Nested Words for Order-2 Pushdown Systems

C. Aiswarya*[1], Paul Gastin[2], and Prakash Saivasan[1]

[1] Chennai Mathematical Institute, India
{aiswarya,saivasan}@cmi.ac.in
[2] LSV, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France
paul.gastin@lsv.ens-cachan.fr

Abstract. We study linear time model checking of higher-order pushdown systems (HOPDS) of level 2 (manipulating stack of stacks) against MSO and PDL (propositional dynamic logic with converse and loop) enhanced with push/pop matching relations. To capture these linear time behaviours with matchings, we propose level-2 nested words. These graphs consist of a word structure augmented with two binary matching relations, one for each level of stack, which relate a push with matching pops at the respective level. Due to the matching relations, satisfiability and model checking are undecidable. Hence we propose an under-approximation, bounding the number of times a level-1 push can be popped. With this under-approximation we get decidability for satisfiability and model checking of both MSO and PDL. The complexity of our decision procedure is ExpTime-Complete for PDL.

1 Introduction

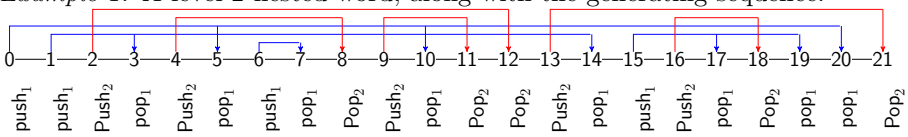
The study of higher-order pushdown systems (HOPDS) has been a prominent line of research [1,10,11,13,19,21,22]. HOPDS are equipped with a stack (level-1 stack, classical pushdown), or a stack of stacks (level-2 stack), or a stack of stacks of stacks (level-3 stack) and so on. They naturally extend the classical pushdown systems to higher orders, and at the same time they can be used to model higher-order functions which is a feature supported by many widely-used programming languages like scala, python, etc. [25].

HOPDS of level-2 (2-HOPDS) is the topic of study of this article. They are equipped with a level-2 stack, i.e., a stack of stacks. The stacks that are entered in the level-2 stack are level-1 stacks. Level-1 push/pop operations apply on the topmost level-1 stack of the level-2 stack. Level-2 push/pop operations will duplicate/remove the topmost level-1 stack.

For classical pushdown systems, nested words [7] prove to be a convenient structure for the behaviours. Nesting edges link matching pushes and pops, and this makes the stack accesses of a run visible. The nested word structure also allows richer specification formalisms which can make use of the nesting edges. We propose a similar structure, called level-2 nested words (2-NW), for 2-HOPDS. The structure has a level-1 nesting relation and a level-2 nesting relation which link matching pushes and pops of the respective level.

* This work was done when she was affiliated to Uppsala University, Sweden.

Example 1. A level-2 nested word, along with the generating sequence:



A 2-NW can encode the branching behaviour of a pushdown system, just like nested trees [5, 6, 17]. In fact, nested trees are to 2-NW as trees are to nested words. In the encoding, the nesting edges on the tree are captured by the level-1 nesting edges, and the tree structure is captured by the level-2 nesting edges.

Level-2 nested words can also model behaviours of recursive program with local stack data-structure which may pass-by-value the local stack to its subroutines. Thus the subroutines may access and manipulate a copy of the stack, and upon return, the caller-process can continue with the original value of the stack.

We provide a characterisation of level-2 nested words, which allows us to compute the nesting edges given a sequence of operations. Based on it, we have linear time algorithm, `Nestify`, for doing the same. Our tool `Nestify`, accessible at `http://www.lsv.ens-cachan.fr/~gastin/hopda`, generates a level-2 nested word representation in several formats, including a pdf picture as in Example 1.

We propose propositional dynamic logic with loop and converse (LCPDL) and monadic second-order logic (MSO(2-NW)) over level-2 nested words to specify properties of 2-HOPDS. These specification formalisms are very expressive since they can use the nesting relations. LCPDL is a navigating logic which can walk in the level-2 nested word by moving along the nesting edges and the linear edges. It is powerful enough to subsume usual temporal logic operators. We provide a jump edge in addition which allows it to go from a level-1 pop to the next level-1 pop that is attached to the same push.

Non-emptiness checking of 2-HOPDS is known to be ExpTime-Complete [17]. However, with the presence of the matching relations, satisfiability checking and model checking of both LCPDL and MSO(2-NW) are undecidable. The reason is that we can interpret a grid in a level-2 nested word using LCPDL (and hence also MSO(2-NW)).

Inspired by the success of under-approximation techniques in verification of other turing powerful settings like multi-pushdown systems, message passing systems etc., we propose an under-approximation for 2-HOPDS. This under-approximation, called bounded-pop, bounds the number of level-1 pops that a push can have. This amounts to bounding the degree of the nodes of the 2-NW graph. With this restriction we gain decidability for satisfiability and model checking of MSO(2-NW). For LCPDL these problems are shown to be ExpTime-Complete.

Our proof of decidability is by showing that bounded-pop 2-NW can be interpreted over trees. Towards a tree-interpretation, we first lift the notion of split-width [3, 15, 16] to 2-NW, and show that bounded-pop 2-NW have a bound on split-width. Bounded split-width 2-NW have bounded (special) tree-width [14], and hence bounded-pop 2-NW can be effectively interpreted over special tree-terms.

Related work. Higher-order pushdown automata was introduced as a generalization of pushdown automata by [1,19,22]. There has been a body of research on this model from studying language-theoretic properties [17] to alternate characterisations [1,13,21] to games on configuration graphs [11,12].

In [10], Broadbent studies nested structures of level-2 HOPDS. A suffix rewrite systems which rewrites nested words are used to captures the graph of ϵ-closure of a level-2 HOPDS. The objective of the paper as well as the use of nested words is different from ours.

On nested trees, μ-calculus was shown to be decidable in [5] and first-order logic model checking was shown to be decidable in [20]. These logics have the child relation in the tree but not the descendent relation (transitive closure). Monadic second-order logic, which has the transitive closure, is undecidable [5]. Though nested trees are closely linked to level-2 nested words, LCPDL allows transitive closure and hence we have undecidability. Our restriction of bounded-pop amounts to bounding the number of matching pops that a node can have, which in turn bounds the degree of the nodes in the tree.

Split-width was first introduced in [16] for MSO-decidability of multiply-nested words. It was later generalised to message sequence charts with nesting (also concurrent behaviours with matching) [3,15]. Here, we lift it to 2-NW.

2 Level-2 Higher-order pushdown systems (2-HOPDS)

These are finite state systems with a stack of stacks as an auxiliary storage.

Level-2 Stack. Let $S = \{s_1, s_2 \ldots\}$ be a finite set of symbols/labels. A level-1 stack is a sequence of symbols (or a word) of the form $w = s_1 s_2 \ldots s_n$ where s_n is the topmost symbol of w. A labelled stack is a pair (s, w). A level 2 stack is a sequence of labelled stacks of the form $W = (s_1, w_1)(s_2, w_2) \ldots (s_m, w_m)$ where each w_i is a level-1 stack, and (s_m, w_m) is the topmost labelled stack of W.

The following operations modify the level-2 stack. Here, $s \in S$.

- $\mathsf{push}_1(s)$: pushes label s to the topmost level-1 stack. That is, if $W = (s_1, w_1) \cdots (s_m, w_m)$, then $\mathsf{push}_1(s)(W) = (s_1, w_1) \cdots (s_m, w_m s)$.
- $\mathsf{pop}_1(s)$: pops the topmost label from the topmost level-1 stack, if this label is s. That is, if $W = (s_1, w_1)(s_2, w_2) \cdots (s_m, w_m s)$, then $\mathsf{pop}_1(s)(W) = (s_1, w_1)(s_2, w_2) \cdots (s_m, w_m)$. If the topmost label in the topmost level-1 stack is not s, then $\mathsf{pop}_1(s)$ is undefined. This can be thought of as a test-and-pop operation.
- $\mathsf{Push}_2(s)$: duplicates the topmost level-1 stack, and labels it by s. That is, if $W = (s_1, w_1) \cdots (s_m, w_m)$, then $\mathsf{Push}_2(s)(W) = (s_1, w_1) \cdots (s_m, w_m)(s, w_m)$
- $\mathsf{Pop}_2(s)$: pops the topmost level-1 stack, if it is labelled by s. That is, if $W = (s_1, w_1)(s_2, w_2) \cdots (s_m, w_m)(s, w_{m+1})$ with $m \geqslant 1$, then $\mathsf{Pop}_2(s)(W) = (s_1, w_1)(s_2, w_2) \cdots (s_m, w_m)$. If the topmost level-1 stack is not labelled s, then $\mathsf{pop}_1(s)$ is undefined.

The above defined operations form the set $\mathsf{Op}(S)$.

2-HOPDS syntax. A level-2 HOPDS (2-HOPDS) over a finite alphabet Σ is a tuple $\mathcal{H} = (Q, S, \Delta, q_0, s_0)$ where Q is the finite set of states, S is the set of stack symbols/labels, q_0 is the initial state, s_0 is the initial stack-label, and $\Delta \subseteq Q \times \Sigma \times \mathsf{Op}(S) \times Q$ is the set of transitions.

2-HOPDS semantics. We give the operational semantics of 2-HOPDS. A *configuration* is a pair $C = (q, W)$ where $q \in Q$ is a state and W is a level-2 stack. The set of configurations is denoted $\mathcal{C}$. The *initial* configuration $C_0 = (q_0, W_0)$ is the pair consisting of the initial state and the initial level-2 stack W_0. The initial level-2 stack $W_0 = (s_0, \epsilon)$ contains an empty level-1 stack which is labelled by the initial stack-label s_0. A configuration $C = (q, W)$ is *accepting* is $W = W_0$. Note that we use an empty stack acceptance criteria.

We write $C \overset{\tau}{\Rightarrow} C'$ for configurations $C = (q, W)$, $C' = (q', W')$ and transition $\tau = (q, a, \mathsf{op}(s), q')$, if $W' = \mathsf{op}(s)(W)$. A *run* ρ of $\mathcal{H}$ is an alternating sequence of configurations and transitions, starting from the initial configuration, and conforming to the relation $\Rightarrow$, i.e., $\rho = C_0 \overset{\tau_1}{\Rightarrow} C_1 \overset{\tau_2}{\Rightarrow} C_2 \ldots \overset{\tau_n}{\Rightarrow} C_n$. We say that ρ is an *accepting run* if C_n is accepting.

Level-2 nested words (2-NW). We propose words augmented with nesting relations to capture the behaviours of 2-HOPDS, analogous to nested words for pushdown systems. A level-2 nested word (2-NW) over an alphabet Σ is a tuple $\mathcal{N} = \langle w, \curvearrowright^1, \curvearrowright^2 \rangle$ where w is a word over Σ, and $\curvearrowright^1$ and $\curvearrowright^2$ are binary relations over positions of w matching pops (second coordinate) with the corresponding pushes (first coordinate). Since we have two kinds of stacks, we provide two matching relations, $\curvearrowright^1$ for level-1 and $\curvearrowright^2$ for level-2.

Language of 2-HOPDS as a set of 2-NW. Given a sequence of operations from the set $\mathsf{Op}(S)$, one can extract a unique 2-NW. The nesting relations should match every pop with a corresponding push action (cf. Example 1). This can be intuitively understood if we assume that at every push event, the label that is pushed is tagged with the current position. Then at a pop, the nesting edge should link the position present in the tag to the current position.

Let us describe these matching relations more precisely. Consider a sequence $\mathsf{op}_1 \mathsf{op}_2 \cdots \mathsf{op}_n$ of operations from the set $\mathsf{Op} = \{\mathsf{push}_1, \mathsf{pop}_1, \mathsf{Push}_2, \mathsf{Pop}_2\}$. Notice that the symbols/labels from S of operations have no influence on the matching relations. They are only useful for the semantics of a 2-HOPDS. Hence, we ignore these labels when constructing the matching relations. Instead, we consider level-2 stacks containing the indexes of push operations. Let $W_0 = (0, \varepsilon)$ is the empty level-2 stack. We have $W_0 \overset{\mathsf{op}_1}{\Longrightarrow} W_1 \overset{\mathsf{op}_2}{\Longrightarrow} W_2 \cdots W_{n-1} \overset{\mathsf{op}_n}{\Longrightarrow} W_n$ only if the following holds. If $W_{j-1} = W'(i, w)$ then $W_j = \begin{cases} W'(i, wj) & \text{if } \mathsf{op}_j = \mathsf{push}_1 \\ W'(i, w)(j, w) & \text{if } \mathsf{op}_j = \mathsf{Push}_2 \end{cases}$. If $\mathsf{op}_j = \mathsf{Pop}_2$, then either 1) $W_{j-1} = (0, w)$ (this Pop_2 cannot be matched: the push/pop sequence is invalid), or 2) $W_{j-1} = W_j(i, w)$ for some $i > 0$ and $i \curvearrowright^2 j$. If $\mathsf{op}_j = \mathsf{pop}_1$, then either 1) $W_{j-1} = W'(k, \varepsilon)$ (this pop_1 cannot be matched: the push/pop sequence is invalid), or 2) $W_{j-1} = W'(k, wi)$ for some $i > 0$ and we get $W_j = W'(k, w)$ and $i \curvearrowright^1 j$.

Given a run $\rho = C_0 \overset{\tau_1}{\Rightarrow} C_1 \overset{\tau_2}{\Rightarrow} C_2 \ldots \overset{\tau_n}{\Rightarrow} C_n$ of a 2-HOPDS, the corresponding 2-NW, denoted $\mathsf{NWof}(\rho)$, is defined to be $\langle w, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ where w is the Σ projection of the sequence of transitions in ρ and $\rightsquigarrow^1$ and $\rightsquigarrow^2$ are obtained by the above procedure on the Op projection of the sequence of transitions in ρ. The language of a 2-HOPDS $\mathcal{H}$, denoted $\mathcal{L}(\mathcal{H})$, is the set of 2-NW defined by $\mathcal{L}(\mathcal{H}) = \{\mathsf{NWof}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{H}\}$.

Internal transition. We do not consider internal transitions since these are not relevant for our purposes, but they can be easily incorporated. An internal transition is a tuple $\tau = (q, a, q')$ and it can be fired from any configuration (q, W) without accessing the level-2 stack. Hence, the semantics is given by $(q, W) \overset{\tau}{\Rightarrow} (q', W)$. In a 2-NW, internal transitions would correspond to positions that are not the source or target of a nesting relation.

Top-tests. The classical pushdown systems and higher-order pushdown systems often provide a top-test, i.e., a "guard" that checks whether the topmost symbol (of the topmost) stack is equal to some particular value. In our case, we can think of two versions of top-tests, one for level 1 stack (similar to the classical case), and one for the level 2 stack.

- $\mathsf{top}_1(s)$: checks whether the topmost label of the topmost stack is s. We write $W \models \mathsf{top}_1(s)$ if the level-2 stack is of the form $W = W'(s', ws)$.
- $\mathsf{Top}_2(s)$: checks whether the label of the topmost stack is s. We write $W \models \mathsf{Top}_2(s)$ if the level-2 stack is of the form $W = W'(s, w)$.

Then, we may add top-test transitions to a 2-HOPDS. Let W be a level-2 stack and let $\tau = (q, a, \mathsf{op}(s), q')$ be a level-1 or level-2 top-test transition with $\mathsf{op} \in \{\mathsf{top}_1, \mathsf{Top}_2\}$. If $W \models \mathsf{op}(s)$ then $(q, W) \overset{\tau}{\Rightarrow} (q', W)$ is added to the operational semantics.

As for classical pushdown systems, it is possible to simulate top-tests by internal transitions which do not access the level-2 stack (Proposition 2). The idea is to maintain the top label information faithfully in the state itself. Thus the top-test can be evaluated, depending only on the current state of the configuration, and not on its stack.

Proposition 2. *For every 2-HOPDS $\mathcal{H}$ with top-tests, we can construct in polynomial time another 2-HOPDS $\mathcal{H}'$ with internal transitions but no top-tests, such that $\mathcal{H}$ and $\mathcal{H}'$ have isomorphic configuration graphs.*

Remark 3. There is another advantage of replacing top-tests with internal transitions. To represent a top-test in a 2-NW, one would naïvely add a nesting edge from the level-1 or level-2 push to the corresponding top-test. This would result in a more complex 2-NW, making verification harder. By replacing top-tests with internal transitions that do not access the stack, we do not provide matching edges to top-tests. Thus we gain a higher coverage for our under-aproximation defined in Section 4.

Characterisation of the matching relations. We give now an alternate characterization of the matching relations, which does not rely on the extra tagging. The characterisation is given in Lemma 4. Due to space constraints, the correctness of this characterisation is given in Appendix B. Here we explain how this characterisation can be used to recover the nesting relations from a push/pop sequence. This is the main idea behind `Nestify`.

The idea is to characterize the push/pop sequences that may occur between a matching pair $\mathsf{Push}_2 \rightsquigarrow^2 \mathsf{Pop}_2$ or $\mathsf{push}_1 \rightsquigarrow^1 \mathsf{pop}_1$ using context-free languages. This is simpler for level-2 since in that case level-1 operations may be treated as internal actions. We prove in Lemma 4 (A2) that the language $L_2 \subseteq \mathsf{Op}^*$ of push/pop sequences that may occur between a matching $\mathsf{Push}_2/\mathsf{Pop}_2$ pair is defined by the context-free grammar:

$$S_2 \to \varepsilon \mid S_2\mathsf{push}_1 \mid S_2\mathsf{pop}_1 \mid S_2\mathsf{Push}_2 S_2 \mathsf{Pop}_2 \,.$$

Given a sequence $\mathsf{op}_1\mathsf{op}_2 \cdots \mathsf{op}_n \in \mathsf{Op}^*$ we may compute by induction the pairs of positions in $\rightsquigarrow^2$ as follows. Assume the level-2 matching pairs have been computed up to $j-1$ and that $\mathsf{op}_j = \mathsf{Pop}_2$. From $j-1$, we walk backwards as much as possible in the sequence, skipping the level-1 operations (push_1 or pop_1) and jumping to the (already computed) matching Push_2 when seeing a Pop_2. By induction and Lemma 4 (A2), the skipped sequence $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1}$ is generated by the above grammar for L_2. If $i \geqslant 1$ then $\mathsf{op}_i = \mathsf{Push}_2$ by maximality of the skipped sequence. We get $i \rightsquigarrow^2 j$ by Lemma 4 (A2). If $i = 0$ then $\mathsf{op}_j = \mathsf{Pop}_2$ is not matched.

We turn now to the more complex level-1 matching. Lemma 4 (A1) establishes that the language $L_1 \subseteq \mathsf{Op}^*$ of push/pop sequences that may occur between a matching $\mathsf{push}_1/\mathsf{pop}_1$ pair is defined by the context-free grammar:

$$S_1 \to \varepsilon \mid S_1\mathsf{Push}_2 \mid S_1\mathsf{Push}_2 S_2 \mathsf{Pop}_2 \mid S_1\mathsf{push}_1 S_1 \mathsf{pop}_1 \,.$$

Notice the use of S_2 between the matching pair $\mathsf{Push}_2/\mathsf{Pop}_2$. As above, we derive from this grammar an algorithm to compute inductively the matching pairs of positions of $\rightsquigarrow^1$. Assume the level-1 and level-2 matching pairs of positions have been computed up to $j-1$ and that $\mathsf{op}_j = \mathsf{pop}_1$. From $j-1$, we walk backwards as much as possible in the sequence, skipping Push_2 operations and jumping to the (already computed) matching Push_2 (resp. push_1) when seeing a Pop_2 (resp. pop_1). By induction and Lemma 4, the skipped sequence $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1}$ is generated by the above grammar for L_1. If $i \geqslant 1$ then $\mathsf{op}_i = \mathsf{push}_1$ by maximality of the skipped sequence. We get $i \rightsquigarrow^1 j$ by Lemma 4 (A1). If $i = 0$ then $\mathsf{op}_j = \mathsf{pop}_1$ is not matched.

Lemma 4. *Let $\mathsf{op}_1\mathsf{op}_2 \cdots \mathsf{op}_j \in \mathsf{Op}^*$ be a valid push/pop sequence. The following holds:*

A1. We have $i \rightsquigarrow^1 j$ iff $\mathsf{op}_i = \mathsf{push}_1$, $\mathsf{op}_j = \mathsf{pop}_1$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_1$.
A2. We have $i \rightsquigarrow^2 j$ iff $\mathsf{op}_i = \mathsf{Push}_2$, $\mathsf{op}_j = \mathsf{Pop}_2$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_2$.

The above characterisation and the backward paths are used in the linear time algorithm `Nestify` (http://www.lsv.ens-cachan.fr/~gastin/hopda/). In the next section we express both these backward walks in PDL (Example 6, π_{mypush2} and π_{mypush1} respectively).

3 PDL and MSO over level-2 nested words

We introduce two logical formalisms for specifications over 2-NW. The first one is propositional dynamic logic which essentially navigates through the edges of a 2-NW, checking positional properties on the way. The second one is the yardstick monadic second-order logic, which extends MSO over words with the $\rightsquigarrow^1$ and $\rightsquigarrow^2$ binary relations.

Propositional Dynamic Logic with converse and loop (LCPDL) can express properties of nodes as boolean combinations of the existence of paths and loops. The paths are built using regular expressions over the edge relations (and their converses) of level-2 nested words. We also provide an additional relation $\hookrightarrow$ and its converse. The relation $\hookrightarrow$ links a level-1 pop with the next level-1 pop that is also matched to the same push. The syntax of the node formulas φ and path formulas π of LCPDL are given by

$$\varphi := a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Loop}(\pi)$$
$$\pi := \{\varphi\}? \mid \rightarrow \mid \leftarrow \mid \rightsquigarrow^1 \mid \leftsquigarrow^1 \mid \rightsquigarrow^2 \mid \leftsquigarrow^2 \mid \hookrightarrow \mid \hookleftarrow \mid \pi \cdot \pi \mid \pi + \pi \mid \pi^*$$

where $a \in \Sigma$. The node formulas are evaluated on positions of a level-2 nested word, whereas path formulas are evaluated on pairs of positions. The semantics is as expected. It is given below for some interesting cases (i, j, i', j' vary over positions of a 2-NW $\mathcal{N} = \langle w, \rightsquigarrow^1, \rightsquigarrow^2\rangle$):

$\mathcal{N}, i \models \langle\pi\rangle\varphi$ if $\mathcal{N}, i, j \models \pi$ and $\mathcal{N}, j \models \varphi$ for some j

$\mathcal{N}, i \models \mathsf{Loop}(\pi)$ if $\mathcal{N}, i, i \models \pi$

$\mathcal{N}, i, j \models \{\varphi\}?$ if $i = j$ and $\mathcal{N}, i \models \varphi$

$\mathcal{N}, i, j \models \rightarrow$ if j is the successor position of i in the word w.

$\mathcal{N}, i, j \models \hookrightarrow$ if there exists i' such that $\mathcal{N}, i', i \models \rightsquigarrow^1$ and $\mathcal{N}, i', j \models \rightsquigarrow^1$ and there is no j' between i and j such that $\mathcal{N}, i', j' \models \rightsquigarrow^1$

$\mathcal{N}, i, j \models \hookleftarrow$ if $\mathcal{N}, j, i \models \hookrightarrow$

An LCPDL *sentence* is a boolean combination of atomic sentences of the form $\mathsf{E}\varphi$. An atomic LCPDL sentence is evaluated on a level-2 nested word $\mathcal{N}$. We have $\mathcal{N} \models \mathsf{E}\varphi$ if there exists a position i of $\mathcal{N}$ such that $\mathcal{N}, i \models \varphi$.

We use common abbreviations to include true, false, conjunction, implication, 'φ holds after all π paths' ($[\pi]\varphi$) etc. We simply write $\langle\pi\rangle$ instead of $\langle\pi\rangle\mathsf{true}$ to check the existence of a π path from the current position.

The *satisfiability problem* of LCPDL asks: Given an LCPDL sentence ϕ, does there exist a 2-NW $\mathcal{N}$ such that $\mathcal{N} \models \phi$? The *model checking problem* of 2-HOPDS against LCPDL asks, given an LCPDL sentence ϕ and a 2-HOPDS $\mathcal{H}$, whether for all 2-NW $\mathcal{N}$ in $\mathcal{L}(\mathcal{H})$ it holds that $\mathcal{N} \models \phi$.

Example 5. The following formula captures the summary-until modality of the FO-complete temporal logic over nested words NWTL [4]. A summary-until formula $p\,\mathsf{U}^{\mathrm{s}}\,q$ holds at a position i of a nested word if there is a later position j which satisfies q, and p holds on all positions i' on the summary path from i to j. A *summary path* from i to j is the shortest path from i to j traversing $\rightarrow$ and $\rightsquigarrow^2$ edges (since we consider here classical nested words, we assume that there are no $\rightsquigarrow^1$ edges).

$$\varphi_{\mathsf{U}^{\mathrm{s}}} = \langle(\{p\}? \cdot (\rightsquigarrow^2 + \rightarrow))^*\rangle q$$

Note that, the path in the formula above is existentially quantified and for nested words, the shortest path linking two nodes is a subpath of any path linking these two nodes. Hence, if the above is true for some path, it is true for the shortest path. In fact, all NWTL modalities can be expressed in LCPDL [8,9].

Example 6. The following formula identifies the matching push of a pop node by a backward path as described in Section 2. This is the main idea used in `Nestify`.

$$\pi_{\mathsf{mypush2}} = \leftarrow \cdot \left((\leftsquigarrow^2 + \{\neg\langle\leftsquigarrow^2 + \rightsquigarrow^2\rangle\}?) \cdot \leftarrow\right)^* \cdot \{\langle\rightsquigarrow^2\rangle\}?$$

$$\pi_{\mathsf{mypush1}} = \leftarrow \cdot \left((\leftsquigarrow^1 + \leftsquigarrow^2 + \{\neg\langle\leftsquigarrow^1 + \leftsquigarrow^2 + \rightsquigarrow^1\rangle\}?) \cdot \leftarrow\right)^* \cdot \{\langle\rightsquigarrow^1\rangle\}?$$

Notice that the forward steps $\rightsquigarrow^1$ and $\rightsquigarrow^2$ are used only in tests for determining the type of a node. The matching push of a level-1 pop should coincide with the one dictated by π_{mypush}. The following sentence characterises the set of all 2-NW by forbidding the existence of an invalid $\rightsquigarrow^1$ or $\rightsquigarrow^2$ edge. Note that it is a valid sentence.

$$\phi_{\text{2-NW}} = \neg\mathsf{E}\left((\langle\leftsquigarrow^1\rangle \wedge \neg\mathsf{Loop}(\pi_{\mathrm{mypush1}} \cdot \rightsquigarrow^1)) \vee (\langle\leftsquigarrow^2\rangle \wedge \neg\mathsf{Loop}(\pi_{\mathrm{mypush2}} \cdot \rightsquigarrow^2))\right)$$

Monadic Second-order Logic over 2-NW, denoted MSO(2-NW), extends the classical MSO over words with two binary predicates $\rightsquigarrow^1$ and $\rightsquigarrow^2$. A formula ϕ can be written using the following syntax:

$$\phi := a(x) \mid x < y \mid x \rightsquigarrow^1 y \mid x \rightsquigarrow^2 y \mid \phi \vee \phi \mid \neg\phi \mid \exists x\,\phi \mid \exists X\phi \mid x \in X$$

where $a \in \Sigma$, x, y are first-order variables and X is a second-order variable. The semantics is as expected. As in the case of LCPDL, we use common abbreviations.

The *satisfiability problem* of MSO(2-NW) asks, given an MSO(2-NW) sentence ϕ, whether there exists a 2-NW $\mathcal{N}$ such that $\mathcal{N} \models \phi$. The *model checking problem* of 2-HOPDS against MSO(2-NW) asks, given an MSO(2-NW) sentence ϕ and a 2-HOPDS $\mathcal{H}$, whether for all 2-NW $\mathcal{N}$ in $\mathcal{L}(\mathcal{H})$ it holds that $\mathcal{N} \models \phi$.

Example 7. The path expression $\hookrightarrow$ can be easily expressed in the first-order fragment as a formula with two free variables.

$$\phi_{\hookrightarrow}(x, y) = \exists z\ \left(z \rightsquigarrow^1 x \wedge z \rightsquigarrow^1 y \wedge \neg\exists z'\,(z \rightsquigarrow^1 z' \wedge x < z' < y)\right)$$

Example 8. The set of all 2-NW can be characterised in MSO(2-NW). It essentially says that $<$ is a total order, and that the matching relations are valid. For the latter it takes the idea from Example 6.

Example 9. Given a 2-HOPDS $\mathcal{H}$, its language $\mathcal{L}(\mathcal{H})$ can be characterised by an MSO(2-NW) formula. The formula essentially guesses the transitions taken at every position using second-order variables and verifies that it corresponds to a valid accepting run.

Both specification formalisms that we introduced in this section are powerful enough to render satisfiability and model checking problems undecidable.

Theorem 10. *The satisfiability problem and model checking problem for LCPDL and MSO(2-NW) (even weaker first-order logic) are undecidable.*

Proof (Sketch). 2-NW can easily embed unbounded grids. The embedding only uses level-2 stacks of depth bounded by 1. The vertices of the grid correspond to level-1 pops. Each row is enclosed within a $\rightsquigarrow^2$ edge. The up and down relations of the grid can be traced back by simply moving right ($\rightarrow$) or left ($\leftarrow$) respectively, staying within the enclosed $\rightsquigarrow^2$ edge of the 2-NW. The right and left relations on the grid is traced by moving to the next level-1 pop matching the same level-1 push ($\hookrightarrow$) or to the previous such level-1 pop ($\hookleftarrow$) respectively. Note that, it is easy to check whether one is in the leftmost/rightmost column or in the topmost/bottommost row. Notice that these relations can be expressed in the first-order fragment of the logic. The undecidability of satisfiability follows. Since we can construct a universal 2-HOPDS which accepts all 2-NW, model checking is also undecidable. More details of the encoding may be found in Appendix C.

Remark 11. We showed that PDL and first-order logic are undecidable for level-2 nested words. As mentioned before level-2 nested words have close connections with nested trees. First-order logic, and μ-calculus are decidable over nested trees, when the signature does not contain the order $<$ (but only the successor $\rightarrow$) [5, 20]. As soon as transitive closure is introduced in the logic, satisfiability and model checking becomes undecidable.

4 Bounded-pop 2-NW

In this section we will define an under-approximation of the level-2 nested words which regains decidability of the verification problems discussed in Section 3.

Bounded-pop 2-NW We define an under-approximation of 2-HOPDS where a pushed symbol may be popped at most a bounded number of times. This does not limit the number of times a level-1 stack may be copied, nor the height of any a level-1 or level-2 stack. The restriction is simply that each element in a level-1 stack can be popped in at most a bounded number of copies. Note that, by definition, a level-2 push is popped at most once. This is the case for classical pushdown systems as well. On level-2 nested words, our restriction amounts to bounding the number of $\rightsquigarrow^1$ partners that a push may have. Let β denote this bound for the rest of the paper. The class of 2-NW in which every level-1 push has at most β many matching pops is called *β-pop-bounded level-2 nested words*. It is denoted 2-NW(β).

Bounded-pop model checking The under-approximate satisfiability problem and model checking problems are defined as expected. The bounded-pop satisfiability problem of LCPDL asks, given an LCPDL sentence ϕ and a natural number β, whether there exists a 2-NW $\mathcal{N} \in$ 2-NW(β) such that $\mathcal{N} \models \phi$. The bounded-pop model checking problem of 2-HOPDS against LCPDL asks, given a LCPDL sentence ϕ, a 2-HOPDS $\mathcal{H}$ and a natural number β, whether for all 2-NW $\mathcal{N} \in \mathcal{L}(\mathcal{H}) \cap$ 2-NW(β) it holds that $\mathcal{N} \models \phi$. The bounded-pop satisfiability problem of MSO(2-NW) and bounded-pop model checking problem of 2-HOPDS against MSO(2-NW) are defined analogously.

We show that the above problems are decidable. The complexity of our decision procedures for all these problems is polynomial in the size of the 2-HOPDS (where applicable), but exponential in the bound β. The complexity is exponential in the size of the LCPDL formula, but non-elementary in the size of the MSO(2-NW) formula.

Theorem 12. *The following problems are decidable.*
P1 Bounded-pop satisfiability problem of MSO(2-NW)
P2 Bounded-pop model checking problem of 2-HOPDS against MSO(2-NW)
P3 Bounded-pop satisfiability problem of LCPDL
P4 Bounded-pop model checking problem of 2-HOPDS against LCPDL
Further, problems P3 and P4 are ExpTime*-Complete.*

The rest of the paper is devoted to proving Theorem 12. We first lift the notion of split-decomposition and split-width to level-2 nested words and show that 2-NW(β) have a linear bound on split-width (Section 5). Then we show that 2-NW of bounded split-width have bounded special tree-width and discuss a tree interpretation followed by decision procedures.

Lower bound. 2-HOPDS can easily simulate nested-word automata (NWA) [7] by not using any level-1 stack operations. Push_2 and Pop_2 will play the role of push and pop of NWA. Satisfiability of PDL over nested words, and model checking of PDL against NWA are known to be ExpTime-Complete [8, 9]. The lower bound of P3 and P4 follows.

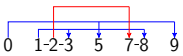
5 Split-width

We introduce the notion of split-width and show that pop-bounded level-2 nested words have bounded split-width. In the next section, we will show that nested words in 2-NW(β) can be interpreted in binary trees, which is the core of our decision procedures.

A Split-2-NW is a 2-NW in which the underlying word has been split in several factors. Formally, a split-2-NW is a tuple $\overline{\mathcal{N}} = \langle u_1, \ldots, u_m, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ such that $\mathcal{N} = \langle u_1 \cdots u_m, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ is a 2-NW. The number m of factors in a split-2-NW is called it's width. A 2-NW is a split-2-NW of width one.

A split-2-NW can be seen as a labelled graph whose vertices are the positions of the underlying word (concatenation of the factors) and we have level-1

edges $\rightsquigarrow^1$, level-2 edges $\rightsquigarrow^2$ and successor edges $\rightarrow$ between consecutive positions *within* a factor. We say that a split-2-NW is connected if the underlying graph is *connected*. If a split-2-NW is not connected, then its connected components form a partition of its factors.

Example 13. Consider the split-2-NW 

. It has two connected components, and its width is five.

The split game is a two-player turn based game $\mathcal{G} = \langle V = V_\forall \uplus V_\exists, E \rangle$ where Eve's positions $V_\exists$ consists of all connected split-2-NWs, and Adam's positions $V_\forall$ consists of all non connected split-2-NWs. The edges E of $\mathcal{G}$ reflect the moves of the players. Eve's moves consist in splitting a factor in two, i.e., removing one successor edge in the graph. Adam's moves amounts to choosing a connected component.

A position is winning for Eve if the split-2-NW is atomic, meaning that all its factors are singletons. An atomic split-2-NW contains either a single internal event, or, a single push with all its corresponding pops.

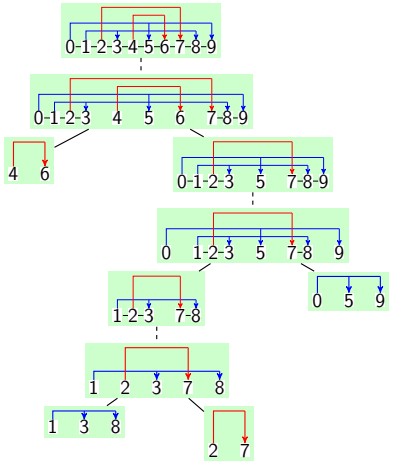
A play on a split-2-NW $\overline{\mathcal{N}}$ is path in $\mathcal{G}$ starting from $\overline{\mathcal{N}}$ to an atomic split-2-NW. The cost of the play is the maximum width of any split-2-NW encountered in the path. Eve's objective is to minimize the cost and Adam's objective is to maximize the cost.

A strategy for Eve from a split-2-NW $\overline{\mathcal{N}}$ can be described with a *split-tree* T which is a binary tree labelled with split-2-NW satisfying:

1. The root is labelled by $\overline{\mathcal{N}}$.
2. Leaves are labelled by atomic split-2-NW.
3. Eve's move: Each unary node is labelled with some connected split-2-NW $\overline{\mathcal{N}}$ and its child is labelled with $\overline{\mathcal{N}'}$ obtained by splitting a factor of $\overline{\mathcal{N}}$ in two. Thus, $\mathsf{width}(\overline{\mathcal{N}'}) = 1 + \mathsf{width}(\overline{\mathcal{N}})$.
4. Adam's move: Each binary node is labelled with some non connected split-2-NW $\overline{\mathcal{N}} = \overline{\mathcal{N}}_1 \uplus \overline{\mathcal{N}}_2$ where $\overline{\mathcal{N}}_1$ and $\overline{\mathcal{N}}_2$ are the labels of its children. Note that $\mathsf{width}(\overline{\mathcal{N}}) = \mathsf{width}(\overline{\mathcal{N}}_1) + \mathsf{width}(\overline{\mathcal{N}}_2)$.

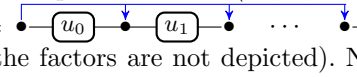
Remark 14. Any subtree of a split-tree is also a split-tree.

The *width* if a split-tree T, denoted $\mathsf{width}(T)$, is the maximum width of the split-2-NWs labelling the tree T. The *split-width* of a split-2-NW $\overline{\mathcal{N}}$ is the minimal width of all split-trees for $\overline{\mathcal{N}}$.



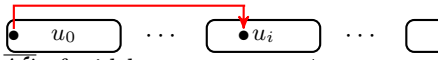
Example 15. A split-tree is depicted above. The intermediate nodes in a sequence of moves by Eve are not shown. The complete split-tree is depicted in Appendix E. The width of the split-2-NW labelling binary nodes are four. Hence the split-width of the split-2-NW labelling the root is at most four.

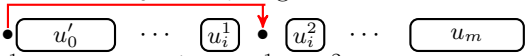
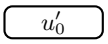
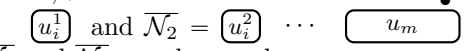
Theorem 16. *Nested words in 2-NW(β) have split-width bounded by $k = 2\beta+2$.*

Proof. First, we say that a split-word $\overline{\mathcal{N}} = \langle u_0, u_1, \ldots, u_m, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ with $m \leqslant \beta$ is *good* if $\mathsf{Close}(\overline{\mathcal{N}}) =$  is a valid 2-NW (nesting edges between the factors are not depicted). Notice that any 2-NW is vacuously good.

Our strategy is to decompose good split-words into atomic split-words or smaller good split-words so that we can proceed inductively. Good split-words have width at most $\beta + 1$. On decomposing a good split-word to obtain smaller good split-words, we may temporarily generate (not necessarily good) split-words of higher width, but we never exceed $k = 2\beta + 2$.

Consider any good split-word $\overline{\mathcal{N}}$. We have two cases: either it begins with a push_1, or it begins with a Push_2.

If $\overline{\mathcal{N}}$ begins with a Push_2: $\overline{\mathcal{N}} =$  . We split factors u_0 and u_i to get $\overline{\mathcal{N}'}$ of width at most $m + 4$.

$\overline{\mathcal{N}'} =$  Note that, there cannot be any $\rightsquigarrow^1$ edges from $u'_0, \ldots, u^1_i$ to $u^2_i, \ldots, u_m$ since the duplicated stack is lost at the Pop_2. Further, there cannot be any $\rightsquigarrow^2$ edges from $u'_0, \ldots, u^1_i$ to $u^2_i, \ldots, u_m$ since $\rightsquigarrow^2$ is well-nested. Hence, $\overline{\mathcal{N}'}$ can be divided into atomic  and split-words $\overline{\mathcal{N}_1} =$  and $\overline{\mathcal{N}_2} =$  . Since $\overline{\mathcal{N}}$ is good, we can prove that $\overline{\mathcal{N}_1}$ and $\overline{\mathcal{N}_2}$ are also good.

Before moving to the more involved case where $\overline{\mathcal{N}}$ begins with a push_1, we first see a couple of properties of 2-NW which become handy.

The *context-pop* of a level-1 pop at position x, denoted $\mathsf{ctxt\text{-}pop}(x)$, is the position of the level-2 pop of the innermost $\rightsquigarrow^2$ edge that encloses x. If $z \rightsquigarrow^2 y$ and $z < x < y$ and there is no $z' \rightsquigarrow^2 y'$ with $z < z' < x < y' < y$, then $\mathsf{ctxt\text{-}pop}(x) = y$. If x does not have a context-pop, then it is *top-level*.

Consider any 2-NW $\langle a_1 a_2 w, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ beginning with two level-1 pushes with corresponding pops at positions $y^1_1, \ldots, y^1_m$ and $y^2_1, \ldots, y^2_n$ respectively. Then for each y^1_i, there is a y^2_j with $y^2_j < y^1_i$ such that either $\mathsf{ctxt\text{-}pop}(y^1_i) \leqslant \mathsf{ctxt\text{-}pop}(y^2_j)$ or y^2_j is top-level. We call this property *existence of covering pop* for later reference.

Consider any 2-NW $\langle aw, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ beginning with a level-1 push with corresponding pops at positions $y_1, \ldots, y_m$. The *context-suffix* of y_i, denoted by $\mathsf{ctxt\text{-}suffix}(y_i)$ is the factor of w strictly between y_i and $\mathsf{ctxt\text{-}pop}(y_i)$, both y_i and $\mathsf{ctxt\text{-}pop}(y_i)$ not included. If y_i is top-level, then $\mathsf{ctxt\text{-}suffix}(y_i)$ is the biggest suffix of w not including y_i. Notice that for each i, $\mathsf{ctxt\text{-}suffix}(y_i)$ is not connected to the remaining of w via $\rightsquigarrow^1$ or $\rightsquigarrow^2$ edges. The notion of $\mathsf{ctxt\text{-}suffix}(y_i)$ may be lifted to split-words $\overline{\mathcal{N}} = \langle a u_0, u_1, u, \ldots, u_n, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ also. In this case, $\mathsf{ctxt\text{-}suffix}(y_i)$ may contain several factors. Still $\mathsf{ctxt\text{-}suffix}(y_i)$ is not connected to the remaining factors. Moreover, if $\overline{\mathcal{N}}$ is good, so is $\mathsf{ctxt\text{-}suffix}(y_i)$.

Now we are ready to describe the decomposition for the second case where the good split-word $\overline{\mathcal{N}}$ begins with a push_1. Let $\overline{\mathcal{N}} = \langle a u_0, u_1, u_2, \ldots u_n, \rightsquigarrow^1, \rightsquigarrow^2 \rangle$ beginning with a level-1 push with corresponding pops at positions $y_1, \ldots, y_m$.

Note that $n, m \leqslant \beta$. We proceed as follows. We split at most two factors of $\overline{\mathcal{N}}$ to get $\overline{\mathcal{N}'_m}$ in which $\mathsf{ctxt\text{-}suffix}(y_m)$ is a (collection of) factor(s). Recall that $\mathsf{ctxt\text{-}suffix}(y_m)$ is not connected to other factors. Hence we divide the split-word $\overline{\mathcal{N}'_m}$ to get $\mathsf{ctxt\text{-}suffix}(y_m)$ as a split-word and the remaining as another split-word $\overline{\mathcal{N}_m}$. Note that $\mathsf{ctxt\text{-}suffix}(y_m)$ is good, so we can inductively decompose it. We proceed with $\overline{\mathcal{N}_m}$ (which needs not be good). We split at most two factors of $\overline{\mathcal{N}_m}$ to get $\overline{\mathcal{N}'_{m-1}}$ in which $\mathsf{ctxt\text{-}suffix}(y_{m-1})$ is a (collection of) factor(s). We divide $\overline{\mathcal{N}'_{m-1}}$ to get $\mathsf{ctxt\text{-}suffix}(y_{m-1})$ which is good, and $\overline{\mathcal{N}_{m-1}}$. We proceed similarly on $\overline{\mathcal{N}_{m-1}}$ until we get $\overline{\mathcal{N}_1}$.

Note that the width of $\overline{\mathcal{N}'_i}$ is at most $n + m - i + 3$, and the width of $\overline{\mathcal{N}_i}$ is at most $n + m - i + 2$. Hence the width of this stretch of decomposition is bounded by $n + m + 2$. Since $n, m \leqslant \beta$, the bound $k = 2\beta + 2$ of split-width is not exceeded.

Now we argue that the width of $\overline{\mathcal{N}_1}$ is at most $m + 1$. This is where we need the invariant of being good of our decompostion. Consider $\mathcal{N}_1 = \mathsf{Close}(\overline{\mathcal{N}_1})$, which is a 2-NW since $\overline{\mathcal{N}_1}$ is good. Now, $\mathcal{N}_1$ starts with two level-1 pushes. Using the existence of covering pop property, we deduce that every split/hole in $\overline{\mathcal{N}_1}$ must belong to some $\mathsf{ctxt\text{-}suffix}(y_i)$. Hence, all splits from $\overline{\mathcal{N}}$ are removed in $\overline{\mathcal{N}_1}$, and only m splits corresponding to the removed $\mathsf{ctxt\text{-}suffix}(y_i)$ persist.

We proceed with $\overline{\mathcal{N}_1}$. We make at most $m + 1$ new splits to get $\overline{\mathcal{N}'}$ in which the first push and its pops at positions $y_1, \ldots, y_m$ are singleton factors. The width of $\overline{\mathcal{N}'}$ is at most $2m + 2 \leqslant 2\beta + 2$. Then we divide $\overline{\mathcal{N}'}$ to get atomic split-word consisting of the first push and its pops, and another split-word $\overline{\mathcal{N}''}$. The width of $\overline{\mathcal{N}''}$ is at most $m + 1 \leqslant \beta + 1$. Further $\overline{\mathcal{N}''}$ is good. By induction $\overline{\mathcal{N}''}$ can also be decomposed. □

6 Tree interpretation and decision procedures

In this section, we show that 2-NW of split-width at most k have special tree-width (STW) at most $2k$. We deduce that 2-NW of bounded split-width can be interpreted in special tree terms (STTs), which are binary trees denoting graphs of bounded STW.

A crucial step towards our decision procedures is then to construct a tree automaton $\mathcal{A}^{k\text{-sw}}_{\text{2-NW}}$ which accepts special tree terms denoting graphs that are 2-NW of split-width at most k. The main difficulty is to make sure that the edge relations $\rightsquigarrow^1$ and $\rightsquigarrow^2$ of the graph are well nested. To achieve this with a tree automaton of size $2^{\mathsf{Poly}(k)}$, we use the characterization given by the LCPDL formula $\phi_{\text{2-NW}}$ of Section 3. Similarly, we can construct a tree automaton $\mathcal{A}^{\beta}_{\text{2-NW}}$ of size $2^{\mathsf{Poly}(\beta)}$ accepting STTs denoting nested words in 2-NW(β).

Special tree terms form an algebra to define graphs. A (Σ, Γ)-labelled graph is a tuple $G = \langle V, (E_\gamma)_{\gamma \in \Gamma}, \lambda \rangle$ where $\lambda \colon V \to \Sigma$ is the vertex labelling and $E_\gamma \subseteq V^2$ is the set of edges for each label $\gamma \in \Gamma$. For 2-NW, we have three types of edges, so $\Gamma = \{\rightarrow, \rightsquigarrow^1, \rightsquigarrow^2\}$. The syntax of k-STTs over (Σ, Γ) is given by

$$\tau = (i, a) \mid \mathsf{Add}^{\gamma}_{i,j}\, \tau \mid \mathsf{Forget}_i\, \tau \mid \mathsf{Rename}_{i,j}\, \tau \mid \tau \oplus \tau$$

where $a \in \Sigma$, $\gamma \in \Gamma$ and $i, j \in [k] = \{1, \ldots, k\}$ are colours.

Each k-STT represents a coloured graph $[\![\tau]\!] = (G_\tau, \chi_\tau)$ where G_τ is a (Σ, Γ)-labelled graph and $\chi_\tau \colon [k] \to V$ is a partial injective function assigning a vertex of G to some colours. $[\![(i, a)]\!]$ consists of a single a-labelled vertex with color i. $\mathsf{Add}^{\gamma}_{i,j}$ adds a γ-labelled edge to the vertices colored i and j (if such vertices exist). Forget_i removes color i and $\mathsf{Rename}_{i,j}$ exchanges the colors i and j. Finally, $\oplus$ constructs the disjoing union of the two graphs provided they use different colors. This operation is undefined otherwise. The special tree-width of a graph G is the least k such that $G = G_\tau$ for some $(k+1)$-STT τ.

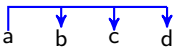
For instance, atomic split-2-NW are denoted by STTs of the following form $\overrightarrow{\mathsf{Add}}^{2}_{1,2}((1, a) \oplus (2, b))$ for level-2 and $\overrightarrow{\mathsf{Add}}^{1}_{1,2} \cdots \overrightarrow{\mathsf{Add}}^{1}_{1,p}((1, a_1) \oplus \cdots \oplus (p, a_p))$ for level-1 push with $p - 1$ pops. We call these STTs *atomic*.

To each split-tree T of width k with root labelled $\overline{\mathcal{N}}$, we associate a $2k$-STT τ such that $[\![\tau]\!] = (\overline{\mathcal{N}}, \chi)$ and all endpoints of factors of $\overline{\mathcal{N}}$ have different colors. Since we have at most k factors, we may use at most $2k$ colors. A leaf of T is labelled with an atomic split-2-NW and we associate the corresponding atomic STT as defined above. At a binary node, assuming that τ_ℓ and τ_r are the STTs of the children, we first define τ'_r by renaming colors in τ_r so that colors in τ_ℓ and τ'_r are disjoint, then we let $\tau = \tau_\ell \oplus \tau'_r$. At a unary node, some factor u is split in two factors u' and u''. The right and left endpoints of u' and u'' respectively are colored, say by i and j, in the STT τ' associated with the child. Then, we add the successor edge $(\overrightarrow{\mathsf{Add}}_{i,j}\, \tau')$. We then forget i if $|u'| > 1$ and j if $|u''| > 1$.

Proposition 17. *There is a tree automaton $\mathcal{A}^{\beta}_{\text{2-NW}}$ of size $2^{\mathsf{Poly}(\beta)}$ accepting k-STTs ($k = 4\beta + 4$) and such that 2-NW$(\beta) = \{G_\tau \mid \tau \in \mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}})\}$.*

The automaton $\mathcal{A}^{\beta}_{\text{2-NW}}$ will accept precisely those k-STTs arising from split-trees as decribed above. The construction of $\mathcal{A}^{\beta}_{\text{2-NW}}$ is given in Appendix F. Since we start from nested words in 2-NW(β), we obtain split-trees of width at most $2\beta + 2$ by Theorem 16. Notice that $\mathcal{A}^{\beta}_{\text{2-NW}}$ needs not accept *all* k-STTs denoting graphs that are nested words in 2-NW(β)

Proposition 18. *For each 2-HOPDS $\mathcal{H}$ we can construct a tree automaton $\mathcal{A}^{\beta}_{\mathcal{H}}$ of size $\mathsf{sizeof}(\mathcal{H})^{\mathsf{Poly}(\beta)}$ such that $\mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}} \cap \mathcal{A}^{\beta}_{\mathcal{H}}) = \{\tau \in \mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}}) \mid G_\tau \in \mathcal{L}(\mathcal{H})\}$.*

The tree automaton $\mathcal{A}^{\beta}_{\mathcal{H}}$ essentially guesses the starting control state and the ending control state for every factor, and verifies that this guess is consistently maintained. At an atomic STT describing a 2-NW of the form 

, if the guessed pair of states for the factors are (q^a_1, q^a_2), (q^b_1, q^b_2), (q^c_1, q^c_2) and (q^d_1, q^d_2) respectively, it ensures that there is a stack label s such that $(q^a_1, a, \mathsf{push}_1(s), q^a_2)$ and $(q^e_1, e, \mathsf{pop}_1(s), q^e_2)$ for each $e \in \{b, c, d\}$ are valid transitions. The size of this tree automaton is $\mathsf{sizeof}(\mathcal{H})^{\mathsf{Poly}(\beta)}$. Thus non-emptiness checking of 2-HOPDS with respect to 2-NW(β) is in ExpTime.

Proposition 19. *For each LCPDL formula ϕ we can construct a tree automaton $\mathcal{A}^{\beta}_{\phi}$ of size $2^{\mathsf{Poly}(\beta,\phi)}$ such that $\mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}} \cap \mathcal{A}^{\beta}_{\phi}) = \{\tau \in \mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}}) \mid G_\tau \in \mathcal{L}(\phi)\}$.*

The idea is to translate the LCPDL formula ϕ to an alternating two-way automaton (A2A) of size $\mathsf{Poly}(\beta, \phi)$. Due to the specific form of the STTs accepted by $\mathcal{A}^{\beta}_{\text{2-NW}}$, it is easy to encode the nesting relations $\rightsquigarrow^1$ and $\rightsquigarrow^2$ with a walking automaton. We can also easily build a walking automaton for the successor relation $\rightarrow$ by tracking the colors until we reach a node labelled $\mathsf{Add}^{\rightarrow}_{i,j}$. One main difficulty is to cope with loops of LCPDL. Here we use the result of [18] for PDL with converse and intersection. In general, this can cause an exponential blow-up in the size of the A2A. But loop is a special case with bounded intersection-width and hence still allows a polynomial sized A2A. Finally, the A2A for ϕ is translated to the normal tree automaton $\mathcal{A}^{\beta}_{\phi}$, causing an exponential blow-up.

We deduce that the bounded-pop satisfiability problem of LCPDL can be solved in exponential time by checking emptiness of $\mathcal{A}^{\beta}_{\text{2-NW}} \cap \mathcal{A}^{\beta}_{\phi}$. Also, the bounded-pop model checking problem of 2-HOPDS against LCPDL can be solved in exponential time by checking emptiness of $\mathcal{A}^{\beta}_{\text{2-NW}} \cap \mathcal{A}^{\beta}_{\mathcal{H}} \cap \mathcal{A}^{\beta}_{\neg\phi}$.

Similarly, for each MSO(2-NW) formula ϕ, we can construct a tree automaton $\mathcal{A}^{\beta}_{\phi}$ such that $\mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}} \cap \mathcal{A}^{\beta}_{\phi}) = \{\tau \in \mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}}) \mid G_\tau \in \mathcal{L}(\phi)\}$. We deduce the decidability of bounded-pop satisfiability problem of MSO(2-NW) and of the bounded-pop model checking problem of 2-HOPDS against MSO(2-NW). This concludes the proof of Theorem 12.

The lower bounds in Theorem 12 implies the following 1) optimality of the decision procedures for 2-NW(β), 2) optimality of the bound on split-width of 2-NW(β) modulo polynomial factors and, 3) optimality of the decision procedures for bounded split-width 2-NW.

7 Conclusion

In this paper we study the linear behaviour of a 2-HOPDS by giving extra structure to words. The specification formalisms can make use of this structure to describe properties of the system. This added structure comes with the cost of undecidable verification problems. We identify an under approximation that regains decidability for verification problems. Our decision procedure makes use of the split-width technique.

This work can only be seen as a first step towards further questions that must be investigated. One direction would be to identify other under-approximations which are orthogonal / more lenient than bounded-pop for decidability. Whether similar results can be obtained for level-n HOPDS is also another interesting future work. The language theory of HOPDS where the language consists of nested-word like structures is another topic of study.

References

1. A. V. Aho. Indexed grammars-an extension of context free grammars. In *8th Annual Symposium on Switching and Automata Theory*, pages 21–31. IEEE, 1967.
2. C. Aiswarya and P. Gastin. Reasoning about distributed systems: WYSIWYG. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, Proceedings*, volume 29 of *Leibniz International Proceedings in Informatics*, pages 11–30. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.
3. C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.
4. R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. *Logical Methods in Computer Science*, 4(4), 2008.
5. R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. In *Computer Aided Verification, 18th International Conference, CAV 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 2006.
6. R. Alur, S. Chaudhuri, and P. Madhusudan. Software model checking using languages of nested trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(5):15, 2011.
7. R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3), 2009.
8. B. Bollig, A. Cyriac, P. Gastin, and M. Zeitoun. Temporal logics for concurrent recursive programs: Satisfiability and model checking. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 132–144. Springer, 2011.
9. B. Bollig, A. Cyriac, P. Gastin, and M. Zeitoun. Temporal logics for concurrent recursive programs: Satisfiability and model checking. *Journal of Applied Logic*, 12(4):395–416, 2014.
10. C. H. Broadbent. Prefix rewriting for nested-words and collapsible pushdown automata. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, Proceedings, Part II*, volume 7392, pages 153–164. Springer, 2012.
11. T. Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003.
12. A. Carayol, M. Hague, A. Meyer, C.-H. Luke Ong, and O. Serre. Winning regions of higher-order pushdown games. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008,*, pages 193–204. IEEE Computer Society, 2008.
13. A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.
14. Bruno Courcelle. Special tree-width and the verification of monadic second-order graph pr operties. In *FSTTCS*, volume 8 of *LIPIcs*, pages 13–29, 2010.

15. A. Cyriac. *Verification of Communicating Recursive Programs via Split-width*. Phd thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, 2014. http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/cyriac-phd14.pdf.
16. A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012.
17. J. Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95(1):21–75, 1991.
18. S. Göller, M. Lohrey, and C. Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *The Journal of Symbolic Logic*, 74(1):279–314, 2009.
19. S. A. Greibach. Full AFLs and nested iterated substitution. *Information and Control*, 16(1):7–35, 1970.
20. A. Kartzow. First-order logic on higher-order nested pushdown trees. *ACM Transactions on Computational Logic (TOCL)*, 14(2):8, 2013.
21. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.
22. A. N. Maslov. Multilevel stack automata. *Problemy Peredachi Informatsii*, 12(1):55–62, 1976.
23. J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.
24. M. Y. Vardi. Reasoning about the past with two-way automata. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.
25. Wikipedia. Higher-order function — wikipedia, the free encyclopedia, 2015. [Online; accessed 12-July-2015], from https://en.wikipedia.org/w/index.php?title=Higher-order_function&oldid=669089706.

A Top tests

Proposition 2. *For every 2-HOPDS $\mathcal{H}$ with top-tests, we can construct in polynomial time another 2-HOPDS $\mathcal{H}'$ with internal transitions but no top-tests, such that $\mathcal{H}$ and $\mathcal{H}'$ have isomorphic configuration graphs.*

Proof. Let $\mathcal{H} = (Q, S, \Delta, q_0, s_0)$ be a 2-HOPDS with top-tests. We construct $\mathcal{H}' = (Q', S', \Delta', q_0', s_0')$ as follows. We let $S_\perp = S \uplus \{\perp\}$ and we write $W \models \mathsf{top}_1(\perp)$ if the topmost level-1 stack of W is empty.

- $Q' = Q \times S_\perp \times S_\perp$. A state (q, s_1, s_2) of $\mathcal{H}'$ will be used for configurations (q, W) of $\mathcal{H}$ such that $W \models \mathsf{top}_1(s_1)$ and $W \models \mathsf{Top}_2(s_2)$. We let $q_0' = (q_0, \perp, s_0)$.
- $S' = (S_\perp \times S_\perp) \cup (S_\perp \times S_\perp \times S_\perp)$. In $\mathcal{H}'$, *symbols in* level-1 stacks are taken from $S_\perp \times S_\perp$, and *labels of* level-1 stacks are taken from $S_\perp \times S_\perp \times S_\perp$. A level-1 stack $w = s_1 s_2 \cdots s_m$ of $\mathcal{H}$ will be encoded as $w' = (\perp, s_1)(s_1, s_2) \cdots (s_{m-1}, s_m)$ in $\mathcal{H}'$. A label (s_1, s_2, s_3) of a level-1 stack of $\mathcal{H}'$ represents a label s_3 of the corresponding level-1 stack of $\mathcal{H}$ and also indicates that the label of the previous level-1 stack is s_2 and that its topmost symbol is s_1. Indeed, $s_1 = s_2 = \perp$ if there is no previous level-1 stack, and $s_1 = \perp$ if the previous level-1 stack is empty. We let $s_0' = (\perp, \perp, s_0)$.
- Δ' is defined as follows:
  1. $((q, s_1, s_2), a, \mathsf{Push}_2(s_1, s_2, s_3), (q', s_1, s_3)) \in \Delta'$ if $(q, a, \mathsf{Push}_2(s_3), q') \in \Delta$. Note that, the top labels (s_1, s_2) at both level-1 and level-2 are also pushed along with the new level-2 label s_3. In the new state, the top label s_1 at level-1 is preserved and the top label at level-2 is updated to s_3.
  2. $((q, s_1, s_2), a, \mathsf{push}_1(s_1, s_3), (q', s_3, s_2)) \in \Delta'$ if $(q, a, \mathsf{push}_1(s_3), q') \in \Delta$. This case is similar, but we don't push the top label at level-2 into a level-1 stack.
  3. $((q, s_1, s_2), a, \mathsf{Pop}_2(s_3, s_4, s_2), (q', s_3, s_4)) \in \Delta'$ if $(q, a, \mathsf{Pop}_2(s_2), q') \in \Delta$.
  4. $((q, s_1, s_2), a, \mathsf{pop}_1(s_3, s_1), (q', s_3, s_2)) \in \Delta'$ if $(q, a, \mathsf{pop}_1(s_1), q') \in \Delta$.
  5. $((q, s_1, s_2), a, (q', s_1, s_2)) \in \Delta'$ if $(q, a, \mathsf{top}_1(s_1), q') \in \Delta$.
  6. $((q, s_1, s_2), a, (q', s_1, s_2)) \in \Delta'$ if $(q, a, \mathsf{Top}_2(s_2), q') \in \Delta$.

Note that $\mathcal{H}'$ has internal transitions corresponding to the top-test transitions of $\mathcal{H}$.

Claim. The configuration graph of $\mathcal{H}'$ is isomorphic to that of $\mathcal{H}$.

B Characterisation of the matching relations

Lemma 4. *Let $\mathsf{op}_1 \mathsf{op}_2 \cdots \mathsf{op}_j \in \mathsf{Op}^*$ be a valid push/pop sequence. The following holds:*

A1. We have $i \rightsquigarrow^1 j$ iff $\mathsf{op}_i = \mathsf{push}_1$, $\mathsf{op}_j = \mathsf{pop}_1$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_1$.
A2. We have $i \rightsquigarrow^2 j$ iff $\mathsf{op}_i = \mathsf{Push}_2$, $\mathsf{op}_j = \mathsf{Pop}_2$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_2$.

To prove the correctness of the above, we start with a push/pop sequence $\mathsf{op}_1\mathsf{op}_2\cdots\mathsf{op}_n \in \mathsf{Op}^*$ and we consider as in Section 2 the tagged sequence of level-2 stacks:

$$W_0 \overset{\mathsf{op}_1}{\Longrightarrow} W_1 \overset{\mathsf{op}_2}{\Longrightarrow} W_2 \cdots W_{n-1} \overset{\mathsf{op}_n}{\Longrightarrow} W_n$$

from which we have defined the matching relations $\rightsquigarrow^1$ and $\rightsquigarrow^2$. We start by establishing the following properties.

Lemma 20. *The following holds:*

B1. If $\mathsf{op}_{i+1}\cdots\mathsf{op}_j \in L_2$ and $W_i = W(k,w)$ then $W_j = W(k,w')$ for some w'.
B2. If $W_i = W(i,w)$ and $W_j = W'(i,w')$ then $W' = W$ and $\mathsf{op}_{i+1}\cdots\mathsf{op}_j \in L_2$.
B3. If $\mathsf{op}_{i+1}\cdots\mathsf{op}_j \in L_1$ and $W_i = W(k,w)$ then $W_j = W'(k',w)$ for some W' and k'.
B4. If $W_i = W(k,wi)$ and $W_j = W'(k',w'i)$ then $w' = w$ and $\mathsf{op}_{i+1}\cdots\mathsf{op}_j \in L_1$.

Proof. B1. We proceed by induction on $j-i$. The claim is trivial if $j = i$. Assume now that $j > i$. Following the grammar for L_2, we distinguish the following cases.

- If $\mathsf{op}_j = \mathsf{push}_1$ or $\mathsf{op}_j = \mathsf{pop}_1$ then $\mathsf{op}_{i+1}\cdots\mathsf{op}_{j-1} \in L_2$. By induction, we get $W_{j-1} = W(k,w'')$ for some w''. From the semantics of push_1 and pop_1 we conclude that $W_j = W(k,w')$ for some w'.
- If $\mathsf{op}_j = \mathsf{Pop}_2$ then we find $i < \ell < j$ such that $\mathsf{op}_\ell = \mathsf{Push}_2$, $\mathsf{op}_{i+1}\cdots\mathsf{op}_{\ell-1} \in L_2$ and $\mathsf{op}_{\ell+1}\cdots\mathsf{op}_{j-1} \in L_2$. By induction we deduce that $W_{\ell-1} = W(k,w')$ for some w'. Then we have $W_\ell = W(k,w')(\ell,w')$. Using again the induction hypothesis, we obtain $W_{j-1} = W(k,w')(\ell,w'')$ for some w''. Therefore, $W_j = W(k,w')$.

B2. Notice that since i occurs in W_j we must have $i \leqslant j$. The proof is by induction on $j-i$ and the case $j = i$ is trivial since $\varepsilon \in L_2$. So assume that $j > i$. We make a case distinction on op_j.

- If $\mathsf{op}_j = \mathsf{push}_1$ or $\mathsf{op}_j = \mathsf{pop}_1$ then we have $W_{j-1} = W'(i,w'')$ for some w''. By induction we get $W' = W$ and $\mathsf{op}_{i+1}\cdots\mathsf{op}_{j-1} \in L_2$. We conclude using the rules $S_2 \to S_2\mathsf{push}_1 \mid S_2\mathsf{pop}_1$.
- The case $\mathsf{op}_j = \mathsf{Push}_2$ is not possible since this would imply $W_j = W'(j,w')$ but $i < j$.
- Assume $\mathsf{op}_j = \mathsf{Pop}_2$ and let op_k be the matching Push_2: $k \rightsquigarrow^2 j$. We have $W_k = W_{k-1}(k,v)$ and $W_{j-1} = W_j(k,v') = W'(i,w')(k,v')$. Since i was pushed before k we deduce that $i < k$. By induction we obtain $W_j = W_{k-1}$ and $\mathsf{op}_{k+1}\cdots\mathsf{op}_{j-1} \in L_2$. Since $W_{k-1} = W_j = W'(i,w')$, we may again use induction to get $W' = W$ and $\mathsf{op}_{i+1}\cdots\mathsf{op}_{k-1} \in L_2$. We conclude from the rule $S_2 \to S_2\mathsf{Push}_2 S_2\mathsf{Pop}_2$.

B3. We proceed by induction on $j-i$. The claim is trivial if $j = i$. Assume now that $j > i$. Following the grammar for L_1, we distinguish three cases.

- If $\mathsf{op}_j = \mathsf{Push}_2$ then $\mathsf{op}_{i+1}\cdots\mathsf{op}_{j-1} \in L_1$ and by induction we obtain $W_{j-1} = W'(k',w)$ for some W' and k'. The result follows since $W_j = W_{j-1}(j,w)$.

– If $\mathsf{op}_j = \mathsf{Pop}_2$ then we find $i < k < j$ such that $\mathsf{op}_k = \mathsf{Push}_2$, $\mathsf{op}_{i+1} \cdots \mathsf{op}_{k-1} \in L_1$ and $\mathsf{op}_{k+1} \cdots \mathsf{op}_{j-1} \in L_2$. We have $W_k = W_{k-1}(k, v)$ hence by (B1) we obtain $W_{j-1} = W_{k-1}(k, v')$ and then $W_j = W_{k-1}$. We conclude by induction since $\mathsf{op}_{i+1} \cdots \mathsf{op}_{k-1} \in L_1$.
– If $\mathsf{op}_j = \mathsf{pop}_1$ then we find $i < k < j$ such that $\mathsf{op}_k = \mathsf{push}_1$, $\mathsf{op}_{i+1} \cdots \mathsf{op}_{k-1} \in L_1$ and $\mathsf{op}_{k+1} \cdots \mathsf{op}_{j-1} \in L_1$. By induction we obtain $W_{k-1} = W'(k', w)$ for some W' and k'. Then, $W_k = W'(k', wk)$. Using again the induction hypothesis, we obtain $W_{j-1} = W''(k'', wk)$ for some W'' and k''. We deduce that $W_j = W''(k'', w)$.

B4. Notice that since i occurs in W_j we must have $i \leqslant j$. Again, the proof is by induction on $j - i$ and the case $j = i$ is trivial since $\varepsilon \in L_1$. So assume that $j > i$. We make a case distinction on op_j.

– The case $\mathsf{op}_j = \mathsf{push}_1$ is not possible. Indeed in that case we would have $W_j = W'(k', w'j)$ but $i < j$.
– The case $\mathsf{op}_j = \mathsf{Push}_2$ is easy. In that case, $W_j = W_{j-1}(j, w'i)$ and $W_{j-1} = W''(k'', w'i)$. By induction, we get $w' = w$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_1$ and the result follows from $S_1 \to S_1\mathsf{Push}_2$.
– Assume $\mathsf{op}_j = \mathsf{Pop}_2$ and let op_k be the matching Push_2: $k \rightsquigarrow^2 j$. We have $W_k = W_{k-1}(k, v)$ and $W_{j-1} = W_j(k, v') = W'(k', w'i)(k, v')$. We deduce that $i < k$ and using (B2) we get $W_j = W_{k-1}$ and $\mathsf{op}_{k+1} \cdots \mathsf{op}_{j-1} \in L_2$. Since $W_{k-1} = W_j = W'(k', w'i)$, by induction we obtain $w' = w$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{k-1} \in L_1$. We conclude from the rule $S_1 \to S_1\mathsf{Push}_2 S_2 \mathsf{Pop}_2$.
– The last case is $\mathsf{op}_j = \mathsf{pop}_1$. Let op_k be the matching push_1: $k \rightsquigarrow^2 j$. We have $W_{j-1} = W'(k', w'ik)$. We deduce that $i < k$ and $W_k = W''(k'', w''k)$. By induction we obtain $w'' = w'i$ and $\mathsf{op}_{k+1} \cdots \mathsf{op}_{j-1} \in L_1$. Therefore, $W_{k-1} = W''(k'', w'') = W''(k'', w'i)$ and by induction again we get $w' = w$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{k-1} \in L_1$. We conclude from the rule $S_1 \to S_1\mathsf{push}_1 S_1 \mathsf{pop}_1$.

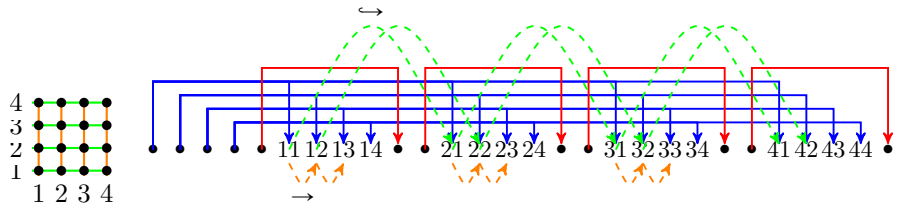
Proof (of Lemma 4). **A1.** (If) Assume that $\mathsf{op}_i = \mathsf{push}_1$, $\mathsf{op}_j = \mathsf{pop}_1$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_1$. Then, $W_i = W(k, wi)$. Using Lemma 20 (B3) we deduce that $W_{j-1} = W'(k', wi)$. By definition of the level-1 nesting relation, we get $i \rightsquigarrow^1 j$.

(Only if) Assume $i \rightsquigarrow^1 j$. By definition of the level-1 matching relation, we get $\mathsf{op}_j = \mathsf{pop}_1$ and $W_{j-1} = W'(k', w'i)$. Also, $\mathsf{op}_i = \mathsf{push}_1$ and $W_i = W(k, wi)$. Using Lemma 20 (B4) we deduce that $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_1$.

A2. (If) Assume that $\mathsf{op}_i = \mathsf{Push}_2$, $\mathsf{op}_j = \mathsf{Pop}_2$ and $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_2$. Then, $W_i = W(i, w)$. Using Lemma 20 (B1) we deduce that $W_{j-1} = W(i, w')$. By definition of the level-2 nesting relation, we get $i \rightsquigarrow^2 j$.

(Only if) Assume $i \rightsquigarrow^2 j$. By definition of the level-2 matching relation, we get $\mathsf{op}_j = \mathsf{Pop}_2$ and $W_{j-1} = W'(i, w')$. Also, $\mathsf{op}_i = \mathsf{Push}_2$ and $W_i = W(i, w)$. Using Lemma 20 (B2) we deduce that $\mathsf{op}_{i+1} \cdots \mathsf{op}_{j-1} \in L_2$.

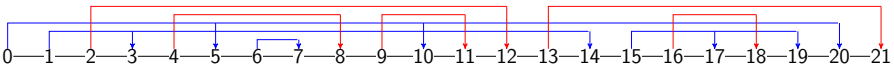
C Encoding grids in 2-NW



In the encoding of a grid in a 2-NW, each row is enclosed with in $\rightsquigarrow^2$. Notice that, while the left and right relations on the grid can be expressed with the word successor, we need $\hookrightarrow$ for the up/down relation on the grid. Though $\hookrightarrow$ edges are not present in the 2-NW, it can be expressed in LCPDL.

D Example showing context-pop and context-suffix

Consider the 2-NW of Example 1.



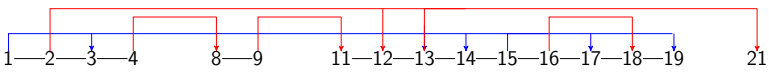
We have,

$\mathsf{ctxt\text{-}pop}(3) = 12$ $\quad$ $\mathsf{ctxt\text{-}pop}(5) = \mathsf{ctxt\text{-}pop}(6) = \mathsf{ctxt\text{-}pop}(7) = 8$
$\mathsf{ctxt\text{-}pop}(10) = 11$ $\quad$ $\mathsf{ctxt\text{-}pop}(17) = 18$
$\mathsf{ctxt\text{-}pop}(14) = \mathsf{ctxt\text{-}pop}(15) = \mathsf{ctxt\text{-}pop}(19) = \mathsf{ctxt\text{-}pop}(20) = 21$

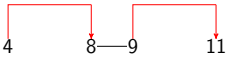
Node 0 and node 1 are top-level.

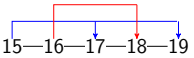
Further, $\mathsf{ctxt\text{-}suffix}(10) = \mathsf{ctxt\text{-}suffix}(20) = \epsilon$, and $\mathsf{ctxt\text{-}suffix}(5) =$ 

.

Consider 

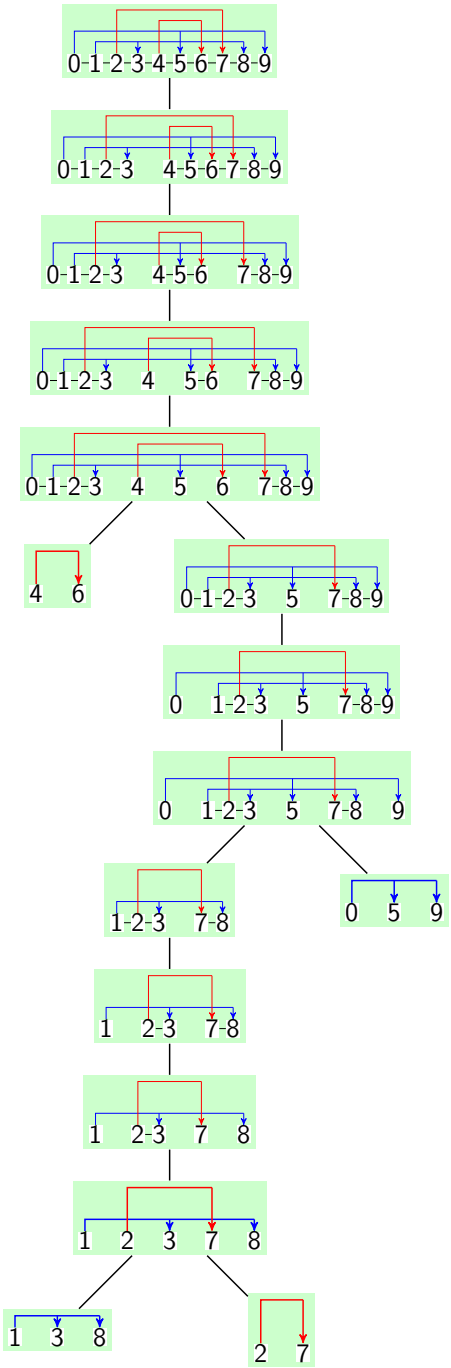
.

Then $\mathsf{ctxt\text{-}suffix}(3) =$ 

and $\mathsf{ctxt\text{-}suffix}(14) =$ 

.

E The complete split-tree of Example 15



F Tree automata

Proposition 17. *There is a tree automaton* $\mathcal{A}^{\beta}_{\text{2-NW}}$ *of size* $2^{\mathsf{Poly}(\beta)}$ *accepting* k*-STTs* ($k = 4\beta + 4$) *and such that* $\text{2-NW}(\beta) = \{G_\tau \mid \tau \in \mathcal{L}(\mathcal{A}^{\beta}_{\text{2-NW}})\}$.

Proof. First, we construct a tree automaton $\mathcal{A}^{\beta}_{\text{word}}$ whose states constists of

- $n \leqslant 2\beta + 2$: Number of factors in the split-2-NW.
- $c_l, c_r : [n] \to [k]$: Functions describing colors of the left and right endpoints of the factor.

To check more easily absence of cycles, factors are numbered according to their guessed ordering in the final $2NW$. The transitions of $\mathcal{A}^{\beta}_{\text{word}}$ ensure the following conditions:

- Atomic STTs: the automaton checks that they denote atomic split nested words in $\text{2-NW}(\beta)$. Then, the number n of factors is at most $\beta + 1$ and c_l, c_r are the identity maps $\mathsf{id}\colon [n] \to [k]$.
- Consider a subterm $\tau = \tau_1 \oplus \tau_2$, we have $n = n_1 + n_2$. We guess how factors of τ_1 and τ_2 will be shuffled on each process and we inherit c_l and c_r accordingly.
- Consider a subterm $\tau = \mathsf{Add}^{\rightarrow}_{i,j}(\tau')$. Let (n', c'_l, c'_r) be the state at τ'. We check that there are factors $x, y \in [n']$ such that $c'_r(x) = i$, $c'_l(y) = j$ and $y = x + 1$ (this checks that the guessed ordering of the factors is correct). The states at τ is easy to compute.
  - $n = n' - 1$
  - $c_l(z) = c'_l(z)$ for $z \leqslant x$ and $c_l(z) = c'_l(z+1)$ for $z < x$
  - $c_r(z) = c'_r(z)$ for $z < x$ and $c_r(z) = c'_r(z+1)$ for $z \leqslant x$
- $\mathsf{Forget}_i\, \tau$: Check that i is not in the image of the mappings c_l and c_r (i.e., $i \notin \mathsf{Im}(c_l) \cup \mathsf{Im}(c_r)$). We always keep the colors of the endpoints of the factors.
- $\mathsf{Rename}_{i,j}$: Update c_l and c_r accordingly.
- Root: Check that n=1 (a single factor).

When an STT τ is accepted by $\mathcal{A}^{\beta}_{\text{word}}$, then the relation $\to$ of the graph G_τ defines a total order on the vertices. Hence, we have an underlying word with some nesting relations $\curvearrowright^1$ and $\curvearrowright^2$. But we did not check that these relations are well-nested. To check this property, we use the LCPDL formula $\phi_{\text{2-NW}}$.

Consider an STT τ accepted by $\mathcal{A}^{\beta}_{word}$, let $[\![\tau]\!] = (G_\tau, \chi_\tau)$ where $G_\tau = (V, \to, \curvearrowright^1, \curvearrowright^2, \lambda)$. we know that $(V, \to, \lambda)$ defines a word in Σ^+. The graph G_τ can be interpreted in τ: we can build walking automata of size $\mathsf{Poly}(k)$ for $\to$, $\curvearrowright^1$, $\curvearrowright^2$ and their converse. Hence we can build an alternating two-way tree automaton of size $\mathsf{Poly}(k)$ checking $\phi_{\text{2-NW}}$. We obtain an equivalent normal tree automaton $\mathcal{A}^{\beta}_{\text{nesting}}$ of size $2^{\mathsf{Poly}(k)}$

The final tree automaton is $\mathcal{A}^{\beta}_{\text{2-NW}} = \mathcal{A}^{\beta}_{\text{word}} \cap \mathcal{A}^{\beta}_{\text{nesting}}$. □