i-child of $\mathsf{node}(k)$.*
2. Let x be a node of rank i. All the returns in W_x are on a stack which is at least i.

If $s = 4$, and $\overline{w}$ has 16 nonempty components, a binomial embedding satisfying the above properties may assign nodes of the binomial tree to components as shown in Figure 3. One can verify that it is in fact the only possible binomial embedding satisfying the stack policy and the ordering policy.

Any ordered MNW w is a 1-SMNW. The binomial tree embedding embeds this only component at its leftmost child (node with id $(s-1)(s-2)\cdots 1$). That is, $\overline{w} = w = W_{(s-1)(s-2)\cdots 1}$. Clearly it satisfies Invariant 7.

We show the decomposition by induction. Let $\overline{w}$ be an SMNW with a s-binomial embedding satisfying the Invariant 7. We do the following case splittings in a greedy manner (we will go to a case only if it is not possible to match any of the previous cases).

1. If there is a nesting edge $\overset{i}{\curvearrowright}$ whose source, labeled a, is the first or the last position of w_k and whose target, labeled b, is the first or the last position of w_l, then $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup a \overset{i}{\curvearrowright} b)$ where $\overline{u}$ is $\overline{w}$ without the nesting edge and its source and target nodes. Clearly $\overline{u}$ has a s-binomial embedding inherited from that of $\overline{w}$, satisfying Invariant 7.
2. If some w_i is of the form $u_i v_i$ where v_i is complete (there are no pending calls or returns in v_i) and u_i and v_i are nonempty, then $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup \overline{v})$ where $\overline{u}$ is $\overline{w}$ minus v_i and $\overline{v}$ is v_i. Also, $\overline{u}$ has a binomial embedding U inherited from $\overline{w}$ and $\overline{v}$ has a binomial embedding V which embeds its only component at its leftmost child. We have a symmetric dual case when u_i is complete. Note that $\overline{u}$ and U as well as $\overline{v}$ and V satisfies Invariant 7.
3. If W has two nonempty nodes x and y both containing no pending returns: Wlog. let y be of smaller rank if the ranks are different. Due to Item 1 of Invariant 7, we can conclude that the subtree rooted at y is disconnected from the rest. $\overline{v}$ is obtained by projecting $\overline{w}$ to those components whose embedding is in the subtree rooted at y and $\overline{u}$ is $\overline{w}$ without $\overline{v}$. Let U be a binomial embedding identical to W on the subtree rooted at y and empty elsewhere, and V be

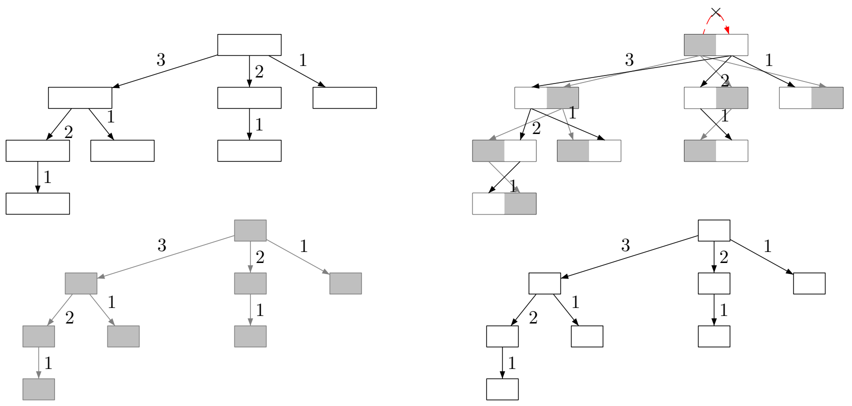

Fig. 4. Splitting of a binomial tree in case 4. Note the left/right alternation of gray (denotes U) and white (denotes V) parts along levels. This is needed since the stacks impose LIFO policy.

identical to W everywhere, except on the subtree rooted at y where it is empty. Clearly $\overline{w} \in \overline{u} \sqcup \overline{v}$. Moreover, $\overline{u}$ and U as well as $\overline{v}$ and V satisfies Invariant 7.

4. This splitting in this case is depicted in Figure 4. Let x be a non-empty node such that W_x is of the form $U_x V_x$ where U_x and V_x are non-empty, and V_x does not have any pending return. We will split its children W_{xi} as $W_{xi} = V_{xi}U_{xi}$ such that all pending returns of U_{xi} are called in U_x and those of V_{xi} are called in V_x and there are no nesting edges between U_{xi} and V_{xi}. For this we can take U_{xi} to be the shortest suffix containing all the pending returns from U_x. Note that U_x is a prefix and U_{xi} is a suffix. This is because among all the nesting edges between W_x and W_{xi} (all of them belong to stack i, thanks to Item 1 of Invariant 7), the first pending call will be returned last and the last pending call will be returned first. All the pending returns of U_{xi} should be called in U_x or V_{xi}. Since U_{xi} starts with a pending return of stack i whose call is in U_x , there are no pending returns of stack i in U_{xi} which is called in V_{xi}. Since the ordering policy on stacks is followed, there cannot be any pending returns of stack $j > i$ in U_{xi} which is called in V_{xi}. Due to Item 2 of Invariant 7, there cannot be any returns of stacks $j < i$ in U_{xi}. Thus we can split its children W_{xi} as $W_{xi} = V_{xi}U_{xi}$. SImilarly, we split recursively all nodes in the subtree of x. For all y, $W_{xy} \in \mathsf{merge}(U_{xy} \sqcup V_{xy})$ (In fact $W_{xy} = U_{xy}V_{xy}$ if $|y|$ is even, $W_{xy} = V_{xy}U_{xy}$ otherwise. For the nodes y which are not split by the above procedure, let $U_y = W_y$ and $V_y = \varepsilon$. Clearly $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup \overline{v})$ where $\overline{u}, \overline{v}$ are such that U and V are the binomial embeddings of $\overline{u}$ and $\overline{v}$. Once again, $\overline{u}$ and U as well as $\overline{v}$ and V satisfies Invariant 7.

In fact if root of W (node ε) is non empty, then one of the above cases apply. We argue why. Let $w_1 = W_\varepsilon \neq \varepsilon$. If w_1 starts with an internal action, then it is a base case, or case 2 or case 3 applies. If w_1 starts with a call to stack $j < s$, thanks to Item 2 of Invariant 7, it is either a base case or case 1 or case 2 or case 3 is applicable. If it is a call to stack s, either case 1 or case 2 or case 4 is applicable.

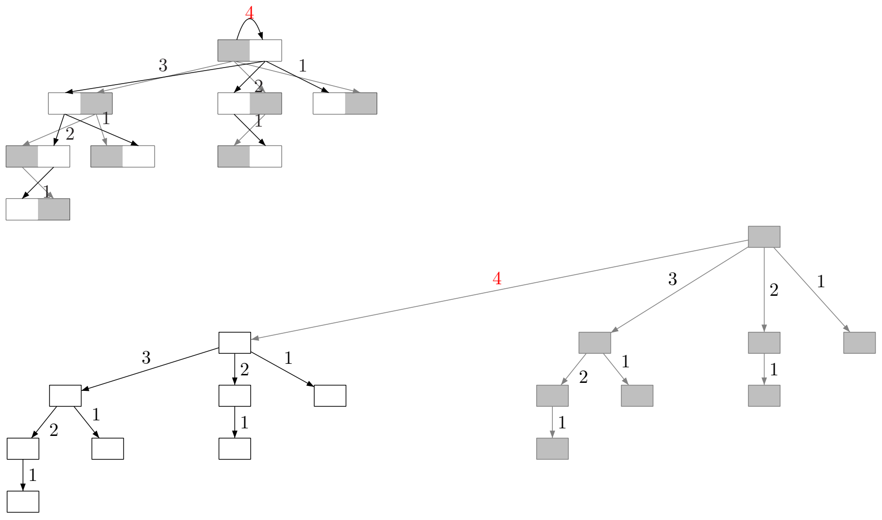

Fig. 5. Splitting of a binomial tree in case 5.

5. From the above remark, the only remaining case is when root is empty. Let xi be the nonempty node of W with the highest rank (which is i). If W_{xi} does not contain any returns of stack i then we shift node xi to x followed by a shift of nodes xiy to xy. It can be verified that shifting of the nodes gives a binomial embedding satisfying Invariant 7. Hence we can safely assume that W is a binomial embedding and xi is the nonempty node of W with highest rank and that it contains a return of stack i. Consider the first return of stack i. We split W_{xi} into $W'_x W'_{xi}$ such that W'_{xi} is the shortest suffix containing all the returns of stack i. This will result in the splitting of the children of W_{xi} which are attached to W'_x or W'_{xi} similar to that in case 4. One can verify that $\overline{w'} \in \mathsf{merge}(\overline{w'})$. Once again $\overline{w}$ and its binomial embedding W' satisfies Invariant 7. The splitting in this case is illustrated in Figure 5.

Notice that in each of the above cases, the length of the SMNW decreases, or the number of components increases (it is bounded by 2^{s-1}). Thus by induction, the proof follows. □

4.3 Bounded Phase MNWs

Claim. p-Bounded pase MNWs have split-width at most 2^p.

Proof. As was done for ordered MNWs, we show that any p-bounded phase MNW admits a decomposition in which the SMNWs have at most 2^p components. For that, we restrict the number of components of each SMNW to 2^{p-1} before any shuffle operation. A shuffle is followed by a few merge operations so that the bound of 2^{p-1} is maintained before the next shuffle.

Our idea of splitting is as follows. We first split the given MNW into (u_1, v_1), where u_1 is the first phase and v_1 is the rest. Then we split v_1 further into two components u_2, v_2 such that u_2 is the second phase and v_2 is the rest. Note that all pending returns in u_2 belong to the same stack and are called in u_1. Then we split v_2 further into u_3, u_4, v_3 such that u_3u_4 is the third phase, and v_3 is the rest. All the pending returns in u_3u_4 are on the same stack. u_3u_4 is split into u_3 and u_4 such that the pending returns in u_3 are called in u_2 and the pending returns in u_4 are called in u_1. In the next round we split v_3 into u_5, u_6, u_7, u_8 and v_4 with $u_5u_6u_7u_8$ being the fourth phase and v_4 being the rest. This procedure is repeated. It can be seen that the number of components we get finally is (2^{s-1}).

In fact the splitting we described above is sufficient for a (2^s)-split-width decomposition. The bound on the number of components can be proved by a binomial tree embedding. The binomial tree embedding and the details of the procedure are very similar to that of ordered MNWs. We will only mention the main differences from that of ordered.

For the sake of easiness, we will identify the phases in the decreasing order. That is, the first phase is called phase_p, second phase is called phase_{p-1} and so on and the last phase is called phase_1.

As in the case of ordered MNWs, our SMNWs $\overline{w}$ will have a p-binomial embedding W satisfying the invariant:

Invariant 8 *1. There is a $\curvearrowright$ edge from a component w_k to a component w_l only if $\mathsf{node}(l)$ is the i-child of $\mathsf{node}(k)$ and the return is in phase_i.*
2. If rank of x is i, then all the returns in W_x are in phase_j where $j \geq i$.

For the inductive decomposition, all the cases remain the same except for case 5. Let W_{xi} be the nonempty node of W with highest rank and assume that it contains at least one return from phase_i. We split W_{xi} into $W'_x W'_{xi}$ such that W'_{xi} is the shortest suffix containing all the returns of phase_i. The figures for ordered MNWs explains the splits for bounded phase as well, except that the edge labels of the binomial tree indicates the phase number of its children rather than the stack to which it belong. The bound follows. $\square$

4.4 SBO MNWs

Claim. m-SBO have split-width at most $2^s(2m+1)$.

Proof. The proof for this case is a generalization of that of ordered MNWs. We first split according to the long nesting edges and obtain a binomial embedding. In order to handle the short edges, we separate the outermost m contexts of these components so that a decomposition similar to that for bounded scope goes through. Thus we have a binomial tree embedding where instead of having a single component in a node of the binomial tree, we have $2m+1$ components.

We show that any SBO admits a decomposition in which the SMNWs have at most $(2^s(2m+1))$ components. For that, we restrict the number of components to $(2^{s-1}(2m+1))$ before a shuffle. A shuffle operation might add more components (but at most $(2^s(2m+1))$), but it will be immediately followed by a merge so that the bound of $(2^{s-1}(2m+1))$ is maintained.

The $(2^{s-1}(2m+1))$ SMNWs we obtain in the decomposition can be embedded in a binomial tree of size (2^{s-1}). Each node in the binomial tree is a sequence of at most $(2m+1)$ consecutive factors of the SMNW. We say that an SMNW $\overline{w}$ has an (n,k)-*binomial embedding* if every component w_i of the SMNW can be assigned a node $\mathsf{node}(i)$ of a binomial tree of rank k such that the number of components assigned to a node is not more than n. We will shortly show that a SBO SMNW $\overline{w} = (w_1, \dots w_{2^{s-1}(2m+1)})$ has a $(2m+1, s)$-binomial embedding, satisfying the following invariant. We denote the $(2m+1, s)$-binomial embedding of $\overline{w}$ by W.

Invariant 9 *1. The sequence of components (ordering of the sequence inherited from $\overline{w}$) assigned to a node are consecutive in the SMNW $\overline{w}$. Let W_x denote the sequence assigned to node x of W. The length of W_x is at most $2m+1$. In order to specify the components, we denote the sequence W_x as $W_x^{-m}, \dots, W_x^0, \dots, W_x^m$, allowing some W_x^i to be empty.*
2. W_x^j is a single context if $j \neq 0$.
3. There is a $\overset{i}{\curvearrowright}$ edge from a component w_k to a component w_l only if $\mathsf{node}(l)$ is the i-child of $\mathsf{node}(k)$ or $\mathsf{node}(l) = \mathsf{node}(k)$.
4. Let x be a node of rank i. All the long returns in W_x are on a stack which is at least i.

Given a SBO MNW w, it is a 1-SMNW. The binomial tree embedding embeds this only component at its leftmost child (node with id $(s-1)(s-2)\cdots 1$). That is, $\overline{w} = w = W^0_{(s-1)(s-2)\cdots 1}$. Clearly it satisfies Invariant 9.

We show the decomposition by induction. Let $\overline{w}$ be an SMNW with a $(2m+1, s)$-binomial embedding. We do the following case splittings in a greedy manner (we will go to a case only if it is not possible to match any of the previous cases).

1. If there is an i-nesting edge e whose source, labeled a, is the first or the last position of w_k and whose target, labeled b, is the first or the last position of w_l, then $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup a \overset{i}{\dashrightarrow} b)$ where $\overline{u}$ is $\overline{w}$ without the edge e and its source and target. Clearly $\overline{u}$ has a $(2m+1, s)$-binomial embedding U inherited from that of $\overline{w}$. $\overline{u}$ and U satisfy Invariant 9.
2. If some w_i is of the form $u_i v_i$ where v_i is complete (there are no pending calls or returns in v_i), then $\overline{w} \in \mathsf{merge}(\overline{u} \sqcup \overline{v})$ where $u_j = w_j$ and $v_j = \varepsilon$ for $j \neq i$. Also, $\overline{u}$ has a binomial embedding U inherited from $\overline{w}$ and $\overline{v}$ has a binomial embedding V which embeds its only component at its rightmost child. We have a symmetric dual case. Note that $\overline{u}$ and U as well as $\overline{v}$ and V satisfies Invariant 9.
3. If W_x^0 has more than one contexts and if some W_x^i is empty, then $W_x \in \mathsf{merge}(U_x)$ where U_x is made full by detaching the outer contexts of W_x^0. Let $U_y = W_y$. This gives $\overline{u}$ (the SMNW whose binomial embedding is U) such that $\overline{w} \in \mathsf{merge}(\overline{u})$. $\overline{u}$ and U satisfies Invariant 9. This is to deal with short nesting edges in a fashion similar to that of scope bounded case.
4. If W has two nonempty nodes x and y both containing no pending returns: Wlog. let y be of smaller rank if the ranks are different. Due to Item 3 of Invariant 9, the subtree rooted at y is disconnected from the rest. Let U, V

be such that $V_z = W_z$ and $U_z = \varepsilon$ if $z = yy'$ else $V_z = \varepsilon$ and $U_z = W_z$. Let $\overline{u}, \overline{v}$ be such that their binomial embeddings are U and V respectively. Clearly $\overline{w} \in \overline{u} \sqcup \overline{v}$. Indeed $\overline{u}$ and U and $\overline{v}$ and V satisfy Invariant 9.

5. This case is similar to Case 4 of the ordered MNWS wrt the long nesting edges. If we split W_x as $U_x V_x$ wrt the long returns, it is problematic since there can be short nesting edges from U_x to V_x. Hence we need to have cleverer fine splittings. Let the 1-split $w \in \mathsf{merge}(W_x)$ be of the form uv with both u and v non empty and with no *long* nesting edge from u to v. Let $c_1, \ldots c_m$ be the first m contexts of v such that $v = c_1 \ldots c_m v'$. Here, we will only consider the case when u and v are split inside W_x^0 since otherwise it is easier. Let $U_x^{-i} = W_x^{-i}$ if it belongs to u and $V_x^i = W_x^i$ if it belongs to v'. Let U_x^0 be the largest common prefix of W_x^0 and u, and let V_x^0 be the largest common suffix of W_x^0 and v'. Since there are no long pending calls from u to v, there can only be short pending calls, and these are bound to have their returns in some c_j.
For each stack i, let c_j contain the last return of stack i whose call is in U_x^0 or U_x^{-k}. This splits $c_j = U_x^j V_x^{-m+j-1}$ such that U_x^j is the smallest prefix containing the last stack i return from U_x^0 or U_x^{-k}. For all k such that c_k involves stack i, $U_x^k = c_k$ and $V_x^{-m+j-1} = \varepsilon$ if $k < j$, and $U_x^k = \varepsilon$ and $V_x^{-m+j-1} = c_k$ if $k > j$. This splitting wrt long edges is depicted in Figure 4. It can be verified that $\overline{u}$ and U and $\overline{v}$ and V satisfy Invariant 9.
6. Following the same reasoning given in the case 5 of ordered MNWs, we can assume that W is a binomial embedding and xi is the node of W with highest rank and that it contains a return of stack i. Consider the first return of stack i. We split W_{xi} into $W_x' W_{xi}'$ such that W_{xi}' is the shortest suffix containing all the returns of stack i. This also has the problem of crossing short nesting edges. Hence we have to do the fine splitting described above. This will result in similar repeated splitting of its children. That gives us $\overline{w} \in \mathsf{merge}(\overline{w'})$. The splitting in this case is illustrated in Figure 5. Once again it can be verified that $\overline{w}$ and its binomial embedding W' satisfy Invariant 9.
7. If root of W (node ε) is non empty, then one of the above cases apply. Hence let xi be the node of W with the highest rank (which is i). If W_{xi} does not contain any long returns of stack i then we shift node xi to x followed by a shift of nodes xiy to xy. Hence we can assume that W_{xi} contains at least one long return from stack i. Then, $W_{xi} \in \mathsf{merge}(W_x' \sqcup W_{xi}')$ where W_x' does not contain any return from stack i and all the pending returns of W_{xi}' belong to stack i and are called in W_x'. This will indeed result in recursive splitting of each W_{xiy} into W_{xiy}' and W_{xy}'. Let $\overline{w'}$ be the SMNW whose binomial embedding is W'. Clearly $\overline{w} \in \mathsf{merge}(\overline{w'})$.

As in the proof for ordered MNWs, in each of the above cases, the length of the SMNW decreases, or the number of components increases (it is bounded by 2^{s-1}). Thus by induction, the proof follows. □

4.5 SPB MNWs

Claim. (m, p)-SPB have split-width at most $2^p(2m + 1)$.

Proof. As in the case of SBO, this is a generalization of that of bounded phase MNWs. The proof is very similar to that of SBO. We will only mention the main differences from that if SBO. For the sake of easiness, we will identify the phases in the decreasing order. That is, the first phase is called phase phase_p, second phase is called phase_{p-1} and so on and the last phase is called phase_1.

As in the case of SBO, our SMNWs $\overline{w}$ will have a $(2m+1,p)$-binomial embedding W satisfying the invariant:

Invariant 10 *1. The sequence of components (ordering of the sequence inherited from $\overline{w}$) assigned to a node are consecutive in the SMNW $\overline{w}$. Let W_x denote the sequence assigned to node x of W. The length of W_x is at most $2m+1$. In order to specify the components, we denote the sequence W_x as $W_x^{-m}, \ldots, W_x^0, \ldots, W_x^m$, allowing some W_x^i to be empty.*

2. W_x^j is a single context if $j \neq 0$.

3. There is a $\curvearrowright$ edge from a component w_k to a component w_l only if $\mathsf{node}(l)$ is the i-child of $\mathsf{node}(k)$ and the return is in phase_i or $\mathsf{node}(l) = \mathsf{node}(k)$.

4. If rank of x is i, then all the long returns in W_x are in phase_j where $j \geq i$.

For the inductive decomposition, all the cases remain the same except for case 6. Let W_{xi} is the node of W with highest rank and assume that it contains at least one return from phase_i. We split W_{xi} into uv such that v is the shortest suffix containing all the returns of phase_i, followed by the fine splittings, so that $= U_{xi}$ will be W'_x and V_{xi} will be W'_{xi}. Then, $W_{xi} \in \mathsf{merge}(W'_x \sqcup W'_{xi})$ where W'_x does not contain any return from stack i and all the pending returns of W'_{xi} belong to stack i and are called in W'_x. This will indeed result in recursive splitting of each W_{xiy} into W'_{xiy} and W'_{xy}. Let $\overline{w'}$ be the SMNW whose binomial embedding is W'. Clearly $\overline{w} \in \mathsf{merge}(\overline{w'})$.

The figures for ordered MNWs explains the SMNWs for SPB as well, except that the edge labels of the binomial tree indicates the phase number of its children rather than the stack to which it belong. The bound follows. □

5 Split-width, Tree-width and Clique-width of MNWs

Split-width compares well to the usual measures of graph complexity: *tree-width* and *clique-width* [7,13,23]. This relation is stated in the following theorem.

Theorem 11. *Let w be a MNW and let $k \geq 2$.*

1. *If w has split-width k then the tree-width of $G_w^{\lessdot}$ is at most $2k-1$.*
2. *If w has split-width k then the clique-width of $G_w^{\lessdot}$ is at most $2k-1$.*
3. *If $G_w^{\lessdot}$ has clique-width k then the split-width of w is at most $2k+1$.*

It is known that any class of graphs with tree-width bounded by k has clique-width bounded by $2^{k-1}-1$ [10]. However, Item 2 gives a better bound on clique-width.

The rest of this section is devoted to the formal definitions of tree-width and clique-width and the proof of the above theorem.

By induction, we will consider graphs associated to SMNW and not only to MNW. We first define these graphs. Let $\overline{w} = (\mathsf{dom}(w), \lambda, \rightarrow, \dashrightarrow, \curvearrowright^{1}, \ldots, \curvearrowright^{s})$ be a SMNW. We ignore edges in $\dashrightarrow$ and are interested in the graphs $G_{\overline{w}}^{\rightarrow} = (\mathsf{dom}(w), \lambda, \rightarrow, \curvearrowright^{1}, \ldots, \curvearrowright^{s})$ and $G_{\overline{w}}^{\rightarrow^{+}} = (\mathsf{dom}(w), \lambda, \rightarrow^{+}, \curvearrowright^{1}, \ldots, \curvearrowright^{s})$. Note that, if $\overline{w}$ has a single component, i.e., if $\dashrightarrow$ is empty, then $G_{\overline{w}}^{\rightarrow} = G_{w}^{\lessdot}$ and $G_{\overline{w}}^{\rightarrow^{+}} = G_{w}^{<}$.

5.1 Tree-width

The concept of *tree-width* of a graph, is one of the most important measures of the algorithmic complexity of a graph and has played a fundamental role in the recent developments in graph theory. The tree-width of a graph is an integer, which intuitively measures how close a graph is to a tree (the tree-width of a graph is 1 if and only if it is a tree or a forest). We say that a class of graphs has *bounded tree-width* if there is an integer k such that the tree-width of every graph in the class is bounded above by k.

We use the algebraic characterization of tree-width as in [6]. For this we define a syntax for generating graphs.[1] Let C be a finite set of colors. Then C-tree-expressions are given by:

$$te \ ::= \ x \ \mid \ x \,\mathsf{E}\, y \ \mid \ te_1 \parallel te_2 \ \mid \ \mathsf{rnm}_{x \leftrightarrow y}(te) \ \mid \ \mathsf{fg}_{x}(te)$$

where $x, y \in C$ and E is an edge relation. In our case, E is from $\{\rightarrow, \curvearrowright^{1}, \ldots, \curvearrowright^{s}\}$. That is, for MNWs, $x \rightarrow y$, $x \curvearrowright^{i} y$ are C-tree-expressions. Each C-tree-expression defines an edge labelled graph (up to isomorphism) as described below:

- The expression x denotes the graph with a single vertex colored x.
- The expression $x \ \mathsf{E} \ y$ (with $x \neq y$)[2] denotes the graph with two vertices colored x and y and these vertices are connected by an edge E.
- The expression $te_1 \parallel te_2$ (parallel composition) denotes the disjoint union of the graphs defined by the expressions te_1 and te_2, where *the nodes with the same labels are fused.*
- The expression $\mathsf{rnm}_{x \leftrightarrow y}(te)$ (renaming) denotes the graph obtained by re-coloring the vertices colored x and y in the graph denoted by te with y and x.
- The expression $\mathsf{fg}_{x}(e)$ (forget color) denotes the graph obtained by removing the color of the vertices colored x in the graph denoted by te.

Notice that there can be at most one vertex colored x for each color x, since the parallel composition fuses nodes with the same color. Also once the color of a vertex is forgotten, that vertex cannot be colored later.

Remark 12. We have ignored the node labels in this definition, as these are not the most interesting. However, one could easily include them.

[1] This is F_{C}^{HR} in [6]

[2] We assume $x \neq y$ since we are only considering graphs without self-loops.

The tree-width of a graph is at most $|C| - 1$ if there is a C-tree-expression denoting it [6]. Using this we will now prove the following theorem.

Theorem 13. *If a SMNW $\overline{w}$ has split-width $k \geq 2$ then the tree-width of $G^{\rightarrow}_{\overline{w}}$ is at most $2k - 1$.*

Proof. There are at most k components in any SMNW of split-width at most k. We use $2k$ colors of the form b_i, e_i for $1 \leq i \leq k$. That is we fix $C = \{b_1, e_1, \ldots, b_k, e_k\}$. We maintain the invariant

Invariant 14 *Color the first node and the last node of factor i by b_i and e_i respectively. If a factor has only one node, its color is b_i.*

We show inductively how to obtain C-tree-expressions for SMNWs of split-width at most k. The base cases are the basic splits: The expression for an internal node is b_1, and that for a nesting edge on stack i is $b_1 \overset{i}{\curvearrowright} b_2$.

For $w \in u \sqcup v$: We identify the index in w of each factor in u and v. Then we do a sequence of renamings in (the C-tree-expressions of) u and v such that each node gets its intended label in w. This is followed by a simple parallel composition. Note that this parallel composition does not result in the fusion of any nodes, as the colors are disjoint. For example, consider $w = (n_1, n_2 n_3, n_4, n_5)$ and $u = (n_1, n_5)$ and $v = (n_2 n_3, n_4)$. Since u and v satisfies Invariant 14, n_1 and n_2 are colored b_1; n_5 and n_4 are colored b_2; and n_3 is colored e_1. Let te_u, te_v denote the C-tree-expressions for u and v respectively. Then $te_w = (\mathsf{rnm}_{b_2 \leftrightarrow b_4}(te_u)) \parallel (\mathsf{rnm}_{b_1 \leftrightarrow b_2}(\mathsf{rnm}_{e_1 \leftrightarrow e_2}(\mathsf{rnm}_{b_2 \leftrightarrow b_3}(te_v))))$.

For $w \in \mathsf{merge}(u)$: If w contains a linear edge from factor i in u to factor $i+1$ in u, we do a parallel composition with $(e_i \rightarrow b_{i+1})$ which is indeed a C-tree-expression. If the factor i is singleton, we do a parallel composition with $(b_i \rightarrow b_{i+1})$. We do this for each linear edge added in w. Finally, in order to maintain Invariant 14, we do a sequence of forgets and renamings. ☐

5.2 Clique Width

Another useful measure of the complexity of a graph is its *clique-width* [9]. In order to define the clique-width of a graph we need to describe a syntax for defining graphs, similarly to the one defined while describing tree-width. Let C be a set of colors. C-clique-expressions are given by

$$ce \ ::= \ x \ \mid \ ce_1 \oplus ce_2 \ \mid \ \mathsf{rnm}_{x \to y}(ce) \ \mid \ \mathsf{add}_{xEy}(ce)$$

where $x, y \in C$, and E is an edge relation. In our case, E is from $\{\rightarrow, \overset{1}{\curvearrowright}, \ldots, \overset{s}{\curvearrowright}\}$. That is, for MNWs, $\mathsf{add}_{x \rightarrow y}$, $\mathsf{add}_{x \overset{i}{\curvearrowright} y}$ are C-clique-expressions. Each expression defines an edge labelled graph (up to isomorphism) as described below:

- The expression x denotes the graph with a single vertex colored x.
- The expression $ce_1 \oplus ce_2$ denotes the disjoint union of the graphs defined by the expressions ce_1 and ce_2.

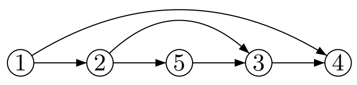

Fig. 6.

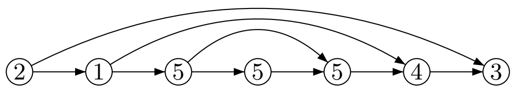

Fig. 7.

- The expression $\mathsf{rnm}_{x\to y}(ce)$ denotes the graph obtained by recoloring with y all the vertices colored x in the graph denoted by ce.
- Finally, the expression $\mathsf{add}_{xEy}(ce)$ adds an E-edge from every vertex colored x to every vertex colored y in the graph denoted by ce.

Clearly every graph on n vertices can be expressed as a C-clique-expression if $|C| \geq n$. The *clique-width* of a graph is at most $|C|$ if it can be described as a C-clique-expression. A class of graphs has clique-width (at most) k if every graph in the class has clique-width at most k. A class of graphs has bounded clique-width if there is an integer k that bounds the clique-width of every graph in the class.

Remark 15. Notice that we have ignored node labels in this definition. This is done for notational convenience, one could easily include them.

Example 16. Let $C = [5]$. The graph denoted by the expression ce_1

$$\mathsf{add}_{3\to 4}(\mathsf{add}_{1\to 2}(\mathsf{add}_{1\curvearrowright 4}(1 \oplus 4) \oplus \mathsf{add}_{2\curvearrowright 3}(\mathsf{add}_{2\to 5}(\mathsf{add}_{5\to 3}((2 \oplus 5) \oplus 3)))))$$

is in Figure 6 (with the resulting coloring) and the graph denoted by the ce_2

$$\mathsf{add}_{2\to 1}(\mathsf{add}_{4\to 3}(\mathsf{add}_{2\curvearrowright 3}(2 \oplus 3) \oplus (\mathsf{rnm}_{2\to 5}(\mathsf{rnm}_{3\to 5}(ce_1)))))$$

is in Figure 7 (with the resulting coloring). □

We now state some results relating clique-width and tree-width.

Fact 17 *[10] The clique-width of any graph with tree-width k is bounded by $2^{k-1} - 1$. In particular, any class of bounded tree-width graphs is also of bounded clique-width.*

The class of all cliques has clique-width bounded by 1 but does not have bounded tree-width. Interestingly, any class of bounded degree graphs with bounded clique width also has bounded tree-width as stated by the following theorem which is an easy consequence of a result of Gurski and Wanke.

Fact 18 *[11] Let $\mathcal{C}$ be a class of graphs whose degree is bounded by n and whose clique-width is bounded by k. Then, $\mathcal{C}$ has tree-width bounded by $3k(n-1) - 1$.*

Recall that, for a SMNW $\overline{w} = (\mathsf{dom}(w), \lambda, \to, \dashrightarrow, \overset{1}{\curvearrowright}, \ldots, \overset{s}{\curvearrowright})$, we have defined two graphs $G_{\overline{w}}^{\to} = (\mathsf{dom}(w), \lambda, \to, \overset{1}{\curvearrowright}, \ldots, \overset{s}{\curvearrowright})$ and $G_{\overline{w}}^{\to^+} = (\mathsf{dom}(w), \lambda, \to^+, \overset{1}{\curvearrowright}, \ldots, \overset{s}{\curvearrowright})$.

Lemma 19. *If the clique-width of $G_{\overline{w}}^{\rightarrow^+}$ is k, then the clique-width of $G_{\overline{w}}^{\rightarrow}$ is at most $2k+1$.*

Proof. Let us fix the SMNW $\overline{w}$ and let its word projection be $w = a_1 a_2 \dots a_n$. Without loss of generality we may identify the vertex corresponding to the ith position in w by the name i. Then the natural order $(<)$ on $\{1, \dots, n\}$ gives us a total order extension of $\rightarrow^+$.

Let e be a C-clique-expression for $G_{\overline{w}}^{\rightarrow^+}$ with $C = \{c_1, \dots, c_k\}$. Any subexpression e' of e defines a graph $G_{e'}$ which is (ignoring colors) a subgraph of $G_{\overline{w}}^{\rightarrow^+}$. By structural induction on the subexpressions e' of e we will construct a C'-clique-expression f' (with $|C'| = 2k+1$) such that $G_{f'}$ is (ignoring colors) the subgraph of $G_{\overline{w}}^{\rightarrow}$ corresponding to $G_{e'}$, i.e., $G_{e'}$ and $G_{f'}$ have the same set of vertices, the same $\curvearrowright$-edges and $i \rightarrow j$ in $G_{f'}$ if and only if $j = i+1$ and $i \rightarrow^+ j$ in $G_{e'}$.

We start with some preliminary remarks which explains the formal construction described below. Note that, if two nodes $i_1 < i_2$ have the same color in $G_{e'}$ then they will have the same color in all subexpressions e'' of e containing e'. Hence, it will no more be possible to add a $\curvearrowright$ edge to or from i_1 or i_2. Also, if three nodes $i_1 < i_2 < i_3$ have the same color in $G_{e'}$ it will no more be possible to add the edges $i_2 - 1 \rightarrow^+ i_2$ or $i_2 \rightarrow^+ i_2 + 1$ (otherwise loops would be introduced as well). Hence, successor edges may only be added to the first node having some color or from the last node having some color.

Inspired by the above observations, we introduce the new set of colors $C' = \{b_1, \ell_1, b_2, \ell_2, \dots, b_k, \ell_k, \texttt{blk}\}$. In our construction, the first (resp. last) node having some color c_m in $G_{e'}$ will be colored b_m (resp. ℓ_m) in $G_{f'}$ and all other nodes colored c_m in $G_{e'}$ will be colored $\texttt{blk}$ (black) in $G_{f'}$.

Formally, the C'-clique-expression f' associated with a subexpression e' of e will satisfy the following properties:

Invariant 20 *The graphs $G_{e'}$ and $G_{f'}$ have the same set of vertices, which is a subset of $\{1, \dots, n\}$. Moreover,*

1. *For all $1 \le m \le k$, if i is a vertex in $G_{e'}$ whose color is c_m, then i appears in $G_{f'}$ with color*
   - *b_m if i is the smallest vertex in $G_{e'}$ with color c_m,*
   - *ℓ_m if i is the largest (and not the smallest) vertex in $G_{e'}$ with color c_m,*
   - $\texttt{blk}$ *otherwise.*
2. *If there is an edge $i \curvearrowright j$ in $G_{e'}$ then it appears in $G_{f'}$ as well.*
3. *If there is an edge $i \rightarrow^+ i+1$ in $G_{e'}$ then there is an edge $i \rightarrow i+1$ in $G_{f'}$.*

We give now the inductive translation. For the base case, we translate an expression $e' = c_p$ as the expression $f' = b_p$ with the understanding that, w.l.o.g., $G_{e'}$ and $G_{f'}$ utilize the same vertex in $\{1, \dots, n\}$. The invariant properties are satisfied trivially.

Suppose $e' = \mathsf{add}_{c_p \curvearrowright c_q}(e'')$. Then there must be at most one vertex colored c_p and one vertex colored c_q in $G_{e''}$. So we let $f' = \mathsf{add}_{b_p \curvearrowright b_q}(f'')$ and it is easy to verify the invariant properties.

Suppose $e' = \mathsf{add}_{c_p \rightarrow^+ c_q}(e'')$. Then we first observe that if i is a c_p-colored vertex and j is a c_q-colored vertex then $i < j$ (otherwise the resulting graph cannot be $G_{\overrightarrow{w}}^{+}$). Thus, if at all, one single $\rightarrow$ edge needs to be added from the highest c_p-colored vertex to the lowest c_q-colored vertex. If the last c_p-colored vertex is not the predecessor of the first c_q-colored vertex then we set $f' = f''$. Otherwise, we set $f' = \mathsf{add}_{\ell_p \rightarrow b_q}(f'')$ if the first and last c_p-colored vertices are different or else we set $f' = \mathsf{add}_{b_p \rightarrow b_q}(f'')$. Once again by construction the invariant properties are satisfied.

Suppose $e' = \mathsf{rnm}_{c_p \rightarrow c_q}(e'')$. If $c_p = c_q$ or if there are no c_p-colored vertices then we let $f' = f''$. If $c_p \neq c_q$ and there are no c_q-colored vertices then we let $f' = \mathsf{rnm}_{b_p \rightarrow b_q}(\mathsf{rnm}_{\ell_p \rightarrow \ell_q}(f''))$. Otherwise, let i_p, i_q be the smallest vertices in $G_{e'}$ with color c_p and c_q respectively and similarly let j_p, j_q be the largest vertices in $G_{e'}$ with color c_p and c_q respectively.

– If $i_p < i_q$ and $j_q < j_p$ then $f' = \mathsf{rnm}_{b_p \rightarrow b_q}(\mathsf{rnm}_{\ell_p \rightarrow \ell_q}(\mathsf{rnm}_{b_q \rightarrow \mathtt{blk}}(\mathsf{rnm}_{\ell_q \rightarrow \mathtt{blk}}(f''))))$.
– If $i_p < i_q$ and $j_p < j_q \neq i_q$ then $f' = \mathsf{rnm}_{b_p \rightarrow b_q}(\mathsf{rnm}_{b_q \rightarrow \mathtt{blk}}(\mathsf{rnm}_{\ell_p \rightarrow \mathtt{blk}}(f'')))$.
– If $i_p < i_q$ and $j_p < j_q = i_q$ then $f' = \mathsf{rnm}_{b_p \rightarrow b_q}(\mathsf{rnm}_{b_q \rightarrow \ell_q}(\mathsf{rnm}_{\ell_p \rightarrow \mathtt{blk}}(f'')))$.
– If $i_q < i_p$ and $j_q < j_p \neq i_p$ then $f' = \mathsf{rnm}_{\ell_p \rightarrow \ell_q}(\mathsf{rnm}_{\ell_q \rightarrow \mathtt{blk}}(\mathsf{rnm}_{b_p \rightarrow \mathtt{blk}}(f'')))$.
– If $i_q < i_p$ and $j_q < j_p = i_p$ then $f' = \mathsf{rnm}_{b_p \rightarrow \ell_q}(\mathsf{rnm}_{\ell_q \rightarrow \mathtt{blk}}(f''))$.
– If $i_q < i_p$ and $j_p < j_q$ then $f' = \mathsf{rnm}_{b_p \rightarrow \mathtt{blk}}(\mathsf{rnm}_{\ell_p \rightarrow \mathtt{blk}}(f''))$.

Item 1 of Invariant 20 is true by construction and Items 2 and 3 are also true since no new edges are added.

Finally, suppose $e' = e_1 \oplus e_2$. By induction, we have expressions f_1 and f_2 satisfying Invariant 20 w.r.t. e_1 and e_2 respectively. For each color c_m such that both G_{e_1} and G_{e_2} have c_m-colored vertices we do the following renamings. Let i_1, j_1 (resp. i_2, j_2) be the first and last c_m-colored vertices in G_{e_1} (resp. in G_{e_2}).

– If $i_1 < i_2 \leq j_2 < j_1$ then we rename b_m and ℓ_m to $\mathtt{blk}$ in f_2.
– If $i_1 < i_2 < j_1 < j_2$ then we rename ℓ_m to $\mathtt{blk}$ in f_1 and b_m to $\mathtt{blk}$ in f_2.
– If $i_1 \leq j_1 < i_2 < j_2$ then we rename ℓ_m to $\mathtt{blk}$ in f_1 and b_m to $\mathtt{blk}$ in f_2.
– If $i_1 \leq j_1 < i_2 = j_2$ then we rename ℓ_m to $\mathtt{blk}$ in f_1 and b_m to ℓ_m in f_2.

We proceed similarly if $i_2 < i_1$. Let f_1' and f_2' be the expressions obtained after all these renamings. Then we set $f' = f_1' \oplus f_2'$. Thanks to the renamings, Item 1 of Invariant 20 is satisfied. Items 2 and 3 are also true since no new edges are added.

Let f be the expression associated with e inductively with the above translation. Since G_e ignoring colors is $G_{\overrightarrow{w}}^{+}$, we deduce from Invariant 20 that G_f ignoring colors is $G_{\overrightarrow{w}}$. This completes the proof of this lemma. □

5.3 Bounded Split-width to Bounded Clique-width

Here we show that bounded split-width implies bounded clique-width. As a consequence of Fact 18, this also shows that bounded split-width implies bounded tree-width. However, the bound we obtain in Theorem 11 by direct translation is better. We need the following notion and result first:

Let e be a C-clique-expression and π be a bijection from C to C. By $\pi(e)$ we denote the C-clique-expression in which every occurrence of each color c in e is replaced by $\pi(c)$. Let $\mathrm{Nodes}(G)$ denote the nodes of a graph G.

Lemma 21. *If G is the graph generated by the C-clique-expression e and $\chi : Nodes(G) \mapsto C$ gives the resulting coloring, then $\pi(e)$ also generates the graph G and the resulting coloring is given by $\chi' = \pi \circ \chi$.*

Proof. The lemma can be proved by induction. The base case and the inductive case for disjoint union are trivial. The inductive cases for $\mathsf{rnm}_{x \to y}$ and add_{xEy} are also easy to check. ☐

Theorem 22. *Let $\overline{w}$ be a SMNW of split-width at most $k \geq 2$. Then,*

1. *the clique-width of $G_{\overline{w}}^{\to +}$ is at most k.*
2. *the clique-width of $G_{\overline{w}}^{\to}$ is at most $2k+1$.*

Proof. By Lemma 19 it will be sufficient to prove the first statement. Let C be a set of colors with $|C| = k$. We will give an inductive proof satisfying a stronger invariant:

Invariant 23 *For every SMNW $\overline{w}$ with split-width at most k, there is a C-clique-expression e_w generating $G_{\overline{w}}^{\to +}$, such that:*

1. *If two nodes belong to the same component then they get the same color.*
2. *For every color, there is at most one component colored by it.*

The base cases, singleton nodes and pairs of nodes connected by $\overset{i}{\curvearrowright}$ edges, can be easily given a C-clique-expression satisfying the invariant.

Let $\overline{w} \in \overline{u} \sqcup \overline{v}$ be of split-width at most k. Since $\overline{u}$ is of split-width at most k, by inductive hypothesis, there is a C-clique-expression e_u generating $G_{\overline{u}}^{\to +}$ and satisfying the invariant. Similarly there is a C-clique-expression e_v generating $G_{\overline{v}}^{\to +}$ and satisfying the invariant. As the total number of components in u and v is less than k, thanks to Invariant 23 and Lemma 21, we can safely assume that if a color is still used for a component in $G_{\overline{u}}^{\to +}$, then it is not used for a component in $G_{\overline{v}}^{\to +}$ and vice versa. Hence $e_w = e_u \oplus e_v$ satisfies the invariant.

Let $\overline{w} \in \mathsf{merge}(\overline{u})$. Since $\overline{u}$ has split-width at most k, there is a C-clique expression e_u generating $G_{\overline{u}}^{\to +}$ and satisfying the invariant. Without loss of generality, we may assume that only two components, say u_i and u_{i+1} are merged. Let c_i, c_{i+1} be their respective colors. We let $e_w = \mathsf{rnm}_{c_i \to c_{i+1}}(\mathsf{add}_{c_i \to^+ c_{i+1}}(e_u))$. Clearly it satisfies the invariant. ☐

5.4 Bounded Clique-Width to Bounded Split-Width.

In fact, converse of the previous theorem is also true. This says that the notion of split-width is powerful enough to capture bounded clique-width.

Theorem 24. *For every MNW w,*

1. *If $G_w^{\lessdot}$ has clique-width k, then w has split-width at most $2k+1$.*
2. *If $G_w^{<}$ has clique-width k, then w has split-width at most $4k+3$.*

Proof. The MNW w may be identified with the SMNW $\overline{w}$ having a single component so that $G_w^{\lessdot} = G_{\overline{w}}^{\rightarrow}$ and $G_w^{<} = G_{\overline{w}}^{\rightarrow^+}$. Hence, by Lemma 19 it suffices to prove the first statement.

Suppose the graph $G_w^{\lessdot} = G_{\overline{w}}^{\rightarrow}$ has clique-width at most k. Let $C = \{c_1, \ldots, c_k\}$ and let e be a C-clique-expression generating $G_{\overline{w}}^{\rightarrow}$. Consider the intermediate graphs formed by the subexpressions of e as well as their resulting coloring. First we make a few observations.

Observation 25 *If there are two vertices with the same color, then they cannot take part in any edge addition later.*

Let us call those vertices *inert*. All the inert vertices can be colored the same. Let us use a new color $\texttt{blk} \notin C$ for the inert vertices[3]. That is, *given a clique expression, we can get an equivalent clique expression in which the inert vertices are all colored* $\texttt{blk}$. For instance, if renaming c to d results in a graph having several vertices colored d, then we rename c and d to $\texttt{blk}$ instead. We deduce that for any color $c \neq \texttt{blk}$, there is at most one vertex colored c.

Also, Lemma 21 tells us that changing the color of a vertex with $\mathsf{rnm}_{c \to d}$ into a color $d \neq \texttt{blk}$ can be avoided by permuting the colors in the expression. Thus we can conclude that, given a C-clique-expression e for a SMNW, it is possible to write an equivalent C-clique-expression e' such that

N_1 If $\mathsf{rnm}_{c \to d}$ appears in e' then $d = \texttt{blk}$ and $c \neq \texttt{blk}$.

N_2 Black vertices do not take part in edge additions and for every color $c \neq \texttt{blk}$, there is at most one vertex colored by c.

For the sake of this proof, we can safely assume that there is a C-clique-expression e satisfying the properties above and witnessing that the MNW w has clique width at most k.

Next we show that for every clique-expression e satisfying (N_1,N_2), we can get another expression e' generating the same graph such that if e' contains a sub-expressions of the form $f = \mathsf{add}_{c \overset{i}{\curvearrowright} d}(f')$ then $f' = c \oplus d$. We call such expressions $\curvearrowright$*-leafed*. This is achieved at the expense of additional colors. However, in this transformation, property (N_1) need not be preserved. Instead, we will maintain the following weaker version.

N_1' A node taking part in a $\curvearrowright$ edge addition is never recolored between its creation and this edge addition: if $\mathsf{add}_{c \overset{i}{\curvearrowright} d}(f')$ is a sub-expression, then the nodes taking part in this edge addition are not recolored in f'.

A clique-expression is said to be in *normal form* if it satisfies properties (N_1',N_2). Since black vertices do not take part in edge additions, our starting expression e is indeed in normal form.

[3] If two vertices get the same color, then we could use this color instead of $\texttt{blk}$ there by saving one color. However we keep $\texttt{blk} \notin C$ for the ease of understanding the proof

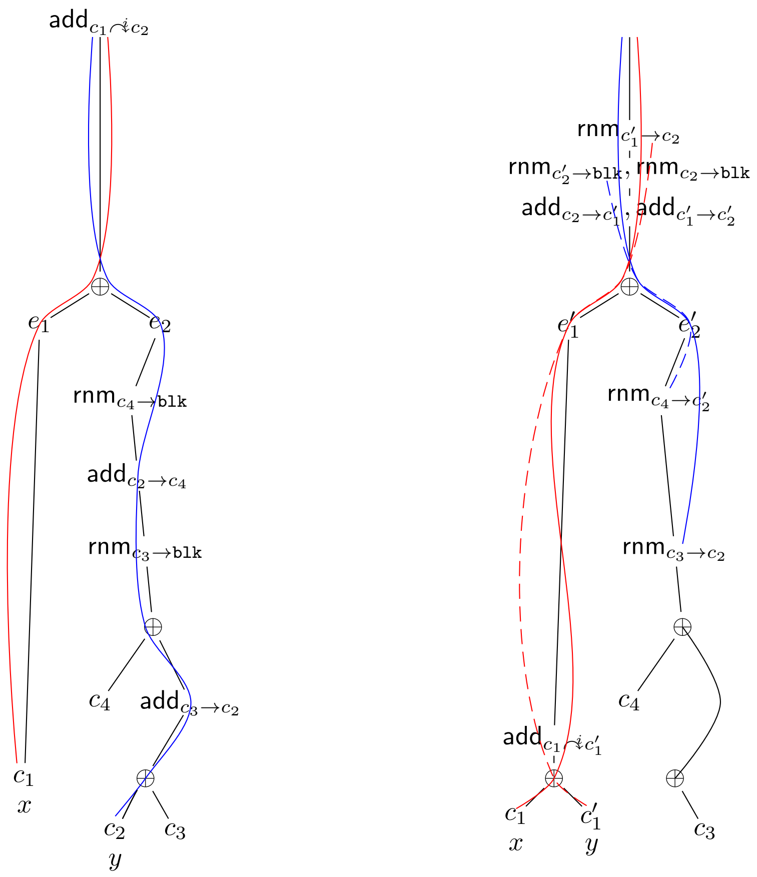
$\mathsf{add}_{c_1 \curvearrowright c_2}$
e_1
e_2
$\mathsf{rnm}_{c_4 \to \mathtt{blk}}$
$\mathsf{add}_{c_2 \to c_4}$
$\mathsf{rnm}_{c_3 \to \mathtt{blk}}$
c_4
$\mathsf{add}_{c_3 \to c_2}$
c_1
x
c_2
c_3
y
$\mathsf{rnm}_{c'_1 \to c_2}$
$\mathsf{rnm}_{c'_2 \to \mathtt{blk}}, \mathsf{rnm}_{c_2 \to \mathtt{blk}}$
$\mathsf{add}_{c_2 \to c'_1}, \mathsf{add}_{c'_1 \to c'_2}$
e'_1
e'_2
$\mathsf{rnm}_{c_4 \to c'_2}$
$\mathsf{rnm}_{c_3 \to c_2}$
c_4
$\mathsf{add}_{c_1 \curvearrowright c'_1}$
c_1
c'_1
c_3
x
y

Fig. 8. A normalized clique-expression is shown on the left. Color c_1 is active along the red part of the tree. Color c_2 is active along the blue part. An equivalent $\curvearrowright$-leafed expression is shown on the right. It uses two additional colors c'_1 and c'_2. The span of these additional colors are shown in dashed lines. Note that the final graphs of the two expressions are isomorphic.

We will now see how to obtain a $\curvearrowright$-leafed expression from an expression in normal form. During the process we may use up to k additional colors from $\{c'_1, \dots, c'_k\}$. We will bring to leaf level the $\mathsf{add}_{c_1 \overset{i}{\curvearrowright} c_2}$ expressions starting from innermost to outermost. First we will see how to bring an innermost $\mathsf{add}_{c_1 \overset{i}{\curvearrowright} c_2}$ to the leaf level. This case is depicted in Figure 8.

Suppose, the expression e contains a subexpression[4] in normal form $f = \mathsf{add}_{c_1 \overset{i}{\curvearrowright} c_2}(f')$ where f' is not $c_1 \oplus c_2$. Thanks to Observation 25, there is at most one vertex of each color c_1 and c_2. If either c_1 or c_2 is not coloring a vertex, we may omit adding this edge. So the only interesting case is when there is exactly one vertex with color c_1 and exactly one vertex with color c_2. Let x and y denote the identity of these vertices, say given by integers denoting their position in the MNW w. Thanks to Property (N'_1), x and y were created with colors c_1 and c_2 respectively and they were not recolored.

In the subexpression f', there needs to be a subexpression of the form $e_1 \oplus e_2$ such that x is created in e_1 and y is created in e_2. Our idea is to detach y from e_2 and attach it next to x as shown in Figure 8. Thus e_1 becomes e'_1. All the operations in e_1, including those done on x, are maintained as such in e'_1.

However there are several subtleties on obtaining e'_2. In the subexpression e_2, node y might have got a left neighbor y^- and a right neighbor y^+. The $\to$ edge addition needs to be postponed until y appears along with its neighbors in the same expression, i.e., above $e'_1 \oplus e'_2$. Hence the spurious edge additions in e_2 are skipped. Moreover, if e_2 involves coloring y^- or y^+ to $\mathtt{blk}$, we need to keep track of these vertices until the postponed edge additions are done. We do so by using the colors c_2 and c'_2. Right after $e'_1 \oplus e'_2$, the delayed edge additions will be performed, followed by recolorings in order to match the original coloring.

Figure 8 corresponds to the case where both edge additions $y^- \to y$ and $y \to y^+$ were performed in e_2 with y^- colored c_3 and y^+ colored c_4, and c_3, c_4 were both recolored $\mathtt{blk}$ in e_2. This is the most complete case. Now if e_2 did not involve recoloring c_3 or c_4 to $\mathtt{blk}$, we can easily adjust the few operations after $e'_1 \oplus e'_2$. If e_2 did not involve the addition of $\to$ edges from y^- or to y^+, the operations may also be adjusted easily.

To summarize, we remove $\to$ edge additions in e_2 between y and its neighbors and postpone these after $e'_1 \oplus e'_2$. We recolor a neighbor of y only when it is recolored to $\mathtt{blk}$ originally inside e_2. The operations after $e'_1 \oplus e'_2$ can always be chosen such that the resulting graphs are isomorphic.

Notice that we have used c_2 to color a neighbor of y in e'_2 only at those graphs where originally y was present and colored c_2. Thus in e'_2, there is still at most one vertex which could be colored by c_2. Note also that we have used c'_i for a graph only along with c_i, for $i = 1, 2$. Since the expression originally satisfied (N_2), there can be at most one vertex colored by c'_i, for $i = 1, 2$. Thus, the resulting $\curvearrowright$-leafed expression still satisfies Property (N_2).

Let us show now that Property (N'_1) is also preserved. The procedure may introduce at most three recolorings to non $\mathtt{blk}$ colors (See Figure 8).

[4] to be precise, we are referring to one occurrence of a subexpression

– The recoloring $\mathsf{rnm}_{c_1' \to c_2}$ concerns node y which already gets its $\curvearrowright$ edge addition in e_1' so y cannot get another $\curvearrowright$ edge addition later.
– Now, the recolorings $\mathsf{rnm}_{c_3 \to c_2}$ and $\mathsf{rnm}_{c_4 \to c_2'}$ in e_2' replace $\mathsf{rnm}_{c_3 \to \mathtt{blk}}$ and $\mathsf{rnm}_{c_4 \to \mathtt{blk}}$ in e_2. Hence by (N_2) the concerned vertices (y^- and y^+) could not take part in any edge addition after these recolorings.

We conclude that the resulting expression is still in normal form. Therefore, we may iterate the transformation for each $\curvearrowright$ edge addition which is not leafed. Finally, we obtain a $\curvearrowright$-leafed expression e' which is equivalent to e, i.e., generates the same colored version of the SMNW $\overline{w}$. Note that e' uses at most $2k+1$ colors.

The final step is to show that if an $[\ell]$-clique-expression e which is $\curvearrowright$-leafed generates the graph $G_w^{\lessdot}$ of a MNW w then w belongs to ℓ-BS. We prove by induction that every subexpression of e describes an SMNW that belongs to ℓ-BS. The base cases are atomic expressions c which are not part of a $\curvearrowright$ edge addition and expressions $\mathsf{add}_{c \curvearrowright d}(c \oplus d)$. Clearly these cases correspond to SMNWs that belong to p-BS for each $p > 1$. Addition of a $\to$ edge with $\mathsf{add}_{c \to d}(e')$ results in an SMNW fewer components which corresponds to a *merge* operation on the SMNW. Renaming of colors has no effect on the SMNW. Finally, the disjoint union of two SMNWs corresponds to *shuffle*. It remains to verify that the number of components is bounded by ℓ in all these cases.

Let f be a subexpression of e generating a SMNW $\overline{v}$. The graph G generated by f is a colored version of $G_{\overline{v}}^{\rightrightarrows}$ which is a subgraph of $G_w^{\lessdot}$. Let x be an endpoint of a $\to$ connected component of G. If x is not the first or last vertex of $G_w^{\lessdot}$ then a $\to$ edge should be added to or from x later in the expression e in order to get the graph $G_{\overline{w}}^{\rightrightarrows} = G_w^{\lessdot}$. By Observation 25, x must be assigned a unique color in G. But there are only ℓ available colors and hence there can be at most ℓ components in G: each subexpression defines an SMNW belonging to ℓ-BS. □

6 MSO is decidable over bounded split-width MNWs

In this section we show that the class of bounded split-width MNWs has a decidable MSO theory. Before starting the technical details we first recall what it means to have a decidable MSO theory.

Let $\mathcal{C}$ be a class of graphs over the vocabulary V. Consider the following problem:

Problem 26. Input: φ: An MSO formula over the vocabulary V.
Question: Is there a graph $G \in \mathcal{C}$ such that $G \models \varphi$?

We say that the class $\mathcal{C}$ has a *decidable MSO-theory* if the above problem is decidable.

Now consider a class $\mathcal{D} \subseteq \mathcal{C}$. It is possible that $\mathcal{D}$ has a decidable MSO theory, but $\mathcal{C}$ does not. It is also possible that $\mathcal{C}$ has decidable MSO theory but $\mathcal{D}$ does not. However, if the class $\mathcal{D}$ is MSO definable, that is to say that there is an MSO formula $\phi_{\mathcal{D}}$ over vocabulary V defining the class $\mathcal{D}$ within the class $\mathcal{C}$, then if the class $\mathcal{C}$ has a decidable MSO theory, then the class $\mathcal{D}$ also has a decidable MSO theory. Indeed, we can reduce Problem 26 for class $\mathcal{D}$ to that for class $\mathcal{C}$.

An MSO definable class with bounded tree-width (or clique-width) has a decidable MSO theory. However, we do not know whether the class of k-BS MNWs is MSO-definable. Thus, we cannot infer from Theorem 11 that k-BS MNWs has a decidable MSO theory. Nevertheless, we have the following theorem:

Theorem 27. *Let $k \in \mathbb{N}$. The class of MNWs with split-width at most k has a decidable MSO theory.*

Before proving the theorem, we will first introduce some notions and prove some lemmas which will be helpful for the proof.

Let $\overline{w}$ be an SMNW in k-BS. By definition, the proof of membership of $\overline{w}$ in k-BS can be seen as a tree whose nodes are labelled by elements of k-BS and whose degree is bounded by 2 such that

T1 the root is labelled by $\overline{w}$.
T2 leaves are labelled by atomic MNWs.
T3 if an internal node labelled $\overline{u}$ has only one child labelled $\overline{v}$ then $\overline{u} \in \mathsf{merge}(\overline{v})$.
T4 if an internal node labelled $\overline{u}$ has two children labelled $\overline{x}$ and $\overline{y}$ then $\overline{u} \in \overline{x} \sqcup\!\sqcup \overline{y}$.

We abstract such a proof as a finitely labelled tree, called a *proof tree*. Internal nodes are labelled by one of

L1 $\overset{i}{\curvearrowright}$ to denote a nesting edge for stack i. Such a node has two children which must be leaves.
L2 $\mathsf{mrg}(m, i_1, i_2, \ldots, i_{n-1}, i_n)$ with $1 \leq i_1 < i_2 < \ldots i_{n-1} < i_n = m \leq k$ and $n < m$. This denotes a merge operation on an SMNW with m components resulting in an SMNW with n components: $1, \ldots, i_1$ coalesce to give the first component, $i_1 + 1, \ldots, i_2$ gives the second component and so on.
L3 $\mathsf{shf}(m, n, f_\ell, f_r)$ with $1 \leq m < m + n \leq k$ where $f_\ell : [m] \longrightarrow [m+n]$ is a monotone map that describes the positions in the shuffled word of the m components coming from the left child and similarly for f_r and the n components coming from the right child.

Notice that a leaf labelled by $a \overset{i}{\curvearrowright} b$ in the proof is represented by a subtree with three nodes in the abstraction.

The labels described by (L1–L3) constitute an alphabet L of size $|L| \leq s + 2^k + 2^{k+1}$. A tree labelled with L is a *valid proof tree* if it is the abstraction of an actual proof of membership of some SMNW in k-BS. In fact we have an MSO formula describing valid proof trees.

Lemma 28. *The set of valid proof trees of k-BS is definable by an existential MSO formula* Proof_k.

Proof. The formula asserts that the binary labelled trees satisfy the properties below. We define for each stack $i \in [s]$ and each node v, a relation denoted $\mathsf{cr}_i(v) \subseteq k^2$ describing the components of v which are linked with nesting edges of stack i.

P1 A leaf v is labelled with some letter from Σ and $\mathsf{cr}_i(v) = \emptyset$ for each $i \in [s]$.
P2 Any node v labelled $\overset{i}{\curvearrowright}$ has two children which are leaves. Moreover, $\mathsf{cr}_i(v) = \{(1,2)\}$ and $\mathsf{cr}_j(v) = \emptyset$ for $j \neq i$.
P3 Any node v labelled $\mathsf{mrg}(m, i_1, \ldots, i_n)$ has one child v'. Moreover, for each stack i the relation $\mathsf{cr}_i(v)$ consists of the pairs (p,q) such that $1 \leq p < q \leq n$ and $(p', q') \in \mathsf{cr}_i(v')$ for some $i_{p-1} < p' \leq i_p$ and $i_{q-1} < q' \leq i_q$ (with $i_0 = 0$).
P4 Any node v labelled $\mathsf{shf}(m, n, f_\ell, f_r)$ has a left child v' and a right child v''. For each stack i we let $\mathsf{cr}_i(v) = f_\ell(\mathsf{cr}_i(v')) \cup f_r(\mathsf{cr}_i(v''))$. We make sure that the well-nesting requirement is not violated by checking that there are no $(p,q) \in \mathsf{cr}_i(v')$ and $(p', q') \in \mathsf{cr}_i(v'')$ with either $f_\ell(p) < f_r(p') < f_\ell(q) < f_r(q')$ or $f_r(p') < f_\ell(p) < f_r(q') < f_\ell(q)$.

We can easily write an existential MSO formula Proof_k which guesses for each stack $i \in [s]$ and relation $R \subseteq [k]^2$ the set $X_{i,R}$ of nodes v such that $\mathsf{cr}_i(v) = R$ and then checks that the 4 properties above are satisfied with a (local) FO formula. ☐

Alternately, we can construct a tree automaton that guesses the relations $\mathsf{cr}_i(v)$ in its states and checks the 4 properties which are local to a node and its children. Thus the tree automaton will have $2^{s \times k^2}$ states. This is summarized in the following lemma.

Lemma 29. *We can construct a tree automaton $\mathcal{A}_k$ accepting precisely the set of valid proof trees of k-BS. The number of states of the automaton is $\mathcal{O}(2^{s \times k^2})$.*

We are now ready to prove Theorem 27.

Proof (of Theorem 27). Our idea is to translate any MSO formula Φ over multiply nested words to an "equivalent" formula Φ' over proof-trees. The intuition is that the positions in the MNW are leaves of the proof-tree. Hence, we will translate the quantifications on positions of the MNW to quantifications on leaves of the proof-tree. This is the classical relativization technique. We translate the atomic formula $x \overset{i}{\curvearrowright} y$ into a formula $N^i(x,y)$ which asserts that x and y have a common parent labelled $\overset{i}{\curvearrowright}$. The more difficult part is to recover the linear ordering $<$ from the proof-tree with some formula $\mathsf{lt}(x,y)$ which is explained below. The formula Φ' is a conjunction of Proof_k and a formula Φ'' which is obtained by the following inductive translation:

F1 $\exists x.\alpha$ translates to $\exists x.\mathsf{leaf}(x) \wedge \alpha''$.
F2 $\exists X.\alpha$ translates to $\exists X.\forall x.(x \in X \implies \mathsf{leaf}(x)) \wedge \alpha''$.
F3 $a(x)$, $x = y$ and $x \in X$ translate to themselves.
F4 $\alpha_1 \vee \alpha_2$ translates to $\alpha_1'' \vee \alpha_2''$ and $(\neg\alpha)$ translates to $\neg(\alpha'')$.
F5 $x \overset{i}{\curvearrowright} y$ translates to $N^i(x,y)$ and $x < y$ translates to $\mathsf{lt}(x,y)$.

To express $\mathsf{lt}(x,y)$ in the proof-tree, we have to trace the components in which x and y occur up to their least common ancestor. The easy case is when

$N^i(x, y)$ which indeed implies $x < y$. Otherwise the closest common ancestor must be a shuffle node z. The existential formula $\mathsf{lt}(x, y)$ is of the form $\exists X_1, \ldots, X_k, Y_1, \ldots, Y_k\, \psi$. The formula ψ ensures that X_i is the set of ancestors of x in which x "belongs" to the ith component. For instance, for a shuffle node v labelled $\mathsf{shf}(m, n, f_\ell, f_r)$ with left child v', if $v' \in X_i$ then we request $v \in X_{f_\ell(i)}$. For a merge node v labelled $\mathsf{mrg}(m, i_1, \ldots, i_n)$ with child v', if $v' \in X_p$ with $i_{q-1} < p \leq i_q$ then we request $v \in X_q$. Similarly, ψ ensures that Y_i is the set of ancestors of y in which y "belongs" to the ith component. Finally, ψ states that for some common ancestor z of x and y, and we have $z \in X_i \cap Y_j$ for some $i < j$.

Notice that the formula Proof_k makes sure that the well-nesting requirement is met. We see that the formula Φ' is satisfiable over trees if and only if Φ is satisfiable over the class of multiply nested words in k-BS. □

We will now consider some parametrized decision problems of multi-pushdown systems with the bound on split-width as a parameter. We first consider the bounded split-width emptiness checking of MPDSs: that is, given an integer k and a multi-pushdown system $\mathcal{M}$, is there an MNW w accepted by $\mathcal{M}$ with split-width at most k?

Theorem 27 and Remark 1 allows us to conclude the decidability of the above problem. However, we can improve the complexity by directly building a tree-automaton $\mathcal{A}_{\mathcal{M}}$ over proof trees such that $\mathcal{A}_{\mathcal{M}}$ accepts a proof tree if and only if the corresponding MNW has split-width at most k and is accepted by $\mathcal{M}$. We explain this below.

Given an MPDS $\mathcal{M}$, we can obtain the tree automaton over proof trees $\mathcal{A}_{\mathcal{M}}$ as follows. Let Q be the set of states of $\mathcal{M}$. The set of states of $\mathcal{A}_{\mathcal{M}}$ are Q^{2k}. The idea is to label the beginning and end of each component of a split MNW by states from Q. Each component j is marked to start in q_j and end in state q'_j in the tree automaton $\mathcal{A}_{\mathcal{M}}$ if and only if that component can potentially take the MPDS $\mathcal{M}$ from q_j to q'_j. This intuition is followed while marking the states of the leaves. Since the $\overset{i}{\curvearrowright}$ edges also appear next to leaves, the states can be marked consistently with the push-pop transitions. For example, suppose a $\overset{i}{\curvearrowright}$-node v has a left child v_1 labelled a and a right child v_2 labelled b. Further suppose that a run labels v_1 with the state (q_1, q'_1) and v_2 with the state (q_2, q'_2). It is possible to label the node v with (q_1, q'_1, q_2, q'_2) if and only if there exist state q and transitions $(q_1, a, q'_1, q, i) \in \Delta_{push}$ and $(q_2, q, b, q'_2, i) \in \Delta_{pop}$ of the MPDS $\mathcal{M}$. For a shuffle node, the start and end states of each component are inherited from the corresponding child. For a merge node v labelled $\mathsf{mrg}(m, i_1, \ldots, i_n)$ with child v', we check that $q'_p = q_{p+1}$ for all $i_{j-1} < p < i_j$, i.e., when components p and $p+1$ from v' are merged, the end state of component p equals the start state of component $p+1$. Then, the start and end states of components of node v are inherited from v' as expected. Thus we get the following theorem:

Theorem 30. *Given a MPDS $\mathcal{M}$ and an integer k, we can construct in exponential time a tree automaton $\mathcal{A}_{k,\mathcal{M}} = \mathcal{A}_k \times \mathcal{A}_{\mathcal{M}}$ over proof trees, such that $\mathcal{A}_{k,\mathcal{M}}$ accepts all the valid proof trees of MNWs in k-BS which have an accepting run in $\mathcal{M}$. The size of $\mathcal{A}_{k,\mathcal{M}}$ is exponential in k and the number of stacks s, but is polynomial (with exponent $\mathcal{O}(k)$) in the number of states of $\mathcal{M}$.*

Theorem 30 gives us an ExpTime procedure for emptiness checking of a multi-pushdown system restricted to bounded split-width behaviors.

Corollary 31. *Emptiness checking of a multi-pushdown system restricted to bounded split-width behaviors is* ExpTime.

Next we consider bounded split-width model checking of MPDS against MSO. That is, given a multi-pushdown system $\mathcal{M}$, an integer k and an MSO formula φ over MNWs, do all MNWs of split-width at most k generated by $\mathcal{M}$ satisfy φ? This problem is decidable. Once the split-width k is given, we can translate the MSO formula φ over MNWs into an equivalent formula φ'' over proof trees as explained in the proof of Theorem 27. We then need to verify that all trees accepted by the tree automaton $\mathcal{A}_{k,\mathcal{M}}$ constructed as per Theorem 30 satisfy φ''. Since MSO model checking of tree automaton is decidable we get decidability for MSO model checking of MPDS wrt. bounded split-width behaviors.

Corollary 32. *MSO-model checking of MPDS restricted to k-BS is decidable in time non-elementary in the length of the MSO formula, exponential in k and the number of stacks s, and polynomial in the number of states of the MPDS.*

Next we consider bounded split-width inclusion checking of MPDS. That is given two MPDS $\mathcal{M}_1$ and $\mathcal{M}_2$ and an integer k, are all the MNWs accepted by $\mathcal{M}_1$ and having split-width at most k also accepted by $\mathcal{M}_2$? In other words, is $L(\mathcal{M}_1) \cap k\text{−BS} \subseteq L(\mathcal{M}_2)$? This can be done by checking whether $L(\mathcal{A}_{k,\mathcal{M}_1}) \subseteq L(\mathcal{A}_{k,\mathcal{M}_2})$. The inclusion test for tree automaton can be done by standard constructions which involves complementing the latter tree automaton. Thus we get the following corollary.

Corollary 33. *Inclusion checking of two MPDS wrt. k-BS is* 2ExpTime.

Since we have an exponential sized tree automata tree automata $\mathcal{A}_k$ recognizing the set of valid proof trees of k-BS (Lemma 29), inclusion checking implies universality checking wrt. k-BS also.

Corollary 34. *Universality checking of MPDS wrt. k-BS is* 2ExpTime.

7 More results

Once we show the bound on split-width, we can derive MSO decidability and bounds on tree-width and clique-width.

Theorem 27 along with Proposition 3 and Theorem 4 gives us the MSO decidability of the classes defined in Section 4:

Corollary 35. *The classes Bounded Scope, Bounded Phase, Ordered,* SBO, SPB *have a decidable MSO theory.*

Theorem 11 along with Theorem 4 gives us new bounds of tree-width of the different classes of MNWs. We improve the $s2^{s-1}$ bound on tree-width of ordered MNWs obtained in [20] to 2^{s+1}. We also improve the $2ms$ bound on tree-width for bounded scope MNWs obtained in [17] to $2(m+2)$.

Corollary 36. *1. m-Bounded scope MNWs have tree-width at most $2(m+2)$.*
2. p-Bounded phase MNWs have tree-width at most 2^{p+1}.
3. Ordered MNWs have tree-width at most 2^{s+1}.
4. m-SBO have tree-width at most $2^{s+1}(2m+1)$.
5. (m,p)-SPB have tree-width at most $2^{p+1}(2m+1)$.

A theorem by Courcelle [8] says that if MSO is decidable for a class $\mathcal{C}$ of graphs with bounded degree, then $\mathcal{C}$ has bounded clique-width. This theorem, along with Item 3 of Theorem 11, says that any class of MNWs with decidable MSO theory indeed has bounded split-width.

Corollary 37. *Let $\mathcal{C}$ be a class of MNWs. If $\mathcal{C}$ has a decidable MSO theory, then $\mathcal{C}$ has bounded split-width.*

8 Discussion and Perspectives

We have introduced and studied a new metric on MNWs called split-width and its relationship with clique-width and tree-width. Using split-width as a tool, decidability of MSO for several existing as well as new classes of MPDS have been shown. We can even extend the decidable classes further.

A next step is to bridge the gap in the translations between split-width and tree-width (or clique-width). Is it possible to obtain a linear translation from tree-width to split-width? Is it possible to close the gap in the back and forth translations between split-width and tree-width (or clique-width)? In other words, is split-width another characterization of tree-width (or clique-width) of MNWs?

Another interesting question is whether MPDS with k bounded split-width restriction are closed under complementation. That is, given a MPDS $\mathcal{M}$ and k, is there another MPDS $\mathcal{M}'$ such that for all k-bounded split-width MNWS w, w is accepted by $\mathcal{M}$ if and only if w is not accepted by $\mathcal{M}'$?

It is interesting to know whether one could employ temporal logics instead of MSO for model checking MPDS wrt. k-split-width-bounded runs, and get a reasonable complexity.

Another important direction is to find notions similar to split-width for other domains like message sequence charts, data words etc.

References

1. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.
2. M. F. Atig. From multi to single stack automata. In P. Gastin and F. Laroussinie, editors, *CONCUR*, volume 6269, pages 117–131. Springer, 2010.
3. M. F. Atig. Global model checking of ordered multi-pushdown systems. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 8, pages 216–227. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

4. M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-Complete. In M. Ito and M. Toyama, editors, *Developments in Language Theory*, volume 5257, pages 121–133. Springer, 2008.
5. L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-pushdown languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
6. B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. *Graph Structure Theory*, 147:565–590, 1993.
7. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars*, pages 313–400. World Scientific, 1997.
8. B. Courcelle. The monadic second-order logic of graphs XV: On a conjecture by D. Seese. *Journal of Applied Logic*, 8:1–40, 2006.
9. B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993.
10. B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.
11. F. Gurski and E. Wanke. The tree-width of clique-width bounded graphs without kn, n. In U. Brandes and D. Wagner, editors, *WG*, volume 1928, pages 196–205. Springer, 2000.
12. A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In C.-H. L. Ong, editor, *FOSSACS*, volume 6014, pages 267–281. Springer, 2010.
13. S. Kreutzer. Algorithmic meta-theorems. *CoRR*, abs/0902.3616, 2009.
14. S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.
15. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In Ramakrishnan and Rehof [22], pages 299–314.
16. S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In J.-P. Katoen and B. König, editors, *CONCUR*, volume 6901, pages 203–218. Springer, 2011.
17. S. La Torre and G. Parlato. Scope-bounded multistack pushdown systems: fixed-point, sequentialization, and tree-width. Technical report, University of Southampton, February 2012.
18. A. Lal and T. W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design*, 35(1):73–97, 2009.
19. A. Lal, T. Touili, N. Kidd, and T. W. Reps. Interprocedural analysis of concurrent programs under a context bound. In Ramakrishnan and Rehof [22], pages 282–298.
20. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In T. Ball and M. Sagiv, editors, *POPL*, pages 283–294. ACM, 2011.
21. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In N. Halbwachs and L. D. Zuck, editors, *TACAS*, volume 3440, pages 93–107. Springer, 2005.
22. C. R. Ramakrishnan and J. Rehof, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963. Springer, 2008.
23. D. Seese. The structure of models of decidable monadic theories of graphs. *Ann. Pure Appl. Logic*, 53(2):169–195, 1991.

24. A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In T. Touili, B. Cook, and P. Jackson, editors, *CAV*, volume 6174, pages 615–628. Springer, 2010.