Acta Informatica (2018) 55:153–189
https://doi.org/10.1007/s00236-016-0290-1

ORIGINAL ARTICLE

Reachability analysis of reversal-bounded automata on series–parallel graphs

Rayna Dimitrova[1] · **Rupak Majumdar**[1]

Received: 16 February 2016 / Accepted: 28 November 2016 / Published online: 18 December 2016

Abstract Extensions to finite-state automata on strings, such as multi-head automata or multi-counter automata, have been successfully used to encode many infinite-state non-regular verification problems. In this paper, we consider a generalization of automata-theoretic infinite-state verification from strings to labelled *series–parallel graphs*. We define a model of non-deterministic, 2-way, concurrent automata working on series–parallel graphs and communicating through shared registers on the nodes of the graph. We consider the following verification problem: given a family of series–parallel graphs described by a context-free graph transformation system (GTS), and a concurrent automaton over series–parallel graphs, is some graph generated by the GTS accepted by the automaton? The general problem is undecidable already for (one-way) multi-head automata over strings. We show that a *bounded* version, where the automata make a fixed number of reversals along the graph and use a fixed number of shared registers is decidable, even though there is no bound on the sizes of series–parallel graphs generated by the GTS. Our decidability result is based on establishing that the number of context switches can be bounded and on an encoding of the computation of bounded concurrent automata that allows us to reduce the reachability problem to the emptiness problem for pushdown automata.

1 Introduction

Automata theory studies abstract models of computation and the computational and decision problems associated with them. The long line of research in this area has led to beautiful theoretical results, and has a tremendous impact on algorithmic formal verification, e.g.,

✉ Rayna Dimitrova
rayna@mpi-sws.org

Rupak Majumdar
rupak@mpi-sws.org

[1] Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern and Saarbrücken, Germany

forming the basis of the hugely successful automata-theoretic approach to (finite-state) model checking [25].

The simplest computational models are finite-state machines, describing languages over finite or infinite words or trees. They have been generalized in many ways to extend the set of languages that may be needed to model more complex (non-regular) computational processes. For example, they can be extended with data structures such as stacks or counters, or with multiple heads or tapes and allowing 2-way traversals of the input [22,24], One of the fundamental problems for a class of abstract machines is the emptiness problem, which asks, given an automaton in that class to decide whether the language it accepts is empty.

Since for many interesting extensions of finite-state automata the emptiness problem is undecidable, research in this direction has focused on finding suitable underapproximations for which language emptiness is algorithmically decidable. For example, the *reversal boundedness* restriction bounds the number of reversals of the counters or of stacks, or the number of traversals of the input [13,14,17,18] and the *bounded language* restriction considers behaviors describable by a bounded language [11,12]. Overall, the approach has led to beautiful theoretical results and has also been quite successful in modeling many infinite-state computational models and reasoning about them algorithmically.

Most previous work in automata-theoretic verification has focused on machine models for string or tree languages. In this paper, we study automata, whose languages consist of *series–parallel* graphs whose edges are labelled with a finite alphabet. Series–parallel graphs generalize strings or multi-tape machines by allowing multiple parallel "tracks" to fork off and rejoin at any point. In contrast to general graphs, they retain enough structure, e.g., having a natural "forward" direction, which leads to useful algorithmic properties. Languages over series–parallel graphs can be described using context-free graph transformation systems (GTSs), which describe the dynamic evolution of graphs through local rewrite rules [3,5,6].

We define and study a class of concurrent finite-state automata traversing series–parallel graphs and communicating through state-holding registers located at the nodes of the graph. More precisely, in our model of computation, a fixed number of finite-state machines traverse the nodes of a series–parallel graph. At each step, one of the machines makes a transition that depends on the current state of the machine, the label it reads on one of the incoming or outgoing arcs, and the value of the register stored at its node. The machine moves along the selected edge, updating its state as well as the register. Machines are thus 2-way and non-deterministic, and communicate through the shared registers. A series–parallel graph is accepted if some subset of machines reaches some final states being at the same node of the graph.

The primary motivation for this model is parametrized verification of dynamical concurrent systems in which components interact with each other by traversing the edges of a common graph. A typical example is a distributed train control system operating over a rail-road network. While most work in parametrized verification has focused on systems with fixed communication topologies (e.g., broadcast or token rings) and varying number of processes, our model introduces a different source of parametrisation, allowing to check whether a property holds for every network from a family of networks generated by a series parallel graph grammar. We study the verification of reachability properties for this model, which are of the form: there is no graph generated by the grammar and no execution of the system, in which at some point there are at least two machines in given states located at the same node. Such properties naturally capture mutual exclusion requirements for parts of the network graph, for example that there is at most one machine at a time in some subgraph whose edges are labelled with a given symbol.

We study the emptiness problem: given a context-free GTS defining a language of series–parallel graphs, and a concurrent finite-state automaton, check if there is a graph in the language of the GTS accepted by the automaton. This problem is, not surprisingly, undecidable: for example, we can encode linear bounded automata over strings. We study a natural restriction of the emptiness problem by restricting the number of reversals along the computation and by putting a bound on the number of shared registers in the graph. With these two restrictions, we show that the emptiness problem is decidable and can be reduced to the emptiness problem for pushdown automata. Note that even with the restrictions, the problem is infinite-state because there is no a priori bound on the size of the series–parallel graphs generated by the GTS.

The reduction is based on two technical observations. First, when the number of reversals and the number of registers are fixed, there is a bound on the number of parallel tracks in the graph that needs to be tracked. We also establish a bound on the number of different times each machine moves along the run (although the length of the run may be unbounded). Second, using the bounds above, we construct a large alphabet that tracks valid runs of the machines on a valid graph generated by the GTS. We do this in several steps. We construct a pushdown automaton that checks that a word is a valid representation of a subgraph of a graph generated by the context-free GTS. We construct a set of nondeterministic finite automata, one for each machine, that checks that the word encodes a correct run of that machine along the graph. Finally, we construct another nondeterministic finite automaton that checks that the run is accepted by the concurrent automaton. Some graph generated by the GTS is accepted if the intersection of all these automata is non-empty.

Other related work The automata-theoretic approach is often called *regular* model checking, when applied to parameterized verification [1]. An extensive study of the decidability of several verification problems for classes of GTSs was carried out in [5]. The problems considered there are *reachability* of a given graph, *coverability* (reachability of a graph that contains a given graph as a subgraph) and *existential coverability*, which asks whether there exists an initial graph such that the answer to the coverability problem is positive. The classes of GTSs they investigate are defined by structural restrictions on the set of transformation rules. Classes with decidable coverability problem are context-free graph grammars, well-structured GTSs and the ones that keep the number of nodes constant. *Hyperedge-replacement graph grammars* [9] and *vertex-replacement graph grammars* [10] are well-studied classes of GTSs. It is known that for such graph grammars satisfiability of Monadic Second Order (MSO) formulas is decidable [6]. A logic for expressing properties that involve interleaving of temporal and graph modalities was developed in [4] as a combination of MSO and the μ-calculus. They employ an approximation of GTS [3] that preserves fragments of the logic to obtain a sound but incomplete verification method for these fragments. A method to refine such approximations based on counterexamples was developed in [19]. [23] describes a tool for model checking finite-state graph transition systems against first order temporal logic properties. Automata-theoretic verification has been also applied to objects other than words and trees. In [2], the verification of Message Sequence Charts against temporal prpoerties has been studied. For Mazurkiewicz traces [7] model checking techniques have been developed as well. In order to model concurrency, branching automata on series–parallel graphs have been introduced in [20], focusing on studying the expressivity of these automata, but not their algorithmic properties. The work [21] studies the emptiness problem for concurrent automata with auxiliary storage and provides a generalization of the decidability results for a number of classes of such automata for which the emptiness problem can be reduced to emptiness of finite-state graph automata defined MSO definable graphs with bounded tree width. It might

be possible to obtain or generalize the results we establish in this paper through arguments similar to theirs.

The conference version of this paper appeared as [8].

2 Graph-grammar transition systems

Let A be a finite set. As usual, the set A^* consists of all finite sequences of elements of A. Let $\pi = a_0 a_1 \cdots a_{n-1} a_n \in A^*$. We define $\pi^{-1} = a_n a_{n-1} \cdots a_1 a_0$. The length $|\pi| = n + 1$ of π is the number of elements of π and given $0 \leq i \leq j \leq n$ we denote $\pi[i] = a_i$ and $\pi[i, j] = a_i \cdots a_j$.

With $\mathbb{M}(A) = \{S \mid S : A \to \mathbb{N}\}$ we denote the set of multisets over A. For $S_1, S_2 \in \mathbb{M}(A)$ we define $S_1 \preceq S_2$ iff for every $a \in A$ we have $S_1(a) \leq S_2(a)$. We use square brackets to denote multisets, for example, $[a_1, a_2, a_2]$ denotes $S \in \mathbb{M}(A)$, where $S(a_1) = 1$, $S(a_2) = 2$ and $S(a) = 0$ for all $a \in A \backslash \{a_1, a_2\}$.

2.1 Series–parallel graph grammars

Fix an alphabet Σ. We consider graphs labelled with letters from Σ. A *graph* is a tuple $G = (N, E, n_b, n_e)$ where N is a finite set of nodes, $E \in \mathbb{M}(N \times N \times \Sigma)$ is a multiset of edges and $n_b, n_e \in N$ are two distinguished nodes called source and sink, respectively. For an edge $e = (n, n', \sigma) \in E$, we write $src(e)$ for n and $trg(e)$ for n', and $\alpha(e)$ for the label σ of e. We write $\mathcal{H}_\Sigma$ for the set of all Σ-labelled graphs.

Let $G = (N, E, n_b, n_e)$ and $G' = (N', E', n'_b, n'_e)$ be graphs on disjoint sets of nodes. For an edge $\widehat{e} = (\widehat{n}_1, \widehat{n}_2, \widehat{\sigma}) \in E$, the *edge replacement* graph $G[\widehat{e} \mapsto G']$ is the (unique up to isomorphism) graph defined by removing one copy of the edge $\widehat{e}$ from G, and adding the nodes and edges of G' by fusing $\widehat{n}_1$ with n'_b, and $\widehat{n}_2$ with n'_e. Formally, $G[\widehat{e} \mapsto G'] = (N'', E'', n_b, n_e)$, where $N'' = N \dot{\cup} (N' \backslash \{n'_b, n'_e\})$, $E'' = (E \backslash \{\widehat{e}\}) \cup \widehat{E}'$, where there is an edge (n_1, n_2, σ) in the multiset $\widehat{E}'$ with some multiplicity $k > 0$ iff one of the following conditions holds:

- there is an edge (n'_b, n'_e, σ) in E' with multiplicity k, $n_1 = src(\widehat{e})$ and $n_2 = trg(\widehat{e})$;
- there is an edge (n_1, n_2, σ) in E' with multiplicity k, $n_1 \neq n'_b$ and $n_2 \neq n'_e$;
- there is an edge (n'_b, n_2, σ) in E' with multiplicity k, and $n_1 = src(\widehat{e})$;
- there is an edge (n_1, n'_e, σ) in E' with multiplicity k, and $n_2 = trg(\widehat{e})$.

Definition 1 (*Series–parallel graph grammar*) A *series parallel graph grammar* (SPGG) is a tuple $\mathcal{G} = (V, \Sigma, R, G_0)$, where V is a finite set of *variables*, Σ is a finite alphabet ($\Sigma \cap V = \emptyset$), $R \subseteq V \times \mathcal{H}_{\Sigma \cup V}$ is a finite set of *rules*, $G_0 = (\{n_b, n_e\}, \{(n_b, n_e, v_0)\}, n_b, n_e) \in \mathcal{H}_V$ is the *initial graph*, with *start node* n_b, *sink node* n_e, where $n_b \neq n_e$ and *start symbol* $v_0 \in V$. Furthermore, each rule $(v, G') \in R$, where $G' = (N', E', n'_b, n'_e)$, satisfies exactly one of the following conditions:

(1) $N' = \{n'_b, n'_e\}$, $E' = \{(n'_b, n'_e, \sigma)\}$ and $\sigma \in \Sigma$, denoted $(v, \sigma) \in R$;
(2) $N' = \{n'_b, n'_e, n'\}$ has three nodes, $E' = \{(n'_b, n', v_1), (n', n'_e, v_2)\}$ and $v_1, v_2 \in V$, denoted by $(v, v_1 \cdot v_2) \in R$ *(series composition)*;
(3) $N' = \{n'_b, n'_e\}$ has two nodes, $E' = \{(n'_b, n'_e, v_1), (n'_b, n'_e, v_2)\}$ and $v_1, v_2 \in V$, denoted by $(v, v_1 \parallel v_2) \in R$ *(parallel composition)*.

An SPGG derives a graph in $\mathcal{H}_\Sigma$ as follows. It starts with the graph G_0. In each step, it picks an arbitrary edge e of the current graph G that is labelled with a variable $v \in V$, and

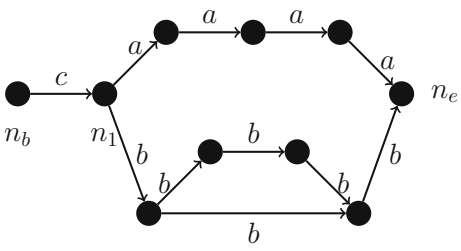

Fig. 1 A series parallel graph generated by the SPGG in Example 1

applies a rule $(v, G') \in R$ to get a new graph $G'' = G[e \mapsto G']$. In this case, we write $G \Longrightarrow G''$. A graph $G \in \mathcal{H}_\Sigma$ is derived if there is a sequence $G_0 \Longrightarrow G_1 \cdots \Longrightarrow G_n = G$ of steps that results in G. Note that every graph thus derived is a series–parallel graph labelled with Σ, so an SPGG represents a set of series–parallel graphs labelled with Σ. We write $\mathcal{L}(\mathcal{G})$ for the set of graphs in $\mathcal{H}_\Sigma$ derived by $\mathcal{G}$. We assume that for every initial variable $v_0 \in V$ it holds that the set of graph derived by $\mathcal{G}$ is non-empty.

Example 1 As an example of an SPGG consider $\mathcal{G} = (V, \Sigma, R, G_0)$ with variables $V = \{v_0, v_1, v_a, v_b, v_c\}$, set of terminal symbols $\Sigma = \{a, b, c\}$, initial graph $G_0 = (\{n_b, n_e\}, \{(n_b, n_e, v_0)\}, n_b, n_e)$ and rules $R = \{(v_0, v_c \cdot v_1), (v_1, v_a \parallel v_b), (v_a, a), (v_b, b), (v_c, c), (v_a, v_a \cdot v_a), (v_b, v_b \cdot v_b), (v_b, v_b \parallel v_b), \}$. Figure 1 shows a (series–parallel) graph G derived from the SPGG $\mathcal{G}$. The directions of the edges denote the direction from the source n_b to sink n_e associated with a series parallel graph. □

A series–parallel graph has a natural "direction" associated with it from the source to the sink, consistent with the direction $n_1 \to n_2$ of an edge (n_1, n_2, σ). In particular, it has no directed cycles. For convenience, we introduce the "symmetric closure" of series–parallel graphs. For each edge (n_1, n_2, σ) labelled with σ, we augment the label with a direction 1 to obtain $(\sigma, 1)$ (1 capturing the "forward" direction), and add an opposite edge $(n_2, n_1, (\sigma, -1))$ labelled with $(\sigma, -1)$ denoting the edge taken in the "backward" direction. Formally, given a series–parallel graph $G = (N, E, n_b, n_e)$, we define its symmetric closure $G' = (N, E', n_b, n_e) \in \mathcal{H}_{\Sigma \times \{1,-1\}}$, where $E' = \{(n, n', (\sigma, 1)) \mid (n, n', \sigma) \in E\} \cup \{(n, n', (\sigma, -1)) \mid (n', n, \sigma) \in E\}$. We write $\mathcal{L}^u(\mathcal{G})$ for the set of symmetric closures of all graphs derived by $\mathcal{G}$.

Remark While for simplicity of the presentation we consider graphs with a single pair of source and sink nodes, our results can in principle be extended to graphs with multiple such nodes. However, the automata construction outlined in Sect. 4.3 relies on the structure of the rules of an SPGG and does not directly generalize to general context-free GTSs defining sets of directed acyclic graphs.

2.2 Graph-grammar transition systems

We now define communicating finite automata on the symmetric closure of series–parallel graphs. Recall that these are series–parallel graphs whose edges are labelled with an alphabet and a direction. Intuitively, a concurrent finite automaton consists of m machines that traverse the edges of a series–parallel graph, some of whose nodes are annotated with Boolean registers. Each automaton traverses the edges of the graph: when the automaton is at a node n of the graph and in state q, it reads the register on the node, chooses an edge with source node n labelled with $(\sigma, d) \in \Sigma \times \{1, -1\}$ based on its current state, the label, and the value

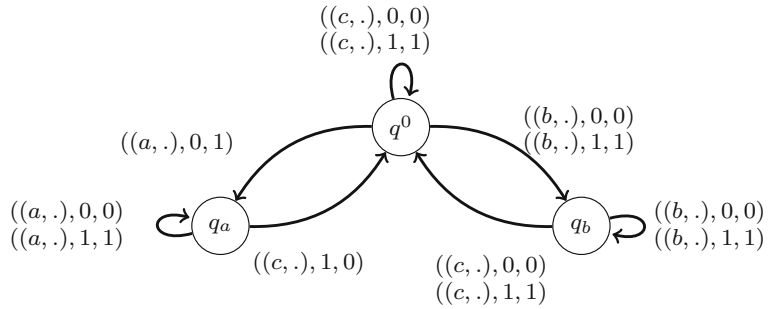

Fig. 2 A finite-state machine with alphabet $\Sigma = \{a, b, c\}$

read from the register, traverses the edge and moves to the target node of that edge and to a new state q', and writes a value to the register at the *source node*.

Let Σ be a finite alphabet. A *finite-state machine* $\mathcal{M} = (Q, q^0, \Sigma, \delta)$ over the alphabet Σ, consists of a finite set of states Q, an initial state $q^0 \in Q$, the input alphabet Σ, and a transition relation $\delta \subseteq Q \times (\Sigma \times \{1, -1\}) \times \mathbb{B} \times Q \times \mathbb{B}$.

The intuitive meaning of a transition $(q, (\sigma, d), b, q', b') \in \delta$ is that when the machine $\mathcal{M}$ is in state q and reads input letter $\sigma \in \Sigma$, direction $d \in \{1, -1\}$, and register value b, then it changes its state to q' and moves along an edge labelled (σ, d) in the graph and writes b' to the register.

Remark For simplicity of the presentation we assume that registers can only hold Boolean values, but our results extend to any fixed finite set of possible values.

Example 2 Figure 2 shows an example of a finite-state machine $\mathcal{M} = (Q, q^0, \Sigma, \delta)$ with states $Q = \{q^0, q_a, q_b\}$, input alphabet $\Sigma = \{a, b, c\}$ and transition relation δ depicted in Fig. 2, where a label $(((\sigma, d), p, p')$ on an edge from state q to state q' stands for the transition $(q, (\sigma, d), p, q', p')$. □

A concurrent finite automaton $(\mathcal{M}, m)$ is a set of m disjoint copies $\mathcal{M}_1, \ldots, \mathcal{M}_m$ of a given finite-state machine $\mathcal{M}$. (The constructions and results in this paper trivially extend to the case when the machines $\mathcal{M}_1, \ldots, \mathcal{M}_m$ are not identical, which we assume here for simplicity of the presentation.)

We now define an infinite-state transition system $T(\mathcal{M}, m, \mathcal{G})$ that captures the behaviour of a concurrent finite-state automaton $(\mathcal{M}, m)$ on the family of series–parallel graphs derived by a given SPGG $\mathcal{G}$.

Definition 2 (*Graph-grammar transition system*) Let $\mathcal{M} = (Q, q^0, \Sigma, \delta)$ be a finite-state machine. A concurrent finite-state automaton $(\mathcal{M}, m)$, together with an SPGG $\mathcal{G} = (V, \Sigma, R, G_0)$ defines a *transition system* $T(\mathcal{M}, m, \mathcal{G}) = (\Gamma, \Gamma_0, \rightarrow)$.

The set of *configurations* Γ consists of all tuples $\langle G, \mu, \beta \rangle$ such that

- $G \in \mathcal{L}^u(\mathcal{G})$ is a graph derived by $\mathcal{G}$;
- $\mu : N \rightarrow 2^{\{1,\ldots,m\} \times Q}$ maps each node in G to the states of the machines at that node; we require that for each $i \in \{1, \ldots, m\}$ there exists exactly one $n \in N$ and exactly one $q \in Q$ with $(i, q) \in \mu(n)$;
- $\beta : N \rightarrow \mathbb{B}$ maps each node to the value of the Boolean register at that node.

The graph G is a graph generated by the SPGG $\mathcal{G}$. Intuitively, the function μ defines the current states of each of the m machines, as well as their positions in the graph G. The function β maps each register to its current value.

The set Γ_0 of *initial configurations* is such that $\gamma = \langle G, \mu, \beta\rangle \in \Gamma_0$ iff $\gamma \in \Gamma$, $\mu(n_b) = \{(i, q^0) \mid i \in \{1, \ldots, m\}\}$, $\mu(n) = \emptyset$ for every $n \in N\backslash\{n_b\}$, and $\beta(n) = 0$ for every $n \in N$. That is, initially all machines are positioned at the source node of the graph and are in their initial state, and all registers are 0.

The *successor relation* $\rightarrow \subseteq \Gamma \times \Gamma$ is defined as the union $\rightarrow = \bigcup_{i=1}^{m} \rightarrow_i$ of the local transition relations $\rightarrow_i$ for $i \in \{1, \ldots, m\}$. We have $(\langle G, \mu, \beta\rangle, \langle G', \mu', \beta'\rangle) \in \rightarrow_i$, (denoted $\langle G, \mu, \beta\rangle \rightarrow_i \langle G', \mu', \beta'\rangle$) iff the following conditions are satisfied:

- $G' = G$, where $G = (N, E, n_b, n_e) \in \mathcal{L}^u(\mathcal{G})$ is a graph generated by $\mathcal{G}$.
- There exist an edge $e = (n, n', (\sigma, d)) \in E$, states $q, q' \in Q$, and $b' \in \mathbb{B}$ such that:
  - (i) $(i, q) \in \mu(n)$ (Note that n and n' are different, since $\mathcal{G}$ is an SPGG);
  - (ii) $(q, \alpha(e), \beta(n), q', b') \in \delta$ is a transition of $\mathcal{M}$;
  - (iii) $\mu'(n) = \mu(n)\backslash\{(i, q)\}$, $\mu'(n') = \mu(n') \cup \{(i, q')\}$ and $\mu'(n'') = \mu(n'')$ for all other nodes $n'' \in N\backslash\{n, n'\}$,
  - (iv) $\beta'(n) = b'$ and $\beta'(n'') = \beta(n'')$ for all other nodes $n'' \in N\backslash\{n\}$.

  We say that the edge e is *compatible* with the transition $\gamma \rightarrow \gamma'$.

The idea is, that at each step a transition $\gamma \rightarrow \gamma'$ of the transition system occurs, at which exactly one machine makes a transition according to δ and also moves along a compatible edge of the graph, updating the functions μ and β accordingly.

A *run* ρ of the transition system $T(\mathcal{M}, m, \mathcal{G}) = (\Gamma, \Gamma_0, \rightarrow)$ is a sequence of configurations $\rho = \gamma_0 \cdots \gamma_f \in \Gamma^*$ where $\gamma_0 \in \Gamma_0$ and $\gamma_{i-1} \rightarrow \gamma_i$ for each $i = 1, \ldots, f$.

A *path* in a graph $G = (N, E, n_b, n_e)$ is a sequence of nodes $n_0 \cdots n_l$ such that for each $0 \leq i < l$ it holds that $(n_i, n_{i+1}, (\sigma, d)) \in E$ for some σ and d.

Let $\rho = \gamma_0 \cdots \gamma_f$ be a run and G be the corresponding underlying graph. A path $n_0, \ldots, n_l$ in G is *compatible* with ρ if and only if there exists a machine $j \in \{1, \ldots, m\}$ such that $\widehat{\gamma_0} \rightarrow \widehat{\gamma_1}, \ldots, \widehat{\gamma_{l-1}} \rightarrow \widehat{\gamma_l}$ is the subsequence of transitions in the run ρ that consists exactly of the transitions made by machine j and it holds that for each $i = 0, \ldots, l-1$ there exists an edge $(n_i, n_{i+1}, (\sigma, d))$ in the graph G that is compatible with the transition $\gamma_i \rightarrow \widehat{\gamma_{i+1}}$. Intuitively, $n_0 \cdots n_l$ is the sequence of nodes in the graph G that is traversed by machine j in the run ρ.

2.3 Reachability in graph-grammar transition systems

We consider the *reachability problem* for graph-grammar transition systems, that is, given a graph-grammar transition system $T(\mathcal{M}, m, \mathcal{G}) = (\Gamma, \Gamma_0, \rightarrow)$ and a set of configurations F, determine whether there is a run of $T(\mathcal{M}, m, \mathcal{G})$ which contains a configuration in F. More specifically, we are interested in sets F defined as follows.

Let $S_{final} \in \mathbb{M}(Q)$ be a multiset of states in Q. Then, S_{final} defines a set of *final configurations* $F \subseteq \Gamma$, where for a configuration $\gamma = \langle G, \mu, \beta\rangle \in \Gamma$ with $G = (N, E, n_b, n_e)$ we have that $\gamma \in F$ iff there exists a node $n \in N$ such that $S_{final} \preceq [q \in Q \mid (i, q) \in \mu(n)$, where $i \in \{1, \ldots, m\}]$. Intuitively, in the graph of a final configuration there exists a node n such that the multiset S_{final} is contained in the multiset consisting of the states of the machines located at the node n.

Example 3 Consider the graph-grammar transition system $T(\mathcal{M}, 2, \mathcal{G})$ where $\mathcal{G}$ is the SPGG from Example 1 and $\mathcal{M}$ is the machine described in Example 2. The multiset $[q_a, q_a]$ defines

a set of final configurations in which there exists a node at which both machines are currently located while both are in state q_a. □

Let $(\mathcal{M}, m)$ be a concurrent automaton and $\mathcal{G}$ an SPGG. Given a multiset S_{final} that defines a set F of final configurations, the *reachability problem* for $(\mathcal{M}, m)$, $\mathcal{G}$ and F, $\mathsf{Reach}(\mathcal{M}, m, \mathcal{G}, F)$, is to decide whether there exists a run $\rho = \gamma_0 \cdots \gamma_f$ of $T(\mathcal{M}, m, \mathcal{G})$ such that $\gamma_i \in F$ for some $0 \leq i \leq f$, i.e., a run that reaches F.

Since our model allows machines to do arbitrarily many "reversals" (i.e., following forward and backward edges) and do not fix a bound on the number of shared registers that are read or written, it easily captures linear bounded automata. Thus, the reachability problem is in general undecidable.

Proposition 1 *The reachability problem is undecidable.*

Proof The proof is by reduction of the emptiness problem for linear bounded automata to the reachability problem for graph-grammar transition systems. A *linear bounded automaton* is a tuple $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, \Gamma_\mathcal{A}, \rho_\mathcal{A}, q^0_\mathcal{A}, \vdash, \dashv, Q^F_\mathcal{A})$, where $Q_\mathcal{A}$ is a finite set of states, $\Sigma_\mathcal{A}$ is a finite input alphabet, $\Gamma_\mathcal{A} \supseteq \Sigma_\mathcal{A}$ is a finite tape alphabet, $\rho_\mathcal{A} \subseteq Q_\mathcal{A} \times \Gamma_\mathcal{A} \times Q_\mathcal{A} \times \Gamma_\mathcal{A} \times \{1, -1\}$ is a transition relation, $\vdash \in \Sigma$ and $\dashv \in \Sigma$ are the left and right endmarkers, and $Q^F_\mathcal{A} \subseteq Q_\mathcal{A}$ is the set of accepting states.

Intuitively, the semantics of a transition $(q, \sigma, q', \sigma', d) \in \rho_\mathcal{A}$ is as follows: If the current state of the automaton is q and the letter at the current position of the head is σ, then the automaton can go to state q' while writing at the current cell of the tape the symbol σ' and moving the head one position to the left, if $d = -1$ or one position to the right if $d = 1$. Additionally, the transition relation of a linear bounded automaton must satisfy the following two conditions:

(1) the head cannot be moved left of the left endmarker $\vdash$ or right of the right endmarker $\dashv$, that is, if $\sigma = \vdash$, then $d = 1$ and if $\sigma = \dashv$, then $d = -1$;
(2) the endmarker symbols cannot be overwritten: if $\sigma \in \{\vdash, \dashv\}$, then $\sigma' = \sigma$.

A word $w \in (\Sigma \backslash \{\vdash, \dashv\})^*$ is accepted by $\mathcal{A}$ if there exists a run of $\mathcal{A}$ on $\vdash w \dashv$ that reaches a state in $Q^F_\mathcal{A}$. The precise definition of linear bounded automata can be found in standard textbooks on automata theory, e.g., [16]. It is well-known that the emptiness problem for linear bounded automata in undecidable.

We now show how to encode the emptiness problem for linear bounded automata as reachability in graph-grammar transition systems. Without loss of generality, we assume a single finite state in the automaton. More precisely, we show how given a linear bounded automaton $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, \Gamma_\mathcal{A}, \rho_\mathcal{A}, q^0_\mathcal{A}, \vdash, \dashv, Q^F_\mathcal{A})$ we can construct an SPGG $\mathcal{G}$ and a concurrent finite automaton $(\mathcal{M}, 1)$ and a set of final configurations F such that there exists a word accepted by $\mathcal{A}$ iff there exists a run of $T(\mathcal{M}, 1, \mathcal{G})$ that reaches a configuration in F. Intuitively, the SPGG $\mathcal{G}$ will generate graphs that correspond to words in $(\Sigma \backslash \{\vdash, \dashv\})^*$, and the single machine $\mathcal{M}$ will simulate the automaton $\mathcal{A}$ on the generated word. The proof relies on two points: we assume that each node in the graph is equipped with a register, thus being able to encode the tape of the linear bounded automaton, and that the machine $\mathcal{M}$ is allowed to reverse its direction an unbounded number of times.

Let $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, \Gamma_\mathcal{A}, \rho_\mathcal{A}, q^0_\mathcal{A}, \vdash, \dashv, Q^F_\mathcal{A})$ be a linear bounded automaton.

We define the SPGG $\mathcal{G} = (V, \Sigma, R, G_0)$ such that $V = \{v_0, v_1, v, v_\vdash, v_\dashv\}$, $\Sigma = \Sigma_\mathcal{A}$, $R = \{(v_0, v_\vdash \cdot v_1), (v_1, v \cdot v_\dashv), (v, v \cdot v), (v_\vdash, \vdash), (v_\dashv, \dashv), \} \cup \{(v, \sigma) \mid \sigma \in \Sigma \backslash \{\vdash, \dashv\}\}$ and $G_0 = (\{n_b, n_e\}, \{(n_b, n_e, v_0)\}, n_b, n_e)$. We suppose that each node in a graph derived by $\mathcal{G}$ is equipped with a register large enough to store a value in $\{0\} \dot{\cup} \Gamma$.

The machine $\mathcal{M} = (Q, q^0, \Sigma, \delta)$ simulates $\mathcal{A}$ on the generated graph. Since in $T(\mathcal{M}, 1, \mathcal{G})$ all registers are initialized with 0, $\mathcal{M}$ first makes a pass through the graph copying in each register the letter on the outgoing edge with direction 1. Then, it simulates $\mathcal{A}$ reading from and writing to the registers that represent $\mathcal{A}$'s tape. We define $Q = Q_{\mathcal{A}} \cup \{(q^0_{\mathcal{A}}, 1), (q^0_{\mathcal{A}}, -1)\}$, $q^0 = (q^0_{\mathcal{A}}, 1)$ and $(q, (\sigma, d), b, q', b') \in \delta$ iff one of the following conditions is satisfied:

- $q = (q^0_{\mathcal{A}}, 1)$, $\sigma \neq \dashv$, $d = 1$, $q' = q$, and $b' = \sigma$;
- $q = (q^0_{\mathcal{A}}, 1)$, $\sigma = \dashv$, $d = 1$, $q' = (q^0_{\mathcal{A}}, -1)$, and $b' = \sigma$;
- $q = (q^0_{\mathcal{A}}, -1)$, $\sigma \neq \vdash$, $d = -1$, $q' = q$, and $b' = b$;
- $q = (q^0_{\mathcal{A}}, -1)$, $\sigma = \vdash$, $d = -1$, $q' = q^0_{\mathcal{A}}$, and $b' = b$;
- $q \in Q_{\mathcal{A}}$ and $(q, b, q', b', d) \in \rho_{\mathcal{A}}$.

Let us define the set of final configurations F such that for $\gamma = \langle G, \mu, \beta \rangle$ with $G = (N, E, n_b, n_e)$ we have that $\gamma \in F$ iff there exists a node $n \in N$ such that $(1, q^F_{\mathcal{A}}) \in \mu(n)$, where $q^F_{\mathcal{A}}$ is the single accepting state of $\mathcal{A}$. The definitions of $\mathcal{G}$, $\mathcal{M}$ and F imply that the language of $\mathcal{A}$ is nonempty iff there exists a run $\rho = \gamma_0 \cdots \gamma_f$ of $T(\mathcal{M}, 1, \mathcal{G})$ such that $\gamma_i \in F$ for some $0 \leq i \leq f$. □

Note that proof of the above undecidability result holds already for the case of *single machine* and *linear graphs*.

2.4 Reversal- and register-bounded reachability problem

Since the general problem is undecidable, we focus on a bounded version. We introduce two restrictions. First, we allow each machine to make only a bounded number of reversals (a reversal occurs when the machine changes direction in the graph). Second, we fix an a priori bound on the number of shared registers. That is, while the SPGG generates a potentially unbounded set of graphs, with unboundedly many nodes, we assume that there is some fixed bound k on the number of Boolean registers located at nodes of a generated graph (these k registers may be situated at arbitrary nodes of the graph though).

In what follows we formalize the above definitions. Let us fix a machine $\mathcal{M} = (Q, q^0, \Sigma, \delta)$, the concurrent automaton $(\mathcal{M}, m)$, and an SPGG $\mathcal{G} = (V, \Sigma, R, G_0)$.

Reversal bound Let us fix a run $\rho = \gamma_0, \ldots, \gamma_f$ where $\gamma_i = \langle G, \mu_i, \beta_i \rangle$. Consider the projection of ρ to $\rightarrow_j$ for each machine $j \in \{1, \ldots, m\}$. The number of *reversals* made by machine j along the run, intuitively, is the number of times it changes from traversing an edge marked with direction 1 to traversing an edge marked with direction -1, or vice versa.

Formally, let $e_1 e_2 \cdots e_n$ be a sequence of edges. A reversal occurs at position i if $\alpha(e_i) = (\cdot, 1)$ and $\alpha(e_{i+1}) = (\cdot, -1)$ or if $\alpha(e_i) = (\cdot, -1)$ and $\alpha(e_{i+1}) = (\cdot, 1)$.

Now, let $\gamma_{i_1} \rightarrow_j \gamma_{i_1+1}, \gamma_{i_2} \rightarrow_j \gamma_{i_2+1}, \ldots$ be the transitions of machine j along the run ρ, and let $e_{i_1}, e_{i_2}, \ldots$ be the compatible edges that were taken by machine j. The number of reversals of machine j along ρ is the number of reversals in the sequence $e_{i_1} e_{i_2} \cdots$.

For $r \geq 0$, the set of *r-reversal bounded* runs of $T(\mathcal{M}, m, \mathcal{G})$ is the set of runs in which each machine makes at most r reversals.

Register bound The register bound fixes a number k of Boolean registers. That is, each graph G derived by $\mathcal{G}$ comes with a mapping $\kappa : N \rightarrow \{0, 1\}$, such that $|\kappa^{-1}(1)| \leq k$, and we allow the machines to read and write register values only when their current node is in $\kappa^{-1}(1)$.

To derive graphs with a mapping κ, we modify an SPGG to "mark" some nodes along the derivation, and ensure that any derived graph has at most k marked nodes. (The formal details are similar to constructing a CFG for a CFL with at most k marked positions from a

CFG for the (unmarked) language.) For an SPGG $\mathcal{G}$, we denote by $\mathcal{G}^k$ the SPGG that marks at most k nodes of a derived graph. We write, by abuse of notation, $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ for a graph G which is the symmetric closure of a graph derived by $\mathcal{G}^k$ together with the mapping κ.

In addition, we modify the successor relation of the graph-grammar transition systems $T(\mathcal{M}, m, \mathcal{G}^k)$ to require (ii)' $(q, \alpha(e), \beta(n), q', b') \in \delta$ if $\kappa(n) = 1$ and $(q, \alpha(e), 0, q', 0) \in \delta$ otherwise.

Example 4 The SPGG $\mathcal{G}$ shown in Example 1 can be modified into an SPGG $\mathcal{G}^2$ that derives graphs in which at most 2 nodes are marked to contain registers. Furthermore, we can consider SPGGs that not only ensure an upper bound on the number of marked nodes, but impose constraints on their location. For example, we can consider an SPGG $\mathcal{G}^2_*$ that additionally requires that by applying the rule $(v_0, v_c \cdot v_1)$ the node between the edges labelled v_c and v_1 contains a register.

If we then consider the graph-grammar transition system $T(\mathcal{M}, 2, \mathcal{G}^2_*)$, where $\mathcal{M}$ is the finite-state machine described in Example 2, and let G be the graph depicted in Fig. 1, then there does not exist a run with underlying graph G that reaches a configuration in the set of final configurations defined by the multiset $[q_a, q_a]$, since the register at node n_1 acts as a semaphore that does not allow two copies of the machine $\mathcal{M}$ to enter the part of the graph containing edges labelled with the letter a. □

The reversal- and register-bounded reachability problem takes as input a concurrent automaton $(\mathcal{M}, m)$, an SPGG $\mathcal{G}$, and parameters r and k, and a set of final configurations F defined by a multiset S_{final}, and asks if there exists an r-reversal bounded run of $(\mathcal{M}, m)$ on some graph derived by $\mathcal{G}^k$ that reaches F.

Our main result is the following.

Theorem 1 *The reversal- and register-bounded reachability problem is decidable.*

In the following sections we prove decidability by reducing the reversal- and register-bounded reachability problem to the emptiness problem for pushdown automata.

Remark Our decidability result holds for a somewhat more general model, in which each machine can also read any of the fixed number of registers that are not at its current node, but can only write to the register at its current node, or vice versa. We work in the simpler setting to keep the notation manageable.

3 Properties of reversal-bounded runs

Fix a machine $\mathcal{M} = (Q, q^0, \Sigma, \delta)$, the concurrent automaton $(\mathcal{M}, m)$, an SPGG $\mathcal{G} = (V, \Sigma, R, G_0)$ and the parameters r and k. In this section we state two properties of r-reversal bounded runs of $T(\mathcal{M}, m, \mathcal{G}^k)$ that allow us to encode such runs as words over a finite alphabet and to reduce the reversal- and register-bounded reachability problem to the emptiness test for a context free language.

Given a run $\rho = \gamma_0 \cdots \gamma_f$ and a machine $i \in \{1, \ldots, m\}$, an *i-block* is a segment $\rho[j_1, j_2] = \gamma_{j_1} \cdots \gamma_{j_2}$ of the run ρ such that $\gamma_j \rightarrow_i \gamma_{j+1}$ for each $j_1 \leq j < j_2$. That is, all transitions in the part $\rho[j_1, j_2]$ of the run are made by machine i. Proposition 2 below establishes that for every r-reversal bounded run ρ we can reorder its transitions to obtain an r-reversal bounded run $\widehat{\rho}$ such that the number of maximal blocks in $\widehat{\rho}$ is bounded from

above by a constant depending on the parameters m, r and k (and not on the length of the run ρ).

The run $\widehat{\rho}$ is constructed by reordering transitions in ρ while keeping in place the transitions that access registers, i.e., we swap transitions that do not access registers. This is formalized in the following lemma.

First, let us define the notion of independent transitions. We call a transition $\gamma \to \gamma'$ in a run ρ a *read/write transition* if $\kappa(n) = 1$, where n and n' are the nodes in G associated with this transition, that is, the machine making the transition from node n to node n' can read from and write to the register at node n. We say that two consecutive transitions $\gamma_0 \to_{m_1} \gamma_1$ and $\gamma_1 \to_{m_2} \gamma_2$ in a run ρ are *independent* if $m_1 \neq m_2$ and none of the transitions is a read or write transition.

Lemma 1 *Let $\gamma_0 \to_{m_1} \gamma_1$ and $\gamma_1 \to_{m_2} \gamma_2$ be independent transitions in a run ρ, where $\gamma_0 = \langle G, \mu_0, \beta_0\rangle$, $\gamma_1 = \langle G, \mu_1, \beta_1\rangle$, $\gamma_2 = \langle G, \mu_2, \beta_2\rangle$. Let $n_1, n_1' \in N$ and $q_1, q_1' \in Q$ correspond to $\gamma_0 \to_{m_1} \gamma_1$ and $n_2, n_2' \in N$ and $q_2, q_2' \in Q$ correspond to $\gamma_1 \to_{m_2} \gamma_2$. Then, there exist configurations $\widehat{\gamma}_1 = \langle G, \widehat{\mu}_1, \widehat{\beta}_1\rangle$ and $\widehat{\gamma}_2 = \langle G, \widehat{\mu}_2, \widehat{\beta}_2\rangle$ such that:*

- *$\gamma_0 \to_{m_2} \widehat{\gamma}_1$, $\widehat{\gamma}_1 \to_{m_1} \widehat{\gamma}_2$, and $\widehat{\mu}_2 = \mu_2$, and*
- *none of the transitions reads or writes a register value and $\beta_0 = \widehat{\beta}_1 = \widehat{\beta}_2 = \beta_2$.*

Proof Suppose that $e_1 \in E$, $b_1 \in \mathbb{B}$ and $b_1' \in \mathbb{B}$ satisfy the conditions of Definition 2 for transition $\gamma_0 \to \gamma_1$ and that $e_2 \in E$, $b_2 \in \mathbb{B}$ and $b_2' \in \mathbb{B}$ satisfy the conditions of Definition 2 for transition $\gamma_1 \to \gamma_2$. By assumption we have $b_1 = b_1' = b_2 = b_2' = 0$.

Let for each $n \in N$ and $j \in \{1, \ldots, k\}$,

$$\widehat{\mu}_1(n) = \begin{cases} \mu_0(n) \backslash \{(m_2, q_2)\} & \text{if } n = n_2, \\ \mu_0(n) \cup \{(m_2, q_2')\} & \text{if } n = n_2', \\ \mu_0(n) & \text{otherwise;} \end{cases}$$

$$\widehat{\mu}_2(n) = \begin{cases} \widehat{\mu}_1(n) \backslash \{(m_1, q_1)\} & \text{if } n = n_1, \\ \widehat{\mu}_1(n) \cup \{(m_1, q_1')\} & \text{if } n = n_1', \\ \widehat{\mu}_1(n) & \text{otherwise.} \end{cases}$$

We define $\widehat{\beta}_1 = \widehat{\beta}_2 = \beta_0$.

These definitions clearly fulfill the conditions required by the lemma. □

The above lemma allows us transform runs by iterative reordering of independent transitions. We define two runs $\rho = \gamma_0, \ldots, \gamma_f$ and $\widehat{\rho} = \widehat{\gamma}_0, \ldots, \widehat{\gamma}_{\widehat{f}}$ to be *equivalent* if $\widehat{f} = f$, $\widehat{\gamma}_0 = \gamma_0$, $\widehat{\gamma}_{\widehat{f}} = \gamma_f$ and $\widehat{\rho}$ can be obtained from ρ by reordering of independent transitions (as in the proof of Lemma 1).

Proposition 2 *For every r-reversal bounded run $\rho = \gamma_0, \ldots, \gamma_f$ of $T(\mathcal{M}, m, \mathcal{G}^k)$ there exist an r-reversal bounded run $\widehat{\rho} = \widehat{\gamma}_0, \ldots, \widehat{\gamma}_f$ of $T(\mathcal{M}, m, \mathcal{G}^k)$ equivalent to ρ, and a sequence of indices $0 = f_0 < f_1 < \cdots < f_u = f$ such that:*

- *for each $i \leq u - 1$, there exists $m_i \in \{1, \ldots, m\}$ such that $\widehat{\rho}[f_i, f_{i+1}]$ is an m_i-block,*
- *$u \leq \big(r \cdot m + k \cdot m \cdot (r + 1) + 1\big) \cdot (m + 1)$.*

Proof Let I' be the set of indices of (source configurations of) reversal transitions. Since ρ is an r-reversal bounded run we have $|I'| \leq r \cdot m$.

Since ρ is a run in $T(\mathcal{M}, m, \mathcal{G}^k)$, there exists a mapping κ that maps the nodes of underlying graph for ρ to Boolean values indicating which nodes are equipped with registers. The number of these nodes, that is, the number of nodes in $\kappa^{-1}(1)$, is at most k.

Let $I'' = \{i \mid 0 \leq i < f, \gamma_i \to \gamma_{i+1}$ is a read/write transition$\}$. Since each machine can do at most k read/write transitions in a run segment where it performs no reversals, we have that $|I''| \leq m \cdot k \cdot (r+1)$.

Let $I' \cup I'' = \{i_1, \ldots, i_l\}$. We split ρ into segments $\rho_0, \ldots, \rho_l$, where $\rho_0 = \rho[1, i_1]$, $\rho_j = \rho[i_j + 1, i_{j+1}]$ for each $1 \leq j < l$ and $\rho_l = \rho[i_l + 1, f]$. Let $i \in \{0, \ldots, l\}$. The segment ρ_i does not contain reversal transitions or read/write transitions.

Applying Lemma 1 we reorder the transitions in ρ_i to obtain a sequence of run segments $\widehat{\rho_0}, \ldots, \widehat{\rho_l}$ such that each $\widehat{\rho_i}$ contains at most m blocks, has no read/write transitions and no reversals and the order of transitions of each machine in preserved. Furthermore, the first and last configurations of ρ_i and $\widehat{\rho_i}$ are the same. Thus, we can combine the segments $\widehat{\rho_0}, \ldots, \widehat{\rho_l}$ into a run $\widehat{\rho}$. We are guaranteed that the run $\widehat{\rho}$ is r-reversal bounded and has the required properties. In particular, since $|I' \cup I''| \leq r \cdot m + k \cdot m \cdot (r+1)$, for the number u of maximal blocks in the run $\widehat{\rho}$ it holds that $u \leq \big(r \cdot m + k \cdot m \cdot (r+1) + 1\big) \cdot (m+1)$. □

The second property uses the bound r on the number of reversals of each machine in an r-reversal bounded run ρ to relate ρ to the set of paths in the underlying graph traversed by the machines in ρ.

A *trace* τ is an element of the set Σ^*. A trace $\tau = \sigma_1 \cdots \sigma_f$ is *compatible* with a run $\rho = \gamma_0, \ldots, \gamma_f$ if there exists a sequence of edges $e_1 e_2 \cdots e_f$ compatible with ρ such that $\alpha(e_i) = (\sigma_i, \cdot)$ for every $0 < i \leq f$. That is, the trace τ consists of the labels of the edges traversed by the machines in the run ρ.

Given a graph $G = (N, E, n_b, n_e) \in \mathcal{L}^u(\mathcal{G}^k)$ and a trace τ we define $\mathsf{Paths}(G, \tau)$ to be the (possibly empty) set of paths from n_b to n_e whose sequence of edge labels is $\tau = \sigma_1 \cdots \sigma_f$. Formally, for a sequence of nodes $\pi = n_0 n_1 \cdots n_f \in N^*$ we have $\pi \in \mathsf{Paths}(G, \tau)$ iff $n_0 = n_b$, $n_f = n_e$ and $(n_{i-1}, n_i, (\sigma_i, 1)) \in E$.

Below we establish a property of an r-reversal bounded run $\rho = \gamma_0 \cdots \gamma_f$ of $T(\mathcal{M}, m, \mathcal{G}^k)$ and a trace τ that is compatible with ρ. Namely, for each machine $i \in \{1, \ldots, m\}$ the corresponding (not necessarily contiguous) subsequence τ_i of τ can be split into at most $r+1$ segments, such that each of those segments can be embedded in a trace labelling a simple path from n_b to n_e or from n_e to n_b.

This property is formalized in the following proposition, and easily follows from the definitions of series–parallel graphs and graph-grammar transition systems.

Proposition 3 *Let ρ be an r-reversal bounded run of $T(\mathcal{M}, m, \mathcal{G}^k)$ and τ be a trace that is compatible with ρ, and let $i \in \{1, \ldots, m\}$. Let π_i be the sequence of nodes visited in ρ by machine i, in the order they occur in ρ, let τ_i be the corresponding subsequence of τ, and $r_i \leq r$ be the number of reversals of machine i in ρ.*

Then, for each $h \in \{1, \ldots, r+1\}$ there exist traces $\tau_h, \tau'_h, \tau''_h, \tau'''_h \in \Sigma^*$ and sequences of nodes $\pi_h, \pi'_h, \pi''_h, \pi'''_h \in N^*$ such that the following conditions are satisfied:

- $\pi_h \in \mathsf{Paths}(G, \tau_h)$, *and* $\tau_h = \tau'_h \cdot \tau''_h \cdot \tau'''_h$, *and* $\pi_h = \pi'_h \cdot \pi''_h \cdot \pi'''_h$;
- *There exists a sequence of indices $0 = j_0 < j_1 < \cdots < j_{r_i+1} = |\pi_i| - 1$ such that for every h with $1 \leq h \leq r_i + 1$ it holds that:*
  - *if h is odd, then $\tau_i[j_{h-1} + 1, j_h] = \tau''_h$ and $\pi_i[j_{h-1}, j_h] = \pi''_h$;*
  - *if h is even, then $\tau_i[j_{h-1} + 1, j_h] = \tau''^{-1}_h$ and $\pi_i[j_{h-1}, j_h] = \pi''^{-1}_h$.*

Proposition 2 allows us to restrict our reasoning to r-reversal bounded runs with at most $\big(r \cdot m + k \cdot m \cdot (r+1) + 1\big) \cdot (m+1)$ blocks. Proposition 3 allows us to reduce from reasoning

about graphs derived by $\mathcal{G}^k$ to reasoning about $m \cdot (r+1)$-tuples of traces in such graphs. Based on these results, we define the two parameters $p = \big(r \cdot m + k \cdot m \cdot (r+1) + 1\big) \cdot (m+1)$ and $t = \widetilde{r} \cdot m$, where $\widetilde{r} = r + 1$.

3.1 Discussion of the imposed bounds

In the definition of the bounded reachability problem we bounded simultaneously two of the parameters: the number of reversals of each machine and the number of registers in the generated graphs. As we saw in this section, the conjunction of these two restrictions implies a bound of the number of *context switches* (the number of times a run switches from executing one of the machines to another) in runs that have to be considered. Bounding the number of context switches is a standard way to regain decidability of otherwise undecidable verification problems for concurrent systems. It is thus not surprising that by imposing this restriction we can decouple the executions of the individual machines by ensuring that their individual executions match accordingly at the bounded number of points of context switching. This is the idea behind Proposition 2 and the automata constructions in Sects. 4.4 and 4.5.

However, bounding the number of reversals of each machine plays a further important role in our setting, in that it implies a bound on the number of single-direction paths in a graph generated by $\mathcal{G}$ that are traversed by the machines in a run: given m machines each of which can only reverse its direction at most r times, we have an upper bound of $m \cdot (r+1)$ on the number of such paths that are involved in a run. This property, formalized in Proposition 3, is crucial for the reduction of the bounded verification problem to an emptiness question for *word automata*. It allows us to construct from an SPGG a pushdown automaton operating on words that describe tuples of paths in a graph generated by the SPGG.

Since the reduction to emptiness of word automata is at the core of our decidability proof, it is not clear how to extend our decidability result to the case of *unbounded number of reversals* (even when bounding the number of registers).

Our proof of undecidability of the general case requires an unbounded number of registers and holds already for a single machine and linear graphs. For the case of a single machine, linear graphs and bounded number of registers, one can use arguments similar to those in the conversion of 2NFA to NFA to show decidability. For the general case, however, there are two main difficulties: the unbounded number of paths and the interaction between the machines. Unlike multihead automata, the exchange of information between machines in our setting is related to their current position in the graph, and thus, the corresponding undecidability results for the emptiness problem for multihead automata do not easily transfer to our case. (Were the communicating machines allowed to read and write the registers remotely, multihead automata could be encoded in our framework by using the registers to model shared state.) The way communication is restricted in our computational model appears to require a new kind of argument to establish decidability or undecibility when only the number of registers is bounded.

The other intermediate case is when we have a *bounded number of reversals* (and *unbounded number of registers*). The bound on the number of reversals is sufficient to show Proposition 3 and reduce the verification problem to reasoning about tuples of traces in graphs derived by the given grammar. However, this restriction alone does not imply a bound on the number of context switches: indeed, it is possible that this number is unbounded, meaning that there are runs of arbitrary length reaching a final configuration and in each of them the number of context switches depends on the length of the run. However, this property does imply that each node in a graph generated by an SPGG is visited at most $m \cdot (r+1)$ times and since read and write access to registers is local, each register is read or written also at most

$m \cdot (r+1)$ times. Thus, it is possible that using this *finite-visit* property runs can be encoded as words of visiting sequences, where a visiting sequence for a node in a graph captures the consecutive visits of the machines to that node. However, it is not easy to see if such a construction can be extended to the case when only one type of register accesses (read or write) is required to be local. Our decidability result for bounded number of registers, on the other hand, can be extended to the case when only one type of register access is required to be local, since one type of restriction suffices to ensure bounded number of context switches.

4 Automata-theoretic algorithm

In this section we present an automata-theoretic algorithm for solving the reversal- and register-bounded reachability problem. Before we give an overview of our algorithm and describe the automata constructions it comprises, we recall some basic definitions from automata theory.

4.1 Preliminaries

A 2-way nondeterministic finite automaton (2NFA) is a tuple $\mathcal{A} = (Q, Q^0, \Sigma, \delta, A)$, where Q is a finite set of states, $Q^0 \subseteq Q$ is a set of initial states, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q \times \{-1, 1\}$ is the transition relation and $A \subseteq Q$ is a set of accepting states.

$\mathcal{A}$ is deterministic iff δ is a function from $Q \times \Sigma$ to $Q \times \{-1, 1\}$. $\mathcal{A}$ is a 1-way NFA (NFA) iff $d = 1$ for each $(q, \sigma, q', d) \in \delta$.

For $q, q' \in Q$, $w', w'', w''', w'''' \in \Sigma^*$, $\sigma \in \Sigma$ and $\sigma' \in \Sigma \cup \{\epsilon\}$, let $\langle q, w', \sigma, w'' \rangle \Rightarrow_{\mathcal{A}} \langle q', w''', \sigma', w'''' \rangle$ iff $(q, \sigma, q', d) \in \delta$ and the following two conditions hold:

(1) if $d = 1$, then $w''' = w'.\sigma$, $w'' = \sigma'.w''''$, either $\sigma' \in \Sigma$ or $\sigma' = \epsilon$ and $w'''' = \epsilon$;
(2) if $d = -1$, then $w'''' = \sigma.w''$, $w' = w'''.\sigma'$, either $\sigma' \in \Sigma$ or $\sigma' = \epsilon$ and $w''' = \epsilon$.

If $\mathcal{A}$ is an NFA, we define $\delta(q, w)$ for $w \in \Sigma^*$ in the obvious way.

Let $\vdash \in \Sigma$ and $\dashv \in \Sigma$, where $\vdash \neq \dashv$, be designated symbols and $w \in (\Sigma \setminus \{\vdash, \dashv\})^*$. We require that for every $(q, \sigma, q, d) \in \delta$ it holds that if $\sigma = \vdash$, then $d = 1$ and if $\sigma = \dashv$, then $d = -1$.

If $\mathcal{A}$ is a 2NFA, then $w \in \mathcal{L}(\mathcal{A})$ iff for some $q^0 \in Q^0$ and $q \in A$ one of the following holds:

- $\langle q^0, \epsilon, \vdash, w \dashv \rangle \Rightarrow^*_{\mathcal{A}} \langle q, \vdash w \dashv, \epsilon, \epsilon \rangle$, or
- $\langle q^0, \epsilon, \vdash, w \dashv \rangle \Rightarrow^*_{\mathcal{A}} \langle q, \epsilon, \epsilon, \vdash w \dashv \rangle$.

If $\mathcal{A}$ is an NFA, then $w \in \mathcal{L}(\mathcal{A})$ iff $\delta(q^0, \vdash w \dashv) \cap A \neq \emptyset$ for some $q^0 \in Q^0$.

A push-down automaton (PDA) is a tuple $\mathcal{P} = (Q, q^0, \Sigma, \Delta, \perp, \delta)$, where Q is a finite set of states, $q^0 \in Q$ is the initial state, Σ is a finite input alphabet, Δ is a finite stack alphabet, $\perp \in \Delta$ is the start stack symbol and $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Delta \times Q \times \Delta^*$ is the transition relation. For $q, q' \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, $w \in \Sigma^*$, $a \in \Delta$, $\alpha, \beta \in \Delta^*$ we define $\langle q, \sigma.w, a.\alpha \rangle \Rightarrow_{\mathcal{P}} \langle q', w, \beta.\alpha \rangle$ iff $(q, \sigma, a, q', \beta) \in \delta$.

For a PDA $\mathcal{P}$, we have $w \in \mathcal{L}(\mathcal{P})$ iff $\langle q^0, w, \perp \rangle \Rightarrow^*_{\mathcal{P}} \langle q, \epsilon, \perp \rangle$.

4.2 Overview of the algorithm

We now outline the construction of a PDA $\mathcal{A}$, which we use in order to reduce the reversal- and register-bounded reachability problem to checking emptiness of a PDA. We begin by

describing the input of the automata involved in the construction and then proceed to give an overview of the construction followed by a formal definition of the input alphabets of these automata.

The automaton $\mathcal{A}$ reads words that consist of traces in Σ^*. In order to reflect sufficient information about the corresponding nodes and registers in the underlying graph, these traces are annotated as follows. First, since graphs derived by $\mathcal{G}^k$ contain at most k registers, we assume these registers to have unique identifiers from the set $\{1, \ldots, k\}$. Thus, a triple $(\sigma, j_1, j_2) \in \Sigma \times \{0, \ldots, k\} \times \{0, \ldots, k\}$ consists of an edge label σ and the identifiers of the registers at the source and target node of the edge, where 0 indicates no register at the respective node. We add additional annotation to reflect which nodes are shared in the corresponding paths, that is, positions where paths in the series–parallel graph branch off or join.

The automaton $\mathcal{A}$ reads such annotated traces and checks the existence of a run by emulating the behaviour of the machines on these traces by guessing an execution for each of them. An *execution* of $\mathcal{M} = (Q, q^0, \Sigma, \delta)$ is a sequence $\xi = q_0, (\sigma_1, b_1, b_1'), q_1, \ldots, (\sigma_f, b_f, b_f'), q_f$, where $q_0, q_1, \ldots, q_f \in Q$, such that $(q_{l-1}, \sigma_l, b_l, q_l, d_l, b_l') \in \delta$ for some $d_l \in \{1, -1\}$. In addition to verifying that each guess is indeed an execution, $\mathcal{A}$ needs to also check that the values written to and read from the shared registers by different machines are consistent.

Formally, an annotated trace and executions of the machines define a *read–write* sequence $\eta = (j_1, b_1, b_1'), \ldots, (j_f, b_f, b_f') \in (\{0, \ldots, k\} \times \mathbb{B} \times \mathbb{B})^*$, where, intuitively, j_i is the location that is read and/or written. Such a read–write sequence η is *valid w.r.t. an initial register valuation* $\beta_0 : \{1, \ldots, k\} \to \mathbb{B}$ iff each read operation reads the value written by the most recent write operation, or the initial value from β_0 if it is not overwritten. Formally, η is consistent with β_0 iff for every index $i \in \{1, \ldots, f\}$ with $j_i > 0$ it holds that if there is $i' < i$ such that $j_{i'} = j_i$, then $b_i = b_{i'}'$ for the largest such i', and otherwise $b_i = \beta_0(j_i)$.

Thus, the automaton $\mathcal{A}$ accepts tuples of traces in some graph derived by $\mathcal{G}^k$, annotated with information about registers and about nodes shared by the corresponding paths in the graph. $\mathcal{A}$ also guesses an execution for each of the m machines. The PDA $\mathcal{A}$ is constructed as the intersection of a PDA $\mathcal{P}_\mathsf{t}$ (Sect. 4.3) and an NFA $\mathcal{A}_\mathsf{e}$. $\mathcal{P}_\mathsf{t}$ checks that its input word encodes a tuple of traces in some graph derived by $\mathcal{G}^k$ and that these are correctly annotated with information about registers and the nodes that are shared among the paths corresponding to these traces. The NFA $\mathcal{A}_\mathsf{e}$ guesses and verifies the executions of the machines. It is obtained as the intersection of $m + 2$ NFAs: m NFAs A_i, one for each $i \in \{1, \ldots, m\}$, an NFA $\mathcal{A}_\mathsf{c}$ and an NFA $\mathcal{A}_\mathsf{s}$. The NFA $\mathcal{A}_i$ verifies that the guess of an execution of machine $i \in \{1, \ldots, m\}$ is correct. We describe the construction of $\mathcal{A}_i$ as a 2NFA (Sect. 4.4) which is then converted to an NFA using standard techniques [16]. Automaton $\mathcal{A}_\mathsf{c}$ checks the validity of the read–write sequence corresponding to the annotated traces and the guessed executions (Sect. 4.5). Automaton $\mathcal{A}_\mathsf{s}$ (Sect. 4.6) checks that a configuration in F is reached. The reversal- and register-bounded reachability problem thus reduces to checking emptiness of the language of the constructed automaton $\mathcal{A}$.

According to Sect. 3, it suffices to reason about $t = m \cdot (r + 1)$ traces in graphs derived by $\mathcal{G}^k$. To this end, we define the *trace alphabet* $\Sigma_\mathsf{t} = \Big(\big(\Sigma \,\dot\cup\, \{\flat\}\big) \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}\Big)^t \,\dot\cup\, \{1, \ldots, t\}^t$. Words over Σ_t are tuples of t traces, annotated with additional information. Each letter in Σ_t contains one row for each of the m machines and each of the $\widetilde{r} = r + 1$ paths corresponding to it. There are two types of letters. Each row in a letter of the first type consists of a letter in Σ (or the special symbol $\flat$) together with two *register*

identifiers in $\{0, \ldots, k\}$ and a *path index* in $\{1, \ldots, t\}$. The letters of the second type are t-tuples of path indices in $\{1, \ldots, t\}$, where equal indices indicate paths sharing a node. The special symbol $\flat$ is used in order to align traces in a way that the letters in the set $\{1, \ldots, t\}^t$ reflect the information about nodes shared by the respective paths (by suitably padding the traces of shorter paths to align them with longer parallel paths).

The *execution alphabet* $\Sigma_{\mathsf{e}} = \big(\{0, \ldots, p\} \times \mathbb{B} \times \mathbb{B} \times Q \times \{1, \ldots, t\}\big)^t$ is used to describe tuples of executions, one for each of the m machines. Each letter contains $t = \widetilde{r} \cdot m$ rows: $\widetilde{r} = r + 1$ rows for each of the m machines, one for each of its paths. Each row in the letter consists of a *block number* in $\{0, \ldots, p\}$, two *register values* (one for the read and one for the write operations), a *successor state* and an *index of a row* in an associated trace word (the trace word is a word in Σ_{t}^*, i.e., a tuple of t traces).

Let $\widetilde{\Sigma} = \Sigma_{\mathsf{t}} \times \Sigma_{\mathsf{e}}$ be the product of the trace and execution alphabets.

In what follows, if $\widetilde{\tau} = \widetilde{\sigma}_1 \cdots \widetilde{\sigma}_f \in \Sigma_{\mathsf{t}}^*$, then $\widetilde{\sigma}_j = (\widetilde{\sigma}_{1,j}, \ldots, \widetilde{\sigma}_{t,j})$ denotes the elements of the jth letter of the word $\widetilde{\tau}$ for $j \in \{1, \ldots, f\}$, and we use $\widetilde{\tau}_i = \widetilde{\sigma}_{i,1} \cdots \widetilde{\sigma}_{i,f}$ to denote the ith row of $\widetilde{\tau}$ for $i \in \{1, \ldots, t\}$. Similarly, if $\widetilde{\tau} = \widetilde{\sigma}_1 \cdots \widetilde{\sigma}_f \in \widetilde{\Sigma}^*$, the jth letter is $\widetilde{\sigma}_j = (\widetilde{\sigma}_{1,j}, \ldots, \widetilde{\sigma}_{t,j}, \widetilde{\eta}_{1,1,j}, \ldots, \widetilde{\eta}_{1,\widetilde{r},j}, \ldots, \widetilde{\eta}_{m,1,j}, \ldots, \widetilde{\eta}_{m,\widetilde{r},j})$, for $j \in \{1, \ldots, f\}$, and the ith row is $\widetilde{\tau}_i = \widetilde{\sigma}_{i,1} \cdots \widetilde{\sigma}_{i,f}$, for $i \in \{1, \ldots, t\}$. For $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $j \in \{1, \ldots, f\}$, the corresponding letter from Σ_{e} is denoted with $\widetilde{\eta}_{n,h,j} = (p_{n,h,j}, b_{n,h,j}, b'_{n,h,j}, q'_{n,h,j}, t_{n,h,j})$.

In the remainder of this section we present the automata constructions and formalize their properties, which together imply the correctness of our algorithm.

4.3 PDA accepting traces in a graph

The PDA $\mathcal{P}_{\mathsf{t}}$ is the product of a PDA $\mathcal{P}$ obtained from $\mathcal{G}^k$, where $\mathcal{G} = (V, \Sigma, R, G_0)$ is an SPGG, and an NFA $\mathcal{A}_{\mathsf{r}}$ that verifies the placement of register identifiers.

The construction of $\mathcal{P}$ resembles the classical construction of a PDA given a CFG. Here, instead of words generated by a CFG the language $\mathcal{L}(\mathcal{P})$ of $\mathcal{P}$ consists of t-tuples of (annotated) traces in some graph generated by the grammar. The stack alphabet of $\mathcal{P}$ contains symbols corresponding to the variables and terminal symbols of the SPGG $\mathcal{G}$, and the transitions of $\mathcal{P}$ can be categorized according to the top symbol on the stack. Transitions for stack symbols corresponding to variables in $\mathcal{G}$ reflect the production rules of $\mathcal{G}$. The most interesting are the transitions for parallel composition, in which $\mathcal{P}$ guesses symbols, in the graphs generated by which the corresponding traces occur, together with the number of traces in the subgraph generated by each symbol. The number of times a new branch is introduced is bounded by t, the number of parallel traces.

For the series composition rules, δ_p employs the additional symbol $\flat$ to allow for traces that are aligned in a way that letters in $\{1, \ldots, t\}^t$ reflect the information about nodes shared by the respective paths.

We now describe the formal construction of the PDA

$$\mathcal{P} = (Q_p, q_p^0, \Sigma_{\mathsf{t}}, \Sigma_{\mathsf{t}} \dot\cup \widetilde{V} \dot\cup \{\bot\}, \bot, \delta_p).$$

– The set of states is $Q_p = \{q_p^0, q\}$. The set of stack symbols is $\Sigma_{\mathsf{t}} \dot\cup \widetilde{V} \dot\cup \{\bot\}$, where the set $\widetilde{V}$ consists of symbols corresponding to the variables in $\mathcal{G}$

$$\widetilde{V} = \Big\{ \big\langle (v_1, j'_1, j''_1, t_1), (v_2, j'_2, j''_2, t_2), \ldots, (v_u, j'_u, j''_u, t_u) \big\rangle \mid \\ \textstyle\sum_{i=1}^{u} t_i = t, \forall i \in \{1, \ldots, u\}.\ v_i \in V \cup \{\flat\} \wedge j'_i, j''_i \in \{0, \ldots, k\} \Big\}.$$

– The transitions in δ_p can be grouped according to the top symbol on the stack: empty stack, top symbol $\widetilde{\sigma} \in \Sigma_{\mathrm{t}}$, and top symbol $\widetilde{v} \in \widetilde{V}$. Transitions for $\widetilde{v} \in \widetilde{V}$ correspond to the production rules of the SPGG $\mathcal{G}$. The transition relation δ_p is defined as follows:

  – $\delta_p(q_p^0, \epsilon, \perp) = \Big\{\big(q, \langle(v_0, 0, 0, t)\rangle\perp\big)\Big\}$, where v_0 is the start symbol of $\mathcal{G}$.
  – $\delta_p(q, \widetilde{\sigma}, \widetilde{\sigma}) = \big\{(q, \epsilon)\big\}$ for $\widetilde{\sigma} \in \Sigma_{\mathrm{t}}$, $\delta_p(q, \epsilon, \perp) = \big\{(q, \perp)\big\}$.
  – For $\widetilde{v} = \big\langle(v_1, j_1, j_1', t_1), (v_2, j_2, j_2', t_2), \ldots, (v_u, j_u, j_u', t_u)\big\rangle \in \widetilde{V}$, we define

$$\delta_p(q, \epsilon, \widetilde{v}) = \delta_p'(q, \epsilon, \widetilde{v}) \cup \delta_p''(q, \epsilon, \widetilde{v}) \cup \delta_p'''(q, \epsilon, \widetilde{v}),$$

  where δ_p', δ_p'' and δ_p''' correspond to cases (1), (2) and (3) for $\mathcal{G}$'s rules.

  The SPGG $\mathcal{G}^k$ obtained from $\mathcal{G}$ marks the nodes of the derived graph, making sure that at most k nodes in the graph are marked. Thus, we suppose that each rule (v, G') of $\mathcal{G}^k$ comes with a mapping κ' that maps the nodes of G' to Boolean values indicating which nodes should be marked. A node in the resulting graph is marked if it is marked by at least one rule (that is, when fusing nodes the result is marked if at least one of the respective nodes that are being fused is marked).

$$\begin{aligned}
\delta_p'(q, \epsilon, \widetilde{v}) = \Big\{\big(q, ((\sigma_1, \widehat{j_1}, \widehat{j_1'}, 1)^{t_1}, (\sigma_2, \widehat{j_2}, \widehat{j_2'}, 2)^{t_2}, \ldots, (\sigma_u, \widehat{j_u}, \widehat{j_u'}, u)^{t_u})\big) \mid \\
\forall i \in \{1, \ldots, u\}.\ \big(v_i, \sigma_i) \in R \text{ and has associated mapping } \kappa' \wedge \\
\qquad (\widehat{j_i} > 0 \Leftrightarrow j_i > 0 \vee \kappa'(n_b')) \wedge (\widehat{j_i'} > 0 \Leftrightarrow j_i' > 0 \vee \kappa'(n_e'))\big) \\
\qquad \vee \big(v_i = \sigma_i = \flat \wedge \widehat{j_i} = j_i \wedge \widehat{j_i'} = j_i'\big)\Big\}, \\
\delta_p''(q, \epsilon, \widetilde{v}) = \Big\{\big(q, \langle(x_1, \widehat{j_1}, \widehat{j_1'}, t_1), (x_2, \widehat{j_2}, \widehat{j_2'}, t_2), \ldots, (x_u, \widehat{j_u}, \widehat{j_u'}, t_u)\rangle \cdot \\
\qquad \langle(y_1, \widehat{j_1'}, \widehat{j_1''}, t_1), (y_2, \widehat{j_2'}, \widehat{j_2''}, t_2), \ldots, (y_u, \widehat{j_u'}, \widehat{j_u''}, t_u))\big) \mid \\
\forall i \in \{1, \ldots, u\}.\big((v_i, x_i \cdot y_i) \in R \text{ has associated mapping } \kappa' \wedge \\
\qquad (\widehat{j_i} > 0 \Leftrightarrow j_i > 0 \vee \kappa'(n_b')) \wedge (\widehat{j_i'} > 0 \Leftrightarrow \kappa'(n')) \wedge \\
\qquad (\widehat{j_i''} > 0 \Leftrightarrow j_i > 0 \vee \kappa'(n_e'))\big) \\
\qquad \vee \big(x_i = v_i \wedge y_i = \flat \wedge \widehat{j_i} = j_i \wedge \widehat{j_i'} = j_i' \wedge \widehat{j_i''} = 0\big) \\
\qquad \vee \big(x_i = \flat \wedge y_i = v_i \wedge \widehat{j_i} = 0 \wedge \widehat{j_i'} = j_i \wedge \widehat{j_i''} = j_i'\big)\Big\}, \\
\delta_p'''(q, \epsilon, \widetilde{v}) = \Big\{\big(q, (1^{t_1}, 2^{t_2}, \ldots, u^{t_u}) \cdot \\
\qquad \langle(x_1, \widehat{j_1}, \widehat{j_1'}, t_1'), (x_2, \widehat{j_2}, \widehat{j_2'}, t_2), \ldots, (x_{u'}, \widehat{j_{u'}}, \widehat{j_{u'}'}, t_{u'}')\rangle \cdot \\
\qquad (1^{t_1}, 2^{t_2}, \ldots, u^{t_u})\big) \mid t \geq u' \geq u \ \wedge \exists f \in \{1, \ldots, u\}^{\{1, \ldots, u'\}}. \\
\quad \forall i_1, i_2, i'.i.\big((i_1 \leq i_2 \Rightarrow f(i_1) \leq f(i_2)) \wedge f(i_1) < f(i_1 + 2)\big) \wedge \\
\qquad \big(f(i_1) = f(i_2) = i \Rightarrow t_{i_1}' + t_{i_2}' = t_i \wedge j_{i_1}'' = j_{i_2}'' \wedge j_{i_1}''' = j_{i_2}''' \wedge \\
\qquad\qquad (v_i, x_{i_1} \parallel x_{i_2}) \in R \text{ has associated mapping } \kappa' \wedge \\
\qquad\qquad (\widehat{j_{i_1}} > 0 \Leftrightarrow j_i > 0 \vee \kappa'(n_b')) \wedge \\
\qquad\qquad (\widehat{j_{i_1}'} > 0 \Leftrightarrow j_i' > 0 \vee \kappa'(n_e'))\big) \wedge \\
\qquad \big(f(i') = i \wedge i > f(i' - 1) \wedge i < f(i' + 1) \Rightarrow \\
\qquad\qquad (\exists x'.(v_i, x_{i'} \parallel x') \in R \text{ has associated mapping } \kappa' \wedge \\
\qquad\qquad (\widehat{j_{i'}} > 0 \Leftrightarrow j_i > 0 \vee \kappa'(n_b')) \wedge \\
\qquad\qquad (\widehat{j_{i'}'} > 0 \Leftrightarrow j_i' > 0 \vee \kappa'(n_e'))) \vee \\
\qquad\qquad (x_{i'} = v_i \wedge \widehat{j_{i'}} = \widehat{j_i} \wedge \widehat{j_{i'}'} = j_i')\big)\Big\}.
\end{aligned}$$

  – δ_p is undefined in all other cases.

$\mathcal{P}$ does not check that the register identifiers in the annotation are consistent among letters corresponding to edges in the graph that share a node, i.e., that letters corresponding to these edges have the same identifier for this node. This is done by the NFA $\mathcal{A}_{\mathsf{r}}$, which also verifies that identifiers for different nodes are unique.

The NFA $\mathcal{A}_{\mathsf{r}} = (Q_{\mathsf{r}}, q^0_{\mathsf{r}}, \Sigma_{\mathsf{t}} \dot\cup \{\vdash, \dashv\}, \delta_{\mathsf{r}}, F_{\mathsf{r}})$ has the following components.

- $Q_{\mathsf{r}} = \{1, \ldots, t\}^t \times \{0, \ldots, k\}^t \times 2^{\{0,\ldots,k\}}$, $q^0_{\mathsf{r}} = (1, \ldots, 1, 0, \ldots, 0, \emptyset)$.

  Each state $\widetilde{q}$ of $\mathcal{A}_{\mathsf{r}}$ contains a path index $l_h \in \{1, \ldots, t\}$ and a register identifier $i_h \in \{0, \ldots, k\}$ for each row $\widetilde{\tau}_h$ of $\widetilde{\tau}$, and a set of already seen register identifiers.
- The transition relation δ_{r} checks that the letters in $\widetilde{\tau}$ that correspond to edges incident with the same node agree on the corresponding register identifier. The path indices l_h in $\widetilde{q}$ are used to identify branching or joining paths and the register identifiers i_h to check the required equalities.δ_{r} also verifies that the register identifiers corresponding to different nodes are different. We let
  $((l_1, \ldots, l_t, i_1, \ldots, i_t, J), (\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t), (l'_1, \ldots, l'_t, i'_1, \ldots, i'_t, J')) \in \delta_{\mathsf{r}}$ iff
  (1) if $h, h' \in \{1, \ldots, t\}$, $\widetilde{\sigma}_h = (\sigma, l, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$ and $\widetilde{\sigma}_{h'} = (\sigma', l', j'_1, j'_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, then:
     - f $J \neq \emptyset$, then $j_1 = i_h$, $j_1 \notin J\backslash\{0\}$, $j_2 \notin J\backslash\{0\}$,
     - if $l' = l$, then $j_1 = j'_1$ and $j_2 = j'_2$,
     - if $l' \neq l$, $l_h \neq l_{h'}$ and $i_h \neq i_{h'}$, then $j_1 \neq j'_1$ or $j_1 = j'_1 = 0$;

  (2) if $\widetilde{\sigma}_h \in \{1, \ldots, t\}$, then for every $h' \in \{1, \ldots, t\}$, if $\widetilde{\sigma}_{h'} = \widetilde{\sigma}_h$ then $i_{h'} = i_h$.
  (3) $J' = J \cup \left\{ j \in \{0, \ldots, k\} \mid \exists h \in \{1, \ldots, t\}.\widetilde{\sigma}_h = (\sigma, d, l, j_1, j_2) \wedge j_1 = j \right\}$;
  (4) for each $h \in \{1, \ldots, t\}$
     - if $\widetilde{\sigma}_h = (\sigma, l, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, then $l'_h = l, i'_h = j_2$,
     - if $\widetilde{\sigma}_h \in \{\flat\} \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2 \cup \{1, \ldots, t\}$, then $l'_h = l_h, i'_h = i_h$.

  - If $\widetilde{\sigma} \in \{\vdash, \dashv\}$, then $((l_1, \ldots, l_t, i_1, \ldots, i_t, J), \widetilde{\sigma}, (l'_1, \ldots, l'_t, i'_1, \ldots, i'_t, J')) \in \delta_{\mathsf{r}}$ iff $J' = J$, and for all $h \in \{1, \ldots, t\}$ it holds that $l'_h = l_h$ and $i'_h = i_h$.
  - $F_{\mathsf{r}} = \left\{ (l_1, \ldots, l_t, i_1, \ldots, i_t, J) \in Q_{\mathsf{r}} \mid \forall h, h' \in \{1, \ldots, t\}.i_h = i_{h'} \right\}$. In an accepting state, the equalities for the sink node of the graph must be satisfied.

Finally, we construct the PDA $\mathcal{P}_{\mathsf{t}}$ with $\mathcal{L}(\mathcal{P}_{\mathsf{t}}) = \mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\mathcal{A}_{\mathsf{r}})$. The construction of $\mathcal{P}$ and $\mathcal{A}_{\mathsf{r}}$ ensures that if $\widetilde{\tau} \in \mathcal{L}(\mathcal{P}_{\mathsf{t}})$, then there exists $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ and for each $i \in \{1, \ldots, t\}$ there exists a sequence of nodes $\widetilde{\pi}_i$ in G such that for each row $\widetilde{\tau}_i$ of $\widetilde{\tau}$ there exists a subsequence $\pi_i \in \mathsf{Paths}(G, \tau_i)$ of $\widetilde{\pi}_i$ corresponding to the projection $\tau_i = \left(\widetilde{\tau}_i|_{\Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}}\right)|_\Sigma$ of $\widetilde{\tau}_i$ on Σ. Furthermore, these paths can be chosen such that edges corresponding to rows with the same path index connect the same pair of nodes. Additionally, the mapping κ for the nodes on these paths agrees with the corresponding register identifiers in $\widetilde{\tau}$. This is formalized in the following proposition, the proof of which is given in the "Appendix".

Proposition 4 *If a word $\widetilde{\tau} = \widetilde{\sigma}_1 \cdots \widetilde{\sigma}_f \in \Sigma^*_{\mathsf{t}}$ is accepted by $\mathcal{P}_{\mathsf{t}}$, then there exists $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ with $G = (N, E, n_b, n_e)$ and for each $i \in \{1, \ldots, t\}$ there exists a sequence of nodes $\widetilde{\pi}_i = \widetilde{n}_{i,0}, \ldots, \widetilde{n}_{i,f}$ in the graph G such that all of the following conditions are satisfied.*

(1) For each $i \in \{1, \ldots, t\}$ it holds that:

- *For each $1 < j \leq f$:*
  - *If $\widetilde{\sigma}_{i,j} = (\sigma, j_1, j_2, l) \in \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$, then there exists $e \in E$ with $src(e) = \widetilde{n}_{i,(j-1)}$, $trg(e) = \widetilde{n}_{i,j}$ and $\alpha(e) = \sigma$,*

– *If* $\widetilde{\sigma}_{i,j} \in \big(\{\flat\} \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}\big) \cup \{1,\ldots,t\}$, *then* $\widetilde{n}_{i,(j-1)} = \widetilde{n}_{i,j}$.
– *Let* $\widetilde{\tau}_i = \big(\widetilde{\tau}_i|_{\Sigma\times\{0,\ldots,k\}^2\times\{1,\ldots,t\}}\big)|_\Sigma$ *be the projection of* $\widetilde{\tau}_i$.
Then, there exists a subsequence $\pi_i = n_0,\ldots,n_{f_i}$ *of* $\widetilde{\pi}_i$ *with* $\pi_i \in \mathsf{Paths}(G,\tau_i)$.

(2) For all $i_1, i_2 \in \{1,\ldots,t\}$ *and* $j \in \{1,\ldots,f\}$ *it holds that:*

– *If* $\widetilde{\sigma}_{i_1,j}, \widetilde{\sigma}_{i_2,j} \in \Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}$, $\widetilde{\sigma}_{i_1,j} = (\sigma, j_1, j_2, l)$ *and* $\widetilde{\sigma}_{i_2,j} = (\sigma', j_1', j_2', l')$, *then* $l = l'$ *implies* $\widetilde{n}_{i_1,(j-1)} = \widetilde{n}_{i_2,(j-1)}$, $\widetilde{n}_{i_1,j} = \widetilde{n}_{i_2,j}$ *and* $\sigma = \sigma'$.
– *If* $\widetilde{\sigma}_j \in \{1,\ldots,t\}^t$, *then* $\widetilde{\sigma}_{i_1,j} = \widetilde{\sigma}_{i_2,j}$ *iff* $\widetilde{n}_{i_1,j-1} = \widetilde{n}_{i_2,j-1}$, $\widetilde{n}_{i_1,j} = \widetilde{n}_{i_2,j}$.

(3) For every $i \in \{1,\ldots,t\}$ *and* $j \in \{1,\ldots,f\}$ *with* $\widetilde{\sigma}_{i,j} = (\sigma, j_1, j_2, l) \in \Sigma\times\{0,\ldots,k\}^2\times\{1,\ldots,t\}$ *it holds that* $j_1 > 0$ *iff* $\kappa(\widetilde{n}_{i,(j-1)})$ *and* $j_2 > 0$ *iff* $\kappa(\widetilde{n}_{i,j})$.

Conversely, if $(G,\kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ and for every $n \in \{1,\ldots,m\}$ and $h \in \{1,\ldots,\widetilde{r}\}$ we are given a path $\widehat{\pi}_{n,h} \in \mathsf{Paths}(G,\widehat{\tau}_{n,h})$ for some trace $\widehat{\tau}_{n,h} \in \Sigma^*$, then there exists a word $\widetilde{\tau} \in \mathcal{L}(\mathcal{P}_\mathrm{t})$, which corresponds to these paths and traces. The word $\widetilde{\tau}$ is obtained by ordering, extending and annotating the given traces. This direction is formalized in the proposition below.

Proposition 5 *Let* $(G,\kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ *and* $G = (N, E, n_b, n_e)$. *Let for every* $n \in \{1,\ldots,m\}$ *and* $h \in \{1,\ldots,\widetilde{r}\}$, $\widehat{\tau}_{n,h} \in \Sigma^*$ *be a trace and* $\widehat{\pi}_{n,h} \in N^*$ *be a path such that* $\widehat{\pi}_{n,h} \in \mathsf{Paths}(G,\widehat{\tau}_{n,h})$.

Then, there exists a bijection $g : \{1,\ldots,m\}\times\{1,\ldots,\widetilde{r}\} \to \{1,\ldots,t\}$ *such that if for each* $i \in \{1,\ldots,t\}$ *we define* $\tau_i = \widehat{\tau}_{g^{-1}(i)} = \sigma_{i,1}\ldots\sigma_{i,f_i}$ *and* $\pi_i = \widehat{\pi}_{g^{-1}(i)} = n_{i,0},\ldots,n_{i,f_i}$, *then there exists a word* $\widetilde{\tau} = \widetilde{\sigma}_1\cdots\widetilde{\sigma}_f \in \mathcal{L}(\mathcal{P}_\mathrm{t})$ *that satisfies the following requirements.*

(1) For each $i \in \{1,\ldots,t\}$, *it holds that* $\tau_i = \big(\widetilde{\tau}_i|_{\Sigma\times\{0,\ldots,k\}^2\times\{1,\ldots,t\}}\big)|_\Sigma$.
(2) For all $i', i'' \in \{1,\ldots,t\}$, $j \in \{1,\ldots,f\}$, $j' \in \{1,\ldots,\widetilde{f}_{i'}\}$, $j'' \in \{1,\ldots,\widetilde{f}_{i''}\}$ *we have:*

– *If* $\widetilde{\sigma}_{i',j} = (\sigma', j_1', j_2', l')$, $\widetilde{\sigma}_{i'',j} = (\sigma'', j_1'', j_2'', l'')$, n *and* n' *are the nodes in* $\pi_{i'}$ *corresponding to the position of the respective occurrence of* σ' *in* $\tau_{i'}$, *and* n'' *and* n''' *are the nodes in* $\pi_{i''}$ *corresponding to the position of the respective occurrence of* σ'' *in* $\tau_{i''}$, *then* $l' = l''$ *implies* $n = n''$ *and* $n' = n'''$.
– *If* $\widetilde{\sigma}_{i',j'} = (\sigma', j_1', j_2', l')$, $\widetilde{\sigma}_{i'',j''} = (\sigma'', j_1'', j_2'', l'')$ *and* $l' \neq l''$, *then* $n_{i',j'} = n_{i'',j''}$ *iff there is* $\widetilde{\sigma}_{\widetilde{j}'''} = (\widetilde{\sigma}_{1,j'''},\ldots,\widetilde{\sigma}_{t,j'''}) \in \{1,\ldots,t\}^t$ *with* $\widetilde{\sigma}_{i',\widetilde{j}'''} = \widetilde{\sigma}_{i'',\widetilde{j}'''}$, *and*
  – $\widetilde{j}''' \le \widetilde{j}'$ *and* $\widetilde{j}''' \le \widetilde{j}''$, *where* $\widetilde{j}'$ *is the index in* $\widetilde{\tau}$ *corresponding to the index* j' *in* $\tau_{i'}$ *and* $\widetilde{j}''$ *is the index in* $\widetilde{\tau}$ *corresponding to* j'' *in* $\tau_{i''}$,
  – *for each* $\widetilde{j}''' \le \widetilde{j}'''' \le \widetilde{j}'$ *it holds that* $\widetilde{\sigma}_{i',\widetilde{j}''''} \notin \Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}$,
  – *for each* $\widetilde{j}''' \le \widetilde{j}'''' \le \widetilde{j}''$ *it holds that* $\widetilde{\sigma}_{i'',\widetilde{j}''''} \notin \Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}$.
– *If* $\widetilde{\sigma}_{i',j'} = (\sigma', j_1', j_2', l')$, $\widetilde{\sigma}_{i'',j''} = (\sigma'', j_1'', j_2'', l'')$ *and* $l' \neq l''$, *then* $n_{i',j'+1} = n_{i'',j''+1}$ *iff there is* $\widetilde{\sigma}_{\widetilde{j}'''} = (\widetilde{\sigma}_{1,j'''},\ldots,\widetilde{\sigma}_{t,j'''}) \in \{1,\ldots,t\}^t$ *such that* $\widetilde{\sigma}_{i',\widetilde{j}'''} = \widetilde{\sigma}_{i'',\widetilde{j}'''}$ *and:*
  – $\widetilde{j}''' \ge \widetilde{j}'$ *and* $\widetilde{j}''' \ge \widetilde{j}''$, *where* $\widetilde{j}'$ *is the index in* $\widetilde{\tau}$ *corresponding to the index* j' *in* $\tau_{i'}$ *and* $\widetilde{j}''$ *is the index in* $\widetilde{\tau}$ *corresponding to* j'' *in* $\tau_{i''}$,
  – *for each* $\widetilde{j}''' \ge \widetilde{j}'''' \ge \widetilde{j}$ *it holds that* $\widetilde{\sigma}_{i',\widetilde{j}''''} \notin \Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}$,
  – *for each* $\widetilde{j}''' \ge \widetilde{j}'''' \ge \widetilde{j}''$ *it holds that* $\widetilde{\sigma}_{i'',\widetilde{j}''''} \notin \Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}$.

(3) For each $i \in \{1,\ldots,t\}$ *and* $j \in \{1,\ldots,f_i\}$, *if* $\widetilde{\sigma} = (\sigma_{i,j}, j_1, j_2, l)$ *is the letter of* $\widetilde{\tau}_i$ *corresponding to the letter* $\sigma_{i,j}$ *of* τ_i, *then* $j_1 > 0$ *iff* $\kappa(n_{i,(j-1)})$ *and* $j_2 > 0$ *iff* $\kappa(n_{i,j})$.

4.4 2NFA accepting executions

We construct a 2NFA $\widetilde{\mathcal{A}_n}$ for each $n \in \{1, \ldots, m\}$ that checks that the sequence described by the rows of the word that correspond to n is indeed an execution of $\mathcal{M}$ that reads the corresponding rows of the trace word. Furthermore, $\widetilde{\mathcal{A}_n}$ verifies that the machine switches between traces described by different rows of the trace word only at positions at which the traces share a node in the corresponding paths. The automaton $\widetilde{\mathcal{A}_n}$ is a two-way automaton, that reverses its direction at the two endmarkers of the word, and makes exactly r reversals. Since the reversals of $\mathcal{M}_n$ can occur at arbitrary positions of the trace, $\widetilde{\mathcal{A}_n}$ additionally verifies that the machine's state remains unchanged in parts of the trace where $\mathcal{M}_n$ is inactive.

The 2NFA $\widetilde{\mathcal{A}_n} = (\widetilde{Q}_n, \widetilde{Q}_n^0, \widetilde{\Sigma} \dot{\cup} \{\vdash, \dashv\}, \widetilde{\delta}_n, \widetilde{F}_n)$ has the following components.

- $\widetilde{Q}_n = Q \times \{1, \ldots, t\} \times \{1, \ldots, \widetilde{r}\} \times \{\top, \bot, *\}$.
  Each state $\widetilde{q} \in \widetilde{Q}_n$ of $\widetilde{\mathcal{A}_n} = (\widetilde{Q}_n, \widetilde{Q}_n^0, \widetilde{\Sigma} \dot{\cup} \{\vdash, \dashv\}, \widetilde{\delta}_n, \widetilde{Q}_n)$ contains a state $q \in Q$ of $\mathcal{M}$, which is the current state of the simulated machine, and an index $i \in \{1, \ldots, t\}$ in the trace-word that is part of the input word. The row of the trace word that is read in state $\widetilde{q}$ is determined by i. The third component of the state counts the number of reversals of $\mathcal{A}_n$. The last component is used to check that block number 0 in $\widetilde{\tau}$ is used to correctly encode the reversals of the machine (which do not have to be at the start or sink nodes of the graph).
- $\widetilde{Q}_n^0 = \{(q^0, i, 1, \bot) \mid i \in \{1, \ldots, t\}\}$ and $\widetilde{F}_n = \widetilde{Q}_n$.
  In the initial state of the automaton the machine is in its initial state q^0, and all states of the automaton are accepting.
- The transition relation $\widetilde{\delta}_n$ refers to the transition relation δ of $\mathcal{M}$ to check the existence of a transition of $\mathcal{M}$ that performs the read and write operations determined by the read letter of $\widetilde{\tau}$. The state q of the machine is updated according to δ and remains unchanged when the machine is inactive in the current part of the trace. The index i of a row in the trace word can be changed by $\widetilde{\delta}_n$ only if for the current letter we have $\widetilde{\sigma}_{t_{n,h}} \in \{1, \ldots, t\}$ and $p_{n,h} > 0$, and for the new value i' it must hold that $\sigma_{i'} = \sigma_i$. That is, the machine can switch between traces only at positions where the paths intersect.
  Fix $\widetilde{\sigma} = (\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t, \widetilde{\eta}_{1,1}, \ldots, \widetilde{\eta}_{1,\widetilde{r}}, \ldots, \widetilde{\eta}_{m,1}, \ldots, \widetilde{\eta}_{m,\widetilde{r}}) \in \widetilde{\Sigma}$, where for all $h \in \{1, \ldots, \widetilde{r}\}$, we have $\widetilde{\eta}_{n,h} = (p_{n,h}, b_{n,h}, b'_{n,h}, q'_{n,h}, t_{n,h})$.
  We let $((q, i, h, z), \widetilde{\sigma}, (q', i', h', z'), d) \in \widetilde{\delta}_n$ iff the following conditions hold.

  (1)
  - If $\widetilde{\sigma}_{t_{n,h}} = (\sigma, l, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$ and $p_{n,h} > 0$, then $(q, \sigma, b_{n,h}, q'_{n,h}, d, b'_{n,h}) \in \widetilde{\delta}$ and $q' = q'_{n,h}$.
  - If $p_{n,h} = 0$ or $\widetilde{\sigma}_{t_{n,h}} \in \{\flat\} \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2 \cup \{1, \ldots, t\}$, then $q' = q$.

  (2)
  - $h' = h$, and if h is odd, then $d = 1$ and if h is even, then $d = -1$;
  - if $p_{n,h} > 0$, then $t_{n,h} = i$;
  - if $\widetilde{\sigma}_{t_{n,h}} \in \{1, \ldots, t\}$ and $p_{n,h} > 0$, then $\sigma_{i'} = \sigma_{t_{n,h}}$, otherwise $i' = i$;

  (3)
  - if $z = \top$, then $p_{n,h} = p_{n,h+1} = 0$, and if $z = *$, then $p_{n,h} = p_{n,h-1} = 0$;
  - if $z = \bot$ and $p_{n,h} = 0$, then $z' = \top$; if $z = *$ and $p_{n,h} > 0$, then $z' = \bot$; $z' = z$, in all other cases.

- For $\widetilde{\sigma} \{\vdash, \dashv\}$ we have $((q, i, h, z), \widetilde{\sigma}, (q', i', h', z'), d) \in \widetilde{\delta}_n$ iff
  - $q' = q$, $i' = i$ and $z' = *$ if $z = \top$ and $z' = z$ otherwise;
  - if $\widetilde{\sigma} = \vdash$ and $h = 1$, then $h' = h$ and $d = 1$;
  - if $\widetilde{\sigma} = \vdash$, $h \neq 1$ and $h < \widetilde{r}$, then $d = 1$ and $h' = h + 1$;
  - if $\widetilde{\sigma} = \vdash$, $h \neq 1$ and $h = \widetilde{r}$, then $d = -1$ and $h' = h$;

– if $\widetilde{\sigma} = \dashv$ and $h = \widetilde{r}$, then $d = 1$ and $h' = h$;
– if $\widetilde{\sigma} = \dashv$ and $h < \widetilde{r}$, then $d = -1$ and $h' = h + 1$.

By construction, $\widetilde{\tau} \in \mathcal{L}(\widetilde{\mathcal{A}_n})$ iff by taking the elements of $\widetilde{\tau}$ corresponding to machine n in the appropriate order we can construct an execution ξ_n. The construction of ξ_n is given in the following proposition, proven in the "Appendix".

Proposition 6 *Let $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}^*$. Then $\widetilde{\tau} \in \mathcal{L}(\widetilde{\mathcal{A}_n})$ iff $\widetilde{\tau}$ satisfies the following conditions.*

(1) For each $1 \leq h \leq \widetilde{r}$ there exist $1 \leq l'_h \leq l''_h \leq f$ such that:

– *for every l such that $1 \leq l < l'_h$ or $l''_h < l \leq f$ it holds that $p_{n,h,l} = 0$,*
– *for every l such that $l'_h \leq l \leq l''_h$ it holds that $p_{n,h,l} > 0$,*
– *if h is odd and $h < \widetilde{r}$, $l''_h = l''_{h+1}$; if h is even and $h > 1$, $l'_h = l'_{h-1}$.*

(2) For each $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$ we define $\widehat{\xi}_{n,h,l}$ as follows:

– *If $\widetilde{\sigma}_{t_{n,h,l},l} = (\sigma, c, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, $p_{n,h,l} > 0$ and h is odd, then let $\widehat{\xi}_{n,h,l} = (\sigma, b_{n,h,l}, b'_{n,h,l}) \cdot q'_{n,h,l}$,*
– *If $\widetilde{\sigma}_{t_{n,h,l},l} = (\sigma, c, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, $p_{n,h,l} > 0$ and h is even, then let $\widehat{\xi}_{n,h,l} = q'_{n,h,l} \cdot (\sigma, b_{n,h,l}, b'_{n,h,l})$.*
– *In all other cases, let $\widehat{\xi}_{n,h,l} = \varepsilon$ be the empty word.*

Let $\widehat{\xi}_{n,h} = \widehat{\xi}_{n,h,1} \cdot \cdots \cdot \widehat{\xi}_{n,h,f}$ if h is odd, and $\widehat{\xi}_{n,h} = \widehat{\xi}_{n,h,f} \cdot \cdots \cdot \widehat{\xi}_{n,h,1}$ otherwise.

The sequence $\xi_n = q^0 \cdot \widehat{\xi}_{n,1} \cdot \cdots \cdot \widehat{\xi}_{n,\widetilde{r}}$ is an execution of $\mathcal{M}$.

4.5 2NFA accepting valid read–write sequences

Here we describe a 2NFA $\widetilde{\mathcal{A}_\mathsf{c}}$ that checks that the executions of the different machines described by the input word are compatible with each other. That, is that the read and write operations of different machines match when executed in the order determined by the input word, where each operation is labelled with a block number. $\widetilde{\mathcal{A}_\mathsf{c}}$ verifies that each block number is used in a single execution and that for each execution the sequence of positive block numbers is nondecreasing. To check the validity of the corresponding read–write sequence w.r.t. the initial register values, $\widetilde{\mathcal{A}_\mathsf{c}}$ tracks the register values at the end and at the beginning of each block and compares the values at the beginning of block $i + 1$ with those at the end of block i. An *assumption* is a partial function $A : \{1, \ldots, p\} \to \mathbb{B}^k$ that maps a block number to a valuation of the registers, representing the obligation to verify that at the beginning of a block the registers have the respective values. Similarly, a *guarantee* is a function $G : \{1, \ldots, p\} \to \mathbb{B}^k$ used to propagate the guarantee that at the end of a block the registers have a certain value. A read–write word is accepted if all the assumptions have a matching guarantee.

The 2NFA $\widetilde{\mathcal{A}_\mathsf{c}} = (\widetilde{Q_\mathsf{c}}, \{\widetilde{q^0_\mathsf{c}}\}, \widetilde{\Sigma} \dot\cup \{\vdash, \dashv\}, \widetilde{\delta_\mathsf{c}}, \widetilde{F_\mathsf{c}})$ has components defined as follows.

– $\widetilde{Q_\mathsf{c}} = \{1, \ldots, \widetilde{r}\} \times \{1, \ldots, p\}^m \times 2^{\{1,\ldots,p\}} \times (\mathbb{B}^k)^m \times \left((\mathbb{B}^k)^{\{1,\ldots,p\}}\right)^2$.
Each state $\widetilde{q} = (h, p_1, \ldots, p_m, P, \beta_1, \ldots, \beta_m, A, G)$ of the automaton $\widetilde{\mathcal{A}_\mathsf{c}}$ contains a block number p_n and a valuation of the registers β_n for machine n, a set P of already seen block numbers, an assumption A and a guarantee G.
– $\widetilde{q^0_\mathsf{c}} = \big(1, 0, \ldots, 0, \emptyset, 0, \ldots, 0, \emptyset, \{(0, 0)\}\big)$ is the initial state.

– The transition relation $\widetilde{\delta_{\mathsf{c}}}$ checks that all read operations of machine n, except those at the beginning of a block, read the value stored in β_n. At the beginning of a block of machine n, $\widetilde{\delta_{\mathsf{c}}}$ guesses a valuation of the registers for read operations and stores them in β_n. The new block number and the guess are added to the set of assumptions A. The values of write operations are used to update β_n and, at the end of a block the respective guarantee is added to G. $\widetilde{\delta_{\mathsf{c}}}$ discharges assumptions in A for which the respective guarantees are in G.
Fix two states of $\widetilde{\mathcal{A}_{\mathsf{c}}}$:

$$\begin{aligned}\widetilde{q} &= (h, p_1, \ldots, p_m, P, \beta_1, \ldots, \beta_m, A, G)\\ \widetilde{q}' &= (h', p'_1, \ldots, p'_m, P', \beta'_1, \ldots, \beta'_m, A', G').\end{aligned}$$

For $\widetilde{\sigma} = (\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t, \widetilde{\eta}_{1,1}, \ldots, \widetilde{\eta}_{1,\widetilde{r}}, \ldots, \widetilde{\eta}_{m,1}, \ldots, \widetilde{\eta}_{m,\widetilde{r}}) \in \widetilde{\Sigma}$,
where $\widetilde{\eta}_{n,h} = (p_{n,h}, b_{n,h}, b'_{n,h}, q'_{n,h}, t_{n,h})$ we have $(\widetilde{q}, \widetilde{\sigma}, \widetilde{q}', d) \in \widetilde{\delta_{\mathsf{c}}}$ iff the requirements (1) and (2) given below are fulfilled.
For $n \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, \widetilde{r}\}$, if $\widetilde{\sigma}_{t_{n,h}} = (\sigma, l, j_1, j_2) \in \Sigma \times \{1, \ldots t\} \times \{0, \ldots, k\}^2$, then let $\widehat{j}_{n,h} = j_1$ if h is odd and $\widehat{j}_{n,h} = j_2$ if h is even. If $\sigma_{t_{n,h}} \notin \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, then let $\widehat{j}_{n,h} = 0$.

(1) All of the following conditions are satisfied:
  – $h' = h$, and if h is odd, then $d = 1$ and if h is even, then $d = -1$.
  – $P' = P \cup \{p_{n,h} \mid 1 \le n \le m,\ p_{n,h} > 0\}$.
  – For every $n \in \{1, \ldots, m\}$, $p'_n = p_{n,h}$ if $p_{n,h} > p_n$, and $p'_n = p_n$ otherwise.
  – For every $n \in \{1, \ldots, m\}$, $p_n \le p_{n,h}$ or $p_{n,h} = 0$.
  – For $n, n' \in \{1, \ldots, m\}$, if $n \neq n'$, then $p_{n,h} = p_{n',h} = 0$ or $p_{n,h} \neq p_{n',h}$.
  – For every $n \in \{1, \ldots, m\}$, if $p_{n,h} \neq p_n$ and $p_{n,h} > 0$, then $p_{n,h} \notin P$.

(2) There exist $\beta_n^\sharp \in \mathbb{B}^k$ for each $n \in \{1, \ldots, m\}$ such that:

  – For every $n \in \{1, \ldots, m\}$, if $\widehat{j}_{n,h} \neq 0$, then $b_{n,h} = \beta_n^\sharp(\widehat{j}_{n,h})$ when $p_{n,h} > p_n$ and $b_{n,h} = \beta_n(\widehat{j}_{n,h})$ otherwise, and if $\widehat{j}_{n,h} = 0$, then $b_{n,h} = 0$.
  – For every $n \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, k\}$, it holds that
$$\beta'_n(j) = \begin{cases} b'_{n,h} & \text{if } \widehat{j}_{n,h} \neq 0,\ j = \widehat{j}_{n,h} \text{ and } p_{n,h} > 0,\\ \beta_n^\sharp(j) & \text{otherwise if } p_{n,h} \neq p_n \text{ and } p_{n,h} > 0,\\ \beta_n(j) & \text{otherwise.}\end{cases}$$
  – $A' = (A \backslash \{(i, \beta) \mid (i-1, \beta) \in G\}) \cup \{(p_{n,h}, \beta_n^\sharp) \mid\ p_{n,h} \neq p_n,\ p_{n,h} > 0\}$,
  – $G' = (G \backslash \{(i, \beta) \mid (i+1, \beta) \in A\}) \cup \{(p_n, \beta_n) \mid p_{n,h} \neq p_n\}$.

  – For $\widetilde{\sigma} \in \{\vdash, \dashv\}$ we have $(\widetilde{q}, \widetilde{\sigma}, \widetilde{q}', d) \in \widetilde{\delta_{\mathsf{c}}}$ iff
    – $P' = P$, $A' = A$, $G' = G$, $p'_n = p_n$ and $\beta'_n = \beta_n$ for $n = 1, \ldots, m$;
    – if $\widetilde{\sigma} = \vdash$ and $h = 1$, then $d = 1$ and $h' = h$;
    – if $\widetilde{\sigma} = \vdash$, $h \neq 1$ and $h < \widetilde{r}$, then $d = 1$ and $h' = h + 1$;
    – if $\widetilde{\sigma} = \vdash$, $h \neq 1$ and $h = \widetilde{r}$, then $d = -1$ and $h' = h$;
    – if $\widetilde{\sigma} = \dashv$ and $h = \widetilde{r}$, then $d = 1$ and $h' = h$;
    – if $\widetilde{\sigma} = \dashv$ and $h < \widetilde{r}$, then $d = -1$ and $h' = h + 1$.
  – $\widetilde{F_{\mathsf{c}}} = \{(\widetilde{r}, p_1, \ldots, p_m, P, \beta_1, \ldots, \beta_m, \emptyset, G) \mid \forall i \in P. \forall 0 < j < i.\ j \in P\}$.
  In an accepting state the set of assumptions should be empty and the set P of block numbers in $\widetilde{\tau}$ should contain all block numbers smaller or equal the maximal one.

By construction, in each word $\widetilde{\tau} \in \mathcal{L}(\widetilde{\mathcal{A}_{\mathsf{c}}})$ each block number is assigned to at most one machine and for each machine the sequence of positive block numbers is nondecreasing.

All such words $\widetilde{\tau}$ are accepted by $\widetilde{\mathcal{A}_{\mathsf{c}}}$ iff the read–write sequence, constructed by ordering elements of $\widetilde{\tau}$ according to block number while preserving the order for each individual machine, is valid w.r.t. the initial register contents.

Proposition 7 *Consider a word $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}^*$. Fix $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $i \in \{1, \ldots, \widetilde{f}\}$ Consider $\widetilde{\eta}_{n,h,i} = (p_{n,h,i}, b_{n,h,i}, b'_{n,h,i}, q'_{n,h,i}, t_{n,h,i})$. If $\widetilde{\sigma}_{t_{n,h,i}} = (\sigma, l, j_1, j_2) \in \Sigma \times \{1, \ldots t\} \times \{0, \ldots, k\}^2$, then we let $j_{n,h,i} = j_1$ if h is odd and $j_{n,h,i} = j_2$ if h is even. In the other cases we let $j_{n,h,i} = 0$.*

Now, let $\widehat{\eta}_{n,h,i} = (p_{n,h,i}, j_{n,h,i}, b_{n,h,i}, h_{n,h,i})$. Then we define $\widehat{\eta}_{n,h} = \widehat{\eta}_{n,h,1}, \ldots, \widehat{\eta}_{n,h,f}$ if h is odd and $\widehat{\eta}_{n,h} = \widehat{\eta}_{n,h,f}, \ldots, \widehat{\eta}_{n,h,1}$ if h is even.

Let us also define $\widehat{\eta}_n = \left(\widetilde{\eta}_{n,1} \cdot \cdots \cdot \eta_{n,\widetilde{r}}\right)|_{\left(\{1,\ldots,u\} \times \{0,\ldots,k\} \times \mathbb{B}^2\right)}$.

The word $\widetilde{\tau}$ is accepted by $\widetilde{\mathcal{A}_{\mathsf{c}}}$ iff it satisfies the two conditions below.

(1) Let $u = \max\left\{p_{n,h,l} \mid n \in \{1, \ldots, m\}, h \in \{1, \ldots, \widetilde{r}\}, l \in \{1, \ldots, f\}\right\}$.

- *For every $1 \leq v \leq u$ there exists exactly one $n \in \{1, \ldots, m\}$ such that there exist $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$ with the property $p_{n,h,l} = v$.*
- *For all $n \in \{1, \ldots, m\}$ and $0 \leq l_1 \leq l_2 < |\widehat{\eta}_n|$ where $\widehat{\eta}_n[l_1] = (p_1, j_1, b_1, b'_1)$ and $\widehat{\eta}_n[l_2] = (p_2, j_2, b_2, b'_2)$ it holds that $p_1 \leq p_2$.*

(2) Let $\widehat{\eta}$ be the unique permutation of $\widehat{\eta}_1 \cdot \cdots \cdot \widehat{\eta}_m$ that is such that

- *the elements of each $\widehat{\eta}_n$ appear in $\widehat{\eta}$ in the same order as in $\widehat{\eta}_n$,*
- *for each $0 \leq l_1 \leq l_2 < |\widehat{\eta}|$, where $\widehat{\eta}[l_1] = (p_1, j_1, b_1, b'_1)$ and $\widehat{\eta}[l_2] = (p_2, j_2, b_2, b'_2)$ it holds that $p_1 \leq p_2$.*

Then, the sequence $\eta = \widehat{\eta}|_{(\{0,\ldots,k\} \times \mathbb{B}^2)}$ is a read–write sequence that is valid w.r.t. initial valuation $\beta_0 : \{1, \ldots, k\} \to \mathbb{B}$, where $\beta_0(j) = 0$ for each $j \in \{1, \ldots, k\}$.

4.6 NFA checking for final configurations

Given a set of final configurations F defined by a multiset S_{final}, we construct an NFA $\mathcal{A}_{\mathsf{s}}$ that checks that some run in $T(\mathcal{M}, m, \mathcal{G}^k)$, corresponding to the input word $\mathcal{A}_{\mathsf{s}}$ reads, reaches a configuration in F.

Since for a configuration to be final we require that there exists a node in the graph with a certain property, potential such configurations can be detected by inspecting (at most) two consecutive letters in the word. The information relevant for determining if a configuration is final consists of the block number and successor state components of the letters of the execution word and the letters of the trace word. Thus, we define the set $C = \{1, \ldots, p\}^t \times (Q \cup \{\perp\})^t \times \Sigma_{\mathsf{t}}$ and consider pairs of elements of C. Let us first define the following two elements of C:

$$c_0 = (p^0_{1,1}, \ldots, p^0_{m,\widetilde{r}}, q^0_{1,1}, \ldots, q^0_{m,\widetilde{r}}, \widetilde{\sigma}^0_{1,1}, \ldots, \widetilde{\sigma}^0_{m,\widetilde{r}}),$$
$$c_\perp = (p^\perp_{1,1}, \ldots, p^\perp_{m,\widetilde{r}}, q^\perp_{1,1}, \ldots, q^\perp_{m,\widetilde{r}}, \widetilde{\sigma}^\perp_{1,1}, \ldots, \widetilde{\sigma}^\perp_{m,\widetilde{r}}),$$

where for $n \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, \widetilde{r}\}$, we have $p^0_{n,h} = p^\perp_{n,h} = 0$, $q^0_{n,h} = q^0$, $q^\perp_{n,h} = \perp$, $\widetilde{\sigma}^0_{n,h} = \widetilde{\sigma}^\perp_{n,h} = (\flat, 1, 0, 0)$.

Let us consider two elements of the set C:

$$c' = (p'_{1,1}, \ldots, p'_{m,\widetilde{r}}, q'_{1,1}, \ldots, q'_{m,\widetilde{r}}, \widetilde{\sigma}'_{1,1}, \ldots, \widetilde{\sigma}'_{m,\widetilde{r}}) \in C,$$
$$c'' = (p''_{1,1}, \ldots, p''_{m,\widetilde{r}}, q''_{1,1}, \ldots, q''_{m,\widetilde{r}}, \widetilde{\sigma}''_{1,1}, \ldots, \widetilde{\sigma}''_{m,\widetilde{r}}) \in C.$$

We say that the pair (c', c'') *occurs in* $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}$ iff there exists a sequence of consecutive letters in $\widetilde{\tau}$ such that those of c' and c'' that are not equal to c_0 and $c_\perp$ match these letters of $\widetilde{\tau}$ in the same order. Formally, (c', c'') *occurs in* $\widetilde{\tau}$ iff one of the following conditions is satisfied.

(1) $c' = c_0$, and $p''_{n,h} = p_{n,h,1}$, $q''_{n,h} = q_{n,h,1}$ and $\widetilde{\sigma}''_{n,h} = \widetilde{\sigma}_{t_{n,h,1},1}$ (c'' matches $\widetilde{\sigma}_1$).
(2) $c'' = c_\perp$, and $p'_{n,h} = p_{n,h,1}$, $q'_{n,h} = q_{n,h,1}$ and $\widetilde{\sigma}'_{n,h} = \widetilde{\sigma}_{t_{n,h,1},f}$ (c' matches $\widetilde{\sigma}_f$).
(3) There exists $1 < l \leq f$ such that

- $p'_{n,h} = p_{n,h,l-1}$, $q'_{n,h} = q_{n,h,l-1}$ and $\widetilde{\sigma}'_{n,h} = \widetilde{\sigma}_{t_{n,h,l-1},l-1}$ (c' matches $\widetilde{\sigma}_{l-1}$),
- $p''_{n,h} = p_{n,h,l}$, $q''_{n,h} = q_{n,h,l}$ and $\widetilde{\sigma}''_{n,h} = \widetilde{\sigma}_{t_{n,h,l},l}$ (c'' matches $\widetilde{\sigma}_l$).

Consider a configuration $\gamma \in \Gamma$ of a run ρ that is in the set of final configurations F. This means that there exists an edge $e \in E$, such that some of the nodes $src(e)$ and $trg(e)$ meets the condition for a final configuration. Furthermore, there exits a set of machines involved in the satisfaction of this condition in γ. Among these machines, we distinguish between the one that executed the last transition in ρ leading to this configuration and the remaining machines. By the definition of runs of $T(\mathcal{M}, m, \mathcal{G}^k)$, the current node and states of these remaining machines should be reached at the end of one of their execution blocks. We define a predicate about pairs of elements of C, sets of machines and corresponding positions in their executions (i.e., rows in the respective letter of the execution word). The automaton $\mathcal{A}_s$ will use this predicate to identify letters of the word that may encode final configurations.

Let $S \in \mathbb{M}(Q)$, $M \subseteq \{1, \ldots, m\}$ and $f_M : M \to \{1, \ldots, \widetilde{r}\}$. For each $n \in M$, let $f_n = f_M(n)$ and if f_n is odd, then $p_n = p'_{n,f_n}$, $q_n = q'_{n,f_n}$ and $\sigma_n = \widetilde{\sigma}'_{n,f_n}$, and if f_n is even, then $p_n = p''_{n,f_n}$, $q_n = q''_{n,f_n}$ and $\sigma_n = \widetilde{\sigma}''_{n,f_n}$. Let $n_0 \in M$ be such that for each $n \in M$, it holds that $p_n \leq p_{n_0}$. We define $p_n(c', c'', M, f_M) = p_n$ for each $n \in M$ and $n_0(c', c'', M, f_M) = n_0$.

The *node predicate* NodeProperty(S, c', c'', M, f_M) is true iff the conditions listed below hold.

- $S = [q_n \mid n \in M]$ and for each $n \in M \backslash \{n_0\}$, $p_n < p_{n_0}$ and $p'_{n,f_n} \neq p''_{n,f_n}$.
- One of the following requirements is satisfied:
  - $\sigma'_{n_0} \in \{1, \ldots, t\}$ and $\sigma'_n = \sigma'_{n_0}$ for each $n \in M$.
  - $\sigma'_{n_0} = (\sigma_0, l_0, j_0, j'_0) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$ and for each $n \in M$ there exist $\sigma \in \Sigma$, $j, j' \in \{0, \ldots, k\}$ such that $\sigma'_n = (\sigma, l_0, j, j')$.
  - $\sigma''_{n_0} \in \{1, \ldots, t\}$ and $\sigma''_n = \sigma''_{n_0}$ for each $n \in M$.
  - $\sigma''_{n_0} = (\sigma_0, l_0, j_0, j'_0) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$ and for each $n \in M$ there exist $\sigma \in \Sigma$, $j, j' \in \{0, \ldots, k\}$ such that $\sigma''_n = (\sigma, l_0, j, j')$.

In order to evaluate NodeProperty on positions of the input word $\widetilde{\tau}$, the NFA $\mathcal{A}_\mathsf{s}$ stores in its state an element c' of C. The current letter of the input word determines an element c'' of C, and $\mathcal{A}_\mathsf{s}$ evaluates the node predicate on the pair (c', c''). In order to verify that some run corresponding to $\widetilde{\tau}$ contains a final configuration in F, $\mathcal{A}_\mathsf{s}$ must detect all pairs on which the predicate NodeProperty holds true, that is, they might encode a final configuration. Then, $\mathcal{A}_\mathsf{s}$ must verify that some of the pairs on which the predicate NodeProperty holds actually fulfils a global condition on $\widetilde{\tau}$. Namely, that for all the involved machines, the currently executed block number is the last one. (We can, w.l.o.g. restrict to runs where only the last configuration can be final.) This guarantees that all involved machines are simultaneously located at the respective node of the graph, in their respective local states, which in turn means that the configuration is in F.

Given the set of final configurations F defined by a multiset S_{final}, we define the NFA $\mathcal{A}_{\mathsf{s}} = (\widetilde{Q}_{\mathsf{s}}, \{\widetilde{q}_{\mathsf{s}}^0\}, \widetilde{\Sigma} \dot{\cup} \{\vdash, \dashv\}, \widetilde{\delta}_{\mathsf{s}}, \widetilde{F}_{\mathsf{s}})$ as follows.

- $\widetilde{Q}_{\mathsf{s}} = \widetilde{Q}_{\mathsf{s}}{}' \cup \widetilde{F}_{\mathsf{s}}$ and $\widetilde{q}_{\mathsf{s}}^0 = (c_0, \emptyset, \ldots, \emptyset, \emptyset, \ldots, \emptyset, p, \ldots, p, 0)$, where $\widetilde{F}_{\mathsf{s}} = \{\widetilde{q}_{\mathsf{s}}^f\}$ and

$$\widetilde{Q}_{\mathsf{s}}{}' = C \times \left(2^{\{1,\ldots,p\}}\right)^m \times \{0, \ldots, p\}^m \times \mathbb{B}.$$

  A state $\widetilde{q} = (c', P_1, \ldots, P_m, p_1, \ldots, p_m, final)$ of $\mathcal{A}_{\mathsf{s}}$ contains an element c' of C, a Boolean value $final \in \mathbb{B}$, and for each machine n it contains components $P_n \in 2^{\{1,\ldots,p\}}$ and $p_n \in \{0, \ldots, p\}$. The set P_n consists of the already seen block numbers for machine n.

- Let $\widetilde{\sigma} = (\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_t, \widetilde{\eta}_{1,1}, \ldots, \widetilde{\eta}_{1,\widetilde{r}}, \ldots, \widetilde{\eta}_{m,1}, \ldots, \widetilde{\eta}_{m,\widetilde{r}}) \in \widetilde{\Sigma}$, where for all $n \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, \widetilde{r}\}$ we have $\widetilde{\eta}_{n,h} = (p_{n,h}, b_{n,h}, b'_{n,h}, q'_{n,h}, t_{n,h})$. Let

$$\begin{aligned}
\widetilde{q} &= (c', P_1, \ldots, P_m, p_1, \ldots, p_m, final)\\
\widetilde{q}' &= (\ p'''_{1,1}, \ldots, p'''_{m,\widetilde{r}}, q'''_{1,1}, \ldots, q'''_{m,\widetilde{r}}, \widetilde{\sigma}'''_{1,1}, \ldots, \widetilde{\sigma}'''_{m,\widetilde{r}},\\
&\qquad p''''_{1,1}, \ldots, p''''_{m,\widetilde{r}}, q''''_{1,1}, \ldots, q''''_{m,\widetilde{r}}, \widetilde{\sigma}''''_{1,1}, \ldots, \widetilde{\sigma}''''_{m,\widetilde{r}},\\
&\qquad P'_1, \ldots, P'_m, p'_1, \ldots, p'_m, final'),\\
c' &= (p'_{1,1}, \ldots, p'_{m,\widetilde{r}}, q'_{1,1}, \ldots, q'_{m,\widetilde{r}}, \widetilde{\sigma}'_{1,1}, \ldots, \widetilde{\sigma}'_{m,\widetilde{r}}), \text{ and}\\
c &= (p_{1,1}, \ldots, p_{m,\widetilde{r}}, q_{1,1}, \ldots, q_{m,\widetilde{r}}, \widetilde{\sigma}_{t_{1,1}}, \ldots, \widetilde{\sigma}_{t_{m,\widetilde{r}}}).
\end{aligned}$$

  Then, $(\widetilde{q}, \widetilde{\sigma}, \widetilde{q}', d) \in \widetilde{\delta}_{\mathsf{s}}$ iff $d = 1$ and the following conditions are satisfied.

  (1) For each $n \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, \widetilde{r}\}$, $p_{n,h} \leq p_n$, $\widetilde{\sigma}'''_{n,h} = \widetilde{\sigma}''_{n,h}$, $\widetilde{\sigma}''''_{n,h} = \widetilde{\sigma}_{t_{n,h}}$, and if $p_{n,h} > 0$ and $\widetilde{\sigma}_{t_{n,h}} \in \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$, then $p'''_{n,h} = p''_{n,h}$, $q'''_{n,h} = q''_{n,h}$, $p''''_{n,h} = p_{n,h}$ and $q''''_{n,h} = q_{n,h}$.
  Based on the currently read letter of $\widetilde{\tau}$, the transition relation $\widetilde{\delta}_{\mathsf{s}}$ updates the component c' of the state.

  (2) For each $n \in \{1, \ldots, m\}$
  - $P'_n = P_n \cup \{p_{n,h} \mid h \in \{1, \ldots, \widetilde{r}\}, p_{n,h} > 0\}$ and
  - if $final' = final$, then $p'_n = p_n$.

  (3) If $final' \neq final$, then $final = 0$ and there exist $M \subseteq \{1, \ldots, m\}$ and a function $f_M : M \to \{1, \ldots, \widetilde{r}\}$, such that:

  - the predicate $\mathsf{NodeProperty}(S_{final}, c', c, M, f_M)$ holds true,
  - for every $n' \in M$, $p'_{n'} = p_{n'}(c', c, M, f_M)$,
  - for every $n' \in \{1, \ldots, m\} \backslash M$ it holds that $p'_{n'} = p_{n'}$.

  If the current letter defines $c \in C$ such that $\mathsf{NodeProperty}(S_{final}, c', c, M, f_M)$ holds true for some $M \subseteq \{1, \ldots, m\}$ and some function f_M, then $final$ can be set to 1 if it is 0, and the current block number of machine n for each $n \in M$ can be stored in p_n, in order to verify later that a final configuration is indeed reached (by checking that p_n is the maximal block number for machine n).

- $(\widetilde{q}, \vdash, \widetilde{q}', d) \in \widetilde{\delta}_{\mathsf{s}}$ iff $\widetilde{q} = \widetilde{q}'$ and $d = 1$;
- Let $\widetilde{q}' \in \widetilde{Q}_{\mathsf{s}}$ and $\widetilde{q} = (c', P_1, \ldots, P_m, p_1, \ldots, p_m, final)$, where $c' = (p'_{1,1}, \ldots, p'_{m,\widetilde{r}}, q'_{1,1}, \ldots, q'_{m,\widetilde{r}}, \widetilde{\sigma}'_{1,1}, \ldots, \widetilde{\sigma}'_{m,\widetilde{r}})$.
  Then, $(\widetilde{q}, \dashv, \widetilde{q}', d) \in \widetilde{\delta}_{\mathsf{s}}$ iff $d = 1$ and it holds that $\widetilde{q}' = \widetilde{q}_{\mathsf{s}}^f$ iff we have that:

- if $final = 1$, then for each $n \in \{1, \ldots, m\}$, $p_n \geq \max P_n$;
- if $final = 0$, then for some $M \subseteq \{1, \ldots, m\}$ and function f_M we have:
  - $\mathsf{NodeProperty}(S, c', c_\perp, M, f_M)$ holds true and
  - for every $n \in M$, $p_n(c', c_\perp, M, f_M) \geq \max P_n$.

  The accepting state $\widetilde{q}_{\mathsf{s}}^{f}$ can be entered after reading $\dashv$ if $final = 1$ and if p_n is the maximal block number for machine n for each n.

The construction of $\mathcal{A}_{\mathsf{s}}$ ensures that $\widetilde{\tau} \in \mathcal{L}(\mathcal{A}_{\mathsf{s}})$ iff there exists a pair (c', c'') occurring in $\widetilde{\tau}$ encoding a configuration that is in F.

Proposition 8 *Consider a word $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}^*$ and let $P_n = \{p_{n,h,l} \mid 1 \leq h \leq \widetilde{r}, 1 \leq l \leq f\}$. The word $\widetilde{\tau}$ is accepted by $\mathcal{A}_{\mathsf{s}}$ iff there exists a tuple (c', c'') occurring in $\widetilde{\tau}$, set $M \subseteq \{1, \ldots, m\}$ and a function $f_M : M \to \{1, \ldots, \widetilde{r}\}$ which satisfy the predicate $\mathsf{NodeProperty}(S_{final}, A, c', c'', M, f_M)$ and are such that for every $n \in M$ it holds that $p_n(c', c'', M, f_M) = \max P_n$.*

4.7 Correctness of the algorithm

Let $\mathcal{A}_1, \ldots, \mathcal{A}_m$ be an NFA obtained respectively from $\widetilde{\mathcal{A}}_1, \ldots, \widetilde{\mathcal{A}}_m$ such that for each $i \in \{1, \ldots, m\}$, $\mathcal{L}(\mathcal{A}_i) = \mathcal{L}(\widetilde{\mathcal{A}}_i)$. Let $\mathcal{A}_{\mathsf{c}}$ be an NFA constructed from $\widetilde{\mathcal{A}}_{\mathsf{c}}$ such that $\mathcal{L}(\mathcal{A}_{\mathsf{c}}) = \mathcal{L}(\widetilde{\mathcal{A}}_{\mathsf{c}})$. We then construct the NFA $\mathcal{A}_{\mathsf{e}}$ by intersecting $\mathcal{A}_1, \ldots, \mathcal{A}_m, \mathcal{A}_{\mathsf{c}}$ and $\mathcal{A}_{\mathsf{s}}$ and projecting the result on Σ_{t}, i.e., $\mathcal{L}(\mathcal{A}_{\mathsf{e}}) = \left(\bigcap_{i=1}^{m} \mathcal{L}(\mathcal{A}_i) \cap \mathcal{L}(\mathcal{A}_{\mathsf{c}}) \cap \mathcal{L}(\mathcal{A}_{\mathsf{s}})\right)|_{\Sigma_{\mathsf{t}}}$. The PDA $\mathcal{A}$ is the intersection of $\mathcal{P}_{\mathsf{t}}$ and $\mathcal{A}_{\mathsf{e}}$.

Theorem 2 *$\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exists an r-reversal bounded run $\rho = \gamma_0 \cdots \gamma_f$ in $T(\mathcal{M}, m, \mathcal{G}^k)$ such that $\gamma_i \in F$ for some $0 \leq i \leq f$, i.e., a run that reaches a configuration in F.*

Proof Suppose that $\mathcal{L}(\mathcal{A}) \neq \emptyset$. We show that there exists an r-reversal bounded run ρ that reaches a configuration in F.

Let us fix $\widetilde{\tau}_{\mathsf{t}} \in \mathcal{L}(\mathcal{A})$. By the construction of $\mathcal{A}$, the word $\widetilde{\tau}_{\mathsf{t}}$ is accepted by $\mathcal{P}$ and $\mathcal{A}_{\mathsf{r}}$, and there exists a word $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f}$ accepted by $\widetilde{\mathcal{A}}_{\mathsf{c}}, \widetilde{\mathcal{A}}_1, \ldots, \widetilde{\mathcal{A}}_m$ and $\widetilde{\mathcal{A}}_{\mathsf{s}}$ such that for every $l \in \{1, \ldots, f\}$ if $\widetilde{\sigma}_l = (\widetilde{\sigma}_{1,l}, \ldots, \widetilde{\sigma}_{t,l}, \widetilde{\eta}_{1,1,l}, \ldots, \widetilde{\eta}_{1,\widetilde{r},l}, \ldots, \widetilde{\eta}_{m,1,l}, \ldots, \widetilde{\eta}_{m,\widetilde{r},l})$ is the lth letter of $\widetilde{\tau}$, then $(\widetilde{\sigma}_{1,l}, \ldots, \widetilde{\sigma}_{t,l})$ is the lth letter of $\widetilde{\tau}_{\mathsf{t}}$. Let for $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$, $\widetilde{\eta}_{n,h,l} = (p_{n,h,l}, b_{n,h,l}, b'_{n,h,l}, q'_{n,h,l}, t_{n,h,l})$.

According to Proposition 4, there exists $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ where $G = (N, E, n_b, n_e)$, and there exist sequences of nodes $\pi_i = n_{i,0}, \ldots, n_{i,f_i} \in N^*$ for each $i \in \{1, \ldots, t\}$, that satisfy the conditions listed in the statement of the proposition.

Since the graph G contains at most k registers positioned at nodes marked by a mapping κ, we can assign to each of those nodes unique identifiers from the set $\{1, \ldots, k\}$ and assign 0 to all other nodes. Furthermore, this assignment can be made consistent with the annotation in the word $\widetilde{\tau}$ according to the correspondence established in Proposition 4. Let us denote with $\widehat{\kappa}(n)$ the identifier assigned to node n in this way.

For each $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$, if $\widetilde{\sigma}_{t_{n,h,l},l} \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$ and $p_{n,h,l} > 0$, then we define $\widehat{\pi}_{n,h,l} = (p_{n,h,l}, n_{t_{n,h,l+1}})$ and $\widehat{\zeta}_{n,h,l} = (p_{n,h,l}, n, q'_{n,h,l}, \widehat{\kappa}(n_{t_{n,h,l+1}}), b'_{n,h,l})$, and otherwise, $\widehat{\pi}_{n,h,l} = \epsilon$ and $\widehat{\zeta}_{n,h,l} = \epsilon$.

$\widehat{\pi}_{n,h} = \widehat{\pi}_{n,h,1} \cdot \cdots \cdot \widehat{\pi}_{n,h,f}$ if h is odd, and $\widehat{\pi}_{n,h} = \widehat{\pi}_{n,h,f} \cdot \cdots \cdot \widehat{\pi}_{n,h,1}$ otherwise, $\widehat{\zeta}_{n,h} = \widehat{\zeta}_{n,h,1} \cdot \cdots \cdot \widehat{\zeta}_{n,h,f}$ if h is odd, and $\widehat{\zeta}_{n,h} = \widehat{\zeta}_{n,h,f} \cdot \cdots \cdot \widehat{\zeta}_{n,h,1}$ otherwise.

Let $\widehat{\pi}$ be the unique permutation of the sequence $\widehat{\pi}_{n,1} \cdot \cdots \cdot \widehat{\pi}_{n,f}$ such that the elements of each $\widehat{\pi}_{n,h}$ appear in $\widehat{\pi}$ in the same order as in $\widehat{\pi}_{n,h}$, and if $0 \leq l_1 \leq l_2 < |\widehat{\pi}|, \widehat{\pi}[l_1] = (p_1, n_1)$ and $\widehat{\pi}[l_2] = (p_2, n_2)$, then $p_1 \leq p_2$.

Let $\widehat{\zeta}$ be the unique permutation of the sequence $\widehat{\zeta}_{n,1} \cdot \cdots \cdot \widehat{\zeta}_{n,f}$ such that the elements of each $\widehat{\zeta}_{n,h}$ appear in $\widehat{\zeta}$ in the same order as in $\widehat{\zeta}_{n,h}$, and if $0 \leq l_1 \leq l_2 < |\widehat{\zeta}|$, $\widehat{\zeta}[l_1] = (p_1, m_1, q_1, j_1, b_1)$ and $\widehat{\zeta}[l_2] = (p_2, m_2, q_2, j_2, b_2)$, then $p_1 \leq p_2$.

By definition $|\widehat{\zeta}| = |\widehat{\pi}|$. Let $|\widehat{\zeta}| = |\widehat{\pi}| = s$, $\widehat{\pi} = (p_1, n_1), \ldots, (p_s, n_s)$ and $\widehat{\zeta} = (p_1, m_1, q_1, j_1, b'_1), \ldots, (p_s, m_s, q_s, j_s, b'_s)$.

We construct a sequence of configurations $\rho = \gamma_0, \ldots, \gamma_s \in \Gamma$ as follows. Let $\gamma_0 = \langle G, K, \kappa, \mu_0, \beta_0 \rangle$, where $\mu_0(n_b) = \{(i, q^0) \mid i \in \{1, \ldots, m\}\}$, $\mu_0(n) = \emptyset$ for every $n \in N \backslash \{n_b\}$, and $\beta_0(j) = 0$ for every $j \in \{1, \ldots, k\}$. For each $1 \leq i \leq s$, if $\widehat{\pi}[i] = (p_i, n_i)$ and $\widehat{\zeta}[i] = (p'_i, m_i, q_i, j_i, b'_i)$, then $p_i = p'_i$ and we define:

- $\mu_i(n) = \mu_{i-1}(n) \backslash \{(m_i, q)\}$ and $\mu_i(n_i) = \mu_{i-1}(n_i) \cup \{(m_i, q_i)\}$,
- $\mu_i(n') = \mu_{i-1}(n')$ for each $n' \in N \backslash \{n, n_i\}$,
- if $j_i > 0$, then $\beta_i(j_i) = b'_i$, $\beta_i(j) = \beta_{i-1}(j)$ for each $j \in \{1, \ldots, k\} \backslash \{j_i\}$,

where $n \in N$ and $q \in Q$ are the unique node and state such that $(m_i, q) \in \mu_{i-1}(n)$.

Propositions 6 and 7 imply that ρ is an r-reversal bounded run. Propositions 4, 6–8 imply that for each $0 \leq i \leq s$ it holds that $\gamma_s \in F$. This completes this direction of the proof.

For the other direction, suppose that there exists an r-reversal bounded run $\rho = \gamma_0 \gamma_1 \cdots \gamma_s$ such that $\gamma_s \in F$. We will show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

Let $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$, where $G = (N, E, n_b, n_e)$, be the corresponding graph. Since the graph G contains at most k registers positioned at nodes marked by a mapping κ, we can assign to each of those nodes unique identifiers from the set $\{1, \ldots, k\}$ and assign 0 to all other nodes.

Let $\gamma_i = \langle G, \mu_i, \beta_i \rangle$ for $0 \leq i \leq s$. According to Proposition 2 we can w.l.o.g. assume that ρ consists of u blocks, where $0 \leq u \leq p$. We number the blocks in ρ sequentially with $1, \ldots, u$.

Let us fix a trace τ compatible with ρ. For each $n \in \{1, \ldots, m\}$ we denote with τ_n the trace obtained from τ by taking the subsequences corresponding to the transitions of machine $\mathcal{M}_n$. Since ρ is r-reversal bounded, according to Proposition 3, given ρ and τ, there exist for each $n \in \{1, \ldots, m\}$ and each $h \in \{1, \ldots, \widetilde{r}\}$ a trace $\widehat{\tau}_{n,h} \in \Sigma^*$ and a path $\widehat{\pi}_{n,h} \in \mathsf{Paths}(G, \widehat{\tau}_{n,h})$ that satisfy the properties in the proposition. Let us fix for each $n \in \{1, \ldots, m\}$ and each $1 \leq h \leq r_n$, where r_n is the number of reversals done by machine n in ρ, the traces $\tau'_{n,h}, \tau''_{n,h}, \tau'''_{n,h}$ and the paths $\pi'_{n,h}, \pi''_{n,h}, \pi'''_{n,h}$ as in Proposition 3.

According to Proposition 5 there exists a bijection $g : \{1, \ldots, m\} \times \{1, \ldots, \widetilde{r}\} \to \{1, \ldots, t\}$ such that if for each $i \in \{1, \ldots, t\}$ we define $\tau_i = \widehat{\tau}_{g^{-1}(i)}$ and $\pi_i = \widehat{\pi}_{g^{-1}(i)}$, then there exists a word $\widetilde{\tau}_\mathsf{t} \in \mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\mathcal{A}_\mathsf{r})$ which satisfies the conditions of this proposition. Let $(\widetilde{\sigma}_{1,l}, \ldots, \widetilde{\sigma}_{t,l})$ be the lth letter of $\widetilde{\tau}_\mathsf{t}$ for $l \in \{1, \ldots, f\}$ and let $\widetilde{\tau}_i = \widetilde{\sigma}_{i,1}, \ldots, \widetilde{\sigma}_{i,f}$ be the ith row of $\widetilde{\tau}_\mathsf{t}$ for $i \in \{1, \ldots, t\}$.

Now, we define a word $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f}$, where for $1 \leq l \leq f$

$$\widetilde{\sigma}_l = (\widetilde{\sigma}_{1,l}, \ldots, \widetilde{\sigma}_{t,l}, \widetilde{\eta}_{1,1,l}, \ldots, \widetilde{\eta}_{1,\widetilde{r},l}, \ldots, \widetilde{\eta}_{m,1,l}, \ldots, \widetilde{\eta}_{m,\widetilde{r},l}),$$

for $i \in \{1, \ldots, t\}$ the ith row of $\widetilde{\tau}$ is $\widetilde{\tau}_i = \widetilde{\sigma}_{i,1} \cdots \widetilde{\sigma}_{i,f}$, and for $n \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, \widetilde{r}\}$,
$\widetilde{\eta}_{n,h,l} = (p_{n,h,l}, b_{n,h,l}, b'_{n,h,l}, q'_{n,h,l}, t_{n,h,l})$.

For each $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$:

- if $h \leq r_n + 1$ and $\widetilde{\sigma}_{g(n,h),l}$ corresponds to a position in $\tau''_{n,h}$, then:
  - the value $p_{n,h,l}$ is the block number of the corresponding transition in ρ,
  - the values $b_{n,h,l}$ and $b'_{n,h,l}$ are obtained from the values of the registers in the corresponding nodes of the graph in the corresponding configurations of the run ρ;

  - the value $q'_{n,h,l}$ is the target state of the corresponding transition in ρ;

- if $h \leq r_n + 1$ and $\widetilde{\sigma}_{g(n,h),l}$ does not correspond to a position in $\tau_{g(n,h)}$ ($\widetilde{\sigma}_{g(n,h),l} \in \{\flat\} \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2 \cup \{1, \ldots, t\} \cup \{1, \ldots, t\}$), then:
  - if h is odd and $l > 0$, then $p_{n,h,l} = p_{n,h,l-1}$,
  - if h is odd, $l = 0$, $h > 1$ then $p_{n,h,l} = p_{n,h-1,0}$,
  - if h is odd, $l = 0$, $h = 1$ then $p_{n,h,l} = 0$,
  - if h is even and $l < f$, then $p_{n,h,l} = p_{n,h,l+1}$,
  - if h is even and $l = f$, then $p_{n,h,l} = p_{n,h-1,f}$,
  - $b_{n,h,l} = b'_{n,h,l} = 0$,
  - $q'_{n,h,l}$ is an arbitrarily fixed element of the set Q;
- if $h > r_n + 1$ or $\widetilde{\sigma}_{g(n,h),l}$ corresponds to a position in $\tau'_{n,h}$ or $\tau'''_{n,h}$, then $p_{n,h,l} = 0$, $b_{n,h,l} = b'_{n,h,l} = 0$, and fix arbitrary $q'_{n,h,l}$.

For each $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$, let $t_{n,h,l} = g(n, h)$.

By construction, the word $\widetilde{\tau}$ fulfills for every $n \in \{1, \ldots, m\}$ the conditions of Proposition 6 and is thus accepted by the automata $\widetilde{\mathcal{A}}_1, \ldots, \widetilde{\mathcal{A}}_m$. Also by construction, $\widetilde{\tau}$ satisfies the conditions of Proposition 7 and is thus accepted by the automaton $\widetilde{\mathcal{A}_c}$. Since $\gamma_s \in F$, the construction of $\widetilde{\tau}_t$ and $\widetilde{\tau}$ ensure that $\widetilde{\tau}$ satisfies the conditions of Proposition 8, and is hence accepted by $\widetilde{\mathcal{A}}_s$. Therefore $\widetilde{\tau}_t \in \mathcal{L}(\mathcal{A})$, and hence $\mathcal{L}(\mathcal{A}) \neq \emptyset$. □

4.8 Complexity

We establish an **EXPTIME** lower bound for the reversal- and register-bounded reachability problem by reduction from the intersection nonemptiness problem for one PDA and several deterministic finite automata, which is well-known to be **EXPTIME**-complete [15].[1]

Proposition 9 *The reversal- and register-bounded reachability problem for graph-grammar transition systems is* **EXPTIME***-hard.*

Proof Let $\mathcal{D}$ be a PDA and $\mathcal{B}_1, \ldots, \mathcal{B}_m$ be m deterministic finite automata over a common alphabet $\widehat{\Sigma}$, with $\mathcal{D} = (Q_\mathcal{D}, q^0_\mathcal{D}, \widehat{\Sigma}, \Delta, \bot, \delta_\mathcal{D})$ and $\mathcal{B}_i = (Q_i, q^0_i, \widehat{\Sigma}, \delta_i, F_i)$ for $i = 1, \ldots, m$. We reduce the intersection nonemptiness of $\mathcal{D}, \mathcal{B}_1, \ldots, \mathcal{B}_m$, that is, the question whether $\mathcal{L}(\mathcal{D}) \cap \mathcal{L}(\mathcal{B}_1) \cap \cdots \cap \mathcal{L}(B_m) \neq \emptyset$ to the 0-reversal m-register bounded reachability problem for a graph-grammar transition system $T(\mathcal{M}, m, \mathcal{G}^m)$ with m machines. The number of states of $\mathcal{M} = (Q, q^0, \Sigma, \delta)$ is $Q = \sum_{i=1}^{n} |Q_i| + 2$, and its alphabet is $\Sigma = \widehat{\Sigma} \cup \{a_1, \ldots, a_m, f\}$, where $a_1, \ldots, a_m, f \notin \widehat{\Sigma}$.

The graph grammar $\mathcal{G}$ generates linear graphs with edges as follows:

$$(n_0, n_1, a_1), \ldots, (n_{m-1}, n_m, a_m), (n_m, \widehat{n}_1, \widehat{\sigma}_1), \ldots (\widehat{n}_{l-1}, \widehat{n}_l, \widehat{\sigma}_l), (\widehat{n}_l, \widehat{n}_{l+1}, f),$$

in which the nodes $n_0, n_1, \ldots, n_{m-1}$ are equipped with registers.

Intuitively, $\mathcal{G}$ is defined such that each graph generated by $\mathcal{G}$ represents a word in $\mathcal{L}(\mathcal{D})$ preceded by the word $a_1 \cdots a_m$ and followed by the single-letter word f. All graphs of this form for words in $\mathcal{L}(\mathcal{D})$ are generated by $\mathcal{G}$.

The SPGG $\mathcal{G}$ has terminal symbols $\widehat{\Sigma} \cup \{a_1, \ldots, a_m, f\}$. It contains two rules for each of the letters $a_1, \ldots, a_m, f$. The rest of the rules of $\mathcal{G}$ encode the transition relation $\delta_\mathcal{D}$ of the PDA $\mathcal{D}$, following the standard construction [16].

[1] This reduction was pointed out to us by Aiswarya Cyriac.

The set $Q = \bigcup_{i=1}^{m} Q_i \cup \{q^0, q_f\}$ of states of $\mathcal{M}$ consists of the union of states of the automata $\mathcal{B}_1, \ldots, \mathcal{B}_m$ together with a dedicated initial state q^0 and dedicated final state q_f. The transition relation δ of $\mathcal{M}$ consists of the following transitions:

- $(q^0, a_{i+1}, 0, q^0_{i+1}, 1, 1)$, for $i = 0, \ldots, m-1$,
- $(q^0, a_{i+1}, 1, q^0, 1, 1)$, for $i = 0, \ldots, m-1$,
- $(q^0_j, a_{i+1}, b, q^0_j, 1, b)$, for $i = 0, \ldots, m-1$ and $b \in \mathbb{B}$,
- $(q, \sigma, 0, q', 1, 0)$, for all $(q, \sigma, q', 1) \in \delta_1 \cup \cdots \cup \delta_m$,
- $(q, f, 0, q_f, 1, 0)$, for all $q \in F_1 \cup \cdots \cup F_m$.

Intuitively, each machine starts in state q^0 and traverses the graph until reaching the first node n_i whose register has value 0, when it transitions to state q^0_{i+1} and updates the value of the register at n_i to 1. The definitions of the transitions in the first three items above guarantee that for each automaton $\mathcal{B}_i$ exactly one of the machines transitions to the initial state q^0_i of $\mathcal{B}_i$. Once this machine reaches node n_m it starts to simulate automaton $\mathcal{B}_i$ on the portion of the linear graph labelled with letters from $\widehat{\Sigma}$. Finally, when the edge labelled with the letter f is reached, if the machine is in state from F_i, it transitions to the end node in the final state q_f.

For the multiset of final states S_{final} we take the multiset consisting of m copies of q_f. By the construction of $\mathcal{M}$ and $\mathcal{G}$ it follows that there exists a run of $T(\mathcal{M}, m, \mathcal{G}^m)$ that reaches S_{final} if and only if there exists a word in $\mathcal{L}(\mathcal{D})$ (the corresponding graph for which is generated by $\mathcal{G}$), that is accepted by all of the automata $\mathcal{B}_1, \ldots, \mathcal{B}_m$ (the machines simulating them all reach the end node in state q_f).

This completes the reduction, and since the sizes of $\mathcal{G}$ and $\mathcal{M}$ are at most polynomial in the sum of sizes of the input automata, the claim follows. □

Our algorithm reduces the reversal- and register-bounded reachability problem to checking non-emptiness of the intersection of word automata: one PDA and several NFA. Some of the nondeterministic automata are given in our construction as two-way automata (NFA), but since two-way NFA and one-way NFA are equally expressive, they can be as well translated to NFA. Moreover, as we explain below, due to the specific structure of the 2NFA in our construction and the fixed number of passes each of them does on the input word, this translation can be done in a more direct matter than the classical conversions from 2NFA to NFA.

Below we give the size of the input alphabets, and the size (number of states) for each of the automata involved in our algorithm. Recall that $t = (r+1) \cdot m$, where m is the number of machines and r is the bound on the number of reversals of each machine, and k is the bound on the number of registers. Recall also that $p = \big(r \cdot m + k \cdot m \cdot (r+1) + 1\big) \cdot (m+1)$ is the derived bound on the number of blocks.

The size of the trace alphabet is $|\widetilde{\Sigma}_{\mathsf{t}}| = \big((|\Sigma| + 1) \cdot (k+1)^2 \cdot t\big)^t + t^t$. The size of the alphabet $\widetilde{\Sigma}$ is $|\widetilde{\Sigma}| = |\widetilde{\Sigma}_{\mathsf{t}}|.\big((p+1) \cdot 4 \cdot |Q| \cdot t\big)^t$.

- The number of states of the PDA $\mathcal{P}_{\mathsf{t}}$ is $|Q_{\mathsf{p}}| = 2$.
- The number of states of the NFA $\mathcal{A}_{\mathsf{r}}$ is

$$|Q_{\mathsf{r}}| = \big(t \cdot (k+1)\big)^t \cdot 2^{(k+1)}.$$

- The number of states of each 2NFA $\widetilde{\mathcal{A}_n}$ is

$$|\widetilde{Q_n}| = |Q| \cdot t \cdot (r+1) \cdot 3.$$

  By the construction of $\widetilde{\mathcal{A}_n}$, its runs make exactly r reversals on the input word, simulating the r reversals of the machine. Thus, $\widetilde{\mathcal{A}_n}$ can be converted to an NFA by guessing at the

beginning of the word states for the passes in the reverse direction and then simulating the machine backwards on the reverse passes on the word, verifying that the guess was correct at the end of the word. The resulting NFA will have $\mathcal{O}(|\widetilde{Q}_n|^{r+1})$ states.

– The number of states of the 2NFA $\widetilde{\mathcal{A}}_{\mathsf{c}}$ is

$$|\widetilde{Q}_{\mathsf{c}}| = (r+1) \cdot p^m \cdot 2^{(p+k\cdot m+2\cdot k\cdot p)}.$$

Similarly to $\widetilde{\mathcal{A}}_n$, the automaton $\widetilde{\mathcal{A}}_{\mathsf{c}}$ also makes $r+1$ passes over the input word and can be converted to an NFA that guesses states from which to simulate $\widetilde{\mathcal{A}}_{\mathsf{c}}$ backwards for the reverse runs, and at the end of the word verifies that these guesses are correct. The resulting NFA will have $\mathcal{O}(|\widetilde{Q}_{\mathsf{c}}|^{r+1})$ states.

– The number of states of the NFA $\mathcal{A}_{\mathsf{s}}$ is

$$|Q_{\mathsf{s}}| = \big(p \cdot (|Q|+1)\big)^t \cdot \Big(\big((|\Sigma|+1)\cdot(k+1)^2 \cdot t\big)^t + t^t\Big) \cdot 2^{p\cdot m} \cdot (p+1)^m \cdot 2 + 1.$$

Thus, the problem can be reduced to checking language emptiness for a product automaton whose size is $\alpha_1(|Q|, |\Sigma|, m, r, k)^{m\cdot(r+1)} \cdot 2^{\alpha_2(m,r,k)}$, where α_1 is a polynomial in $|Q|$, $|\Sigma|$, m, r and k, and α_2 is a polynomial in m, r and k. Language emptiness of the product PDA can be checked in time polynomial in its size.

Thus, the algorithm based on our construction gives us an **EXPTIME** upper bound. Hence we can conclude that the problem is **EXPTIME**-complete.

Theorem 3 *The reversal- and register-bounded reachability problem for graph-grammar transition systems is* **EXPTIME***-complete.*

5 Conclusion

In this paper we define and study a class of concurrent finite-state automata traversing series–parallel graphs and communicating through shared finite registers located at the nodes of the graph. We considered a model in which a *fixed number* of finite-state machines traverse the nodes of a *series–parallel graph*. The series–parallel graphs are generated by a graph grammar, and as we do not impose an a priori bound on the size of the graphs, the resulting system is infinite-state. Since the emptiness problem for this model is in general undecidable, we consider a natural restriction by putting bounds on the number of reversals along the computation and the number of shared registers in the graph. With these two restrictions, we show that the emptiness problem is decidable and can be reduced to PDA emptiness.

As we noted in Sect. 2.4, our decidability result holds for a more general model of communication between the machines, in which either read or write (but not both) operations on registers can be non-local, that is, access a register that is not at the node where the machine is currently located. Another possible extension that we omitted for simplicity concerns the set of final configurations in the reachability problem. While here we consider properties that quantify over individual nodes in the graph, we can, in principle, extend the construction described in Sect. 4.6 to handle properties asserting the existence of edges with certain labels, or a fixed number of adjacent nodes and edges.

Interesting directions for future work include establishing the complexity of the bounded emptiness problem for our model, as well as studying different extensions. One possibility is to allow parametrization in the number of concurrent machines, another is to consider other classes of context-free GTSs. For example, using the techniques from [21] one can try to extend our results to a more general class of graphs of bounded tree width.

Acknowledgements Open access funding provided by Max Planck Society.

Appendix: Proofs omitted in Sect. 4

Proposition 10 *If a word $\widetilde{\tau} = \widetilde{\sigma}_1 \cdots \widetilde{\sigma}_f \in \Sigma_t^*$ is accepted by $\mathcal{P}_t$, then there exists $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ with $G = (N, E, n_b, n_e)$ and for each $i \in \{1, \ldots, t\}$ there exists a sequence of nodes $\widetilde{\pi}_i = \widetilde{n}_{i,0}, \ldots, \widetilde{n}_{i,f}$ in the graph G such that all of the following conditions are satisfied.*

(1) For each $i \in \{1, \ldots, t\}$ it holds that:

- *For each $1 < j \leq f$:*
  - *If $\widetilde{\sigma}_{i,j} = (\sigma, j_1, j_2, l) \in \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$, then there exists $e \in E$ with $src(e) = \widetilde{n}_{i,(j-1)}$, $trg(e) = \widetilde{n}_{i,j}$ and $\alpha(e) = \sigma$,*
  - *If $\widetilde{\sigma}_{i,j} \in \big(\{\flat\} \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}\big) \cup \{1, \ldots, t\}$, then $\widetilde{n}_{i,(j-1)} = \widetilde{n}_{i,j}$.*
- *Let $\tau_i = \big(\widetilde{\tau}_i|_{\Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}}\big)|_\Sigma$ be the projection of $\widetilde{\tau}_i$.*
  Then, there exists a subsequence $\pi_i = n_0, \ldots, n_{f_i}$ of $\widetilde{\pi}_i$ with $\pi_i \in \mathsf{Paths}(G, \tau_i)$.

(2) For all $i_1, i_2 \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, f\}$ it holds that:

- *If $\widetilde{\sigma}_{i_1,j}, \widetilde{\sigma}_{i_2,j} \in \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$, $\widetilde{\sigma}_{i_1,j} = (\sigma, j_1, j_2, l)$ and $\widetilde{\sigma}_{i_2,j} = (\sigma', j_1', j_2', l')$, then $l = l'$ implies $\widetilde{n}_{i_1,(j-1)} = \widetilde{n}_{i_2,(j-1)}$, $\widetilde{n}_{i_1,j} = \widetilde{n}_{i_2,j}$ and and $\sigma = \sigma'$.*
- *If $\widetilde{\sigma}_j \in \{1, \ldots, t\}^t$, then $\widetilde{\sigma}_{i_1,j} = \widetilde{\sigma}_{i_2,j}$ iff $\widetilde{n}_{i_1,j-1} = \widetilde{n}_{i_2,j-1}$, $\widetilde{n}_{i_1,j} = \widetilde{n}_{i_2,j}$.*

(3) For every $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, f\}$ with $\widetilde{\sigma}_{i,j} = (\sigma, j_1, j_2, l) \in \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$ it holds that $j_1 > 0$ iff $\kappa(\widetilde{n}_{i,(j-1)})$ and $j_2 > 0$ iff $\kappa(\widetilde{n}_{i,j})$.

Proof (*Proof sketch*) The automaton $\mathcal{P}$ simulates the derivation by rules of $\mathcal{G}^k$ of a t-tuple of traces in some graph G derived by $\mathcal{G}^k$. On its stack it stores tuples of variables, again one for each of the traces, or terminal letters. For shared parts of the paths corresponding to these traces, a shared copy of the variables is used. New copies are introduced by expanding variables using the parallel composition rules. As the number of symbols in a tuple is bounded by t, $\mathcal{P}$ nondeterministically guesses which of the two (or both) variables on the right hand side of a parallel composition rule should be present in the tuple that is put on the stack. This corresponds to nondeterministically guessing the paths for the traces in the graph. Since we assume that each variable in $\mathcal{G}^k$ generates at least one graph, we are guaranteed that if $\mathcal{P}$ accepts some tuple of traces, the corresponding set of paths can be extended to a graph G derived by $\mathcal{G}$.

Transitions of $\mathcal{P}$ that add to the stack symbols which are matched against symbols of the input word from the alphabet $\big(\Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}\big)^t$ correspond to rules of the grammar of type (1). Thus, such letters correspond to tuples of edges in the graph and since the automaton simulates a derivation, the edges for each row in the word form a path. Thus, for each $i \in \{1, \ldots, t\}$ we can construct a path $\pi_i = n_0, \ldots, n_{f_i}$ of $\widetilde{\pi}_i$ with $\pi_i \in \mathsf{Paths}(G, \tau_i)$, where $\tau_i = \big(\widetilde{\tau}_i|_{\Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}}\big)|_\Sigma$. By duplicating nodes we can extend the sequence π_i to a sequence $\widetilde{\pi}_i = \widetilde{n}_{i,0}, \ldots, \widetilde{n}_{i,f}$ that satisfies condition (1) of the proposition.

The automaton $\mathcal{P}$ uses a common part of the stack symbols $\widetilde{V}$ for the common parts of the paths for different traces. These shared parts are expanded using their multiplicity before

being compared to letters of the input word. In the same way $\mathcal{P}$ checks that parts of the input letter corresponding to a shared part also share the same path index as part of the letter. In this way $\mathcal{P}$ verifies the first part of condition (2). The second part of this conditions is guaranteed by the fact that the shared parts of the stack symbols are only split into separate symbol components during transitions that correspond to parallel composition rules in the grammar. These transitions are also the exact point when symbols of the form $\{1, \ldots, t\}^t$ are put on the stack, using in the tuple the same number for all the branches emerging from a given trace. Thus the automaton checks that input symbols of this type satisfy the second part of condition (2).

To check that condition (3) is satisfied, the transitions of $\mathcal{P}$ take into account the local mapping of nodes to boolean values associated with each rule in R. Thus $\mathcal{P}$ ensures that if a register index in a symbol that is put on the stack is different from 0, then the corresponding node is marked by the mapping associated with the rule. The automaton $\mathcal{A}_r$ verifies that the letters corresponding to edges in the graph incident with a node carry the same register index. This is done in conditions (1), (2) and (4) of the definition of δ_r. Conditions (1), (3) and (4) of δ_r check that the same register index is not used for more than one node in the graph. This guarantees that if we define $\widehat{\kappa}$ such that for every $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, f\}$ we let $\widehat{\kappa}(\widetilde{n}_{i,j}) = true$ iff $j_2 > 0$, where $\widetilde{\sigma}_{i,j} = (\sigma, j_1, j_2, l)$, then we can extend $\widehat{\kappa}$ to κ such that $(G, \kappa) \in L^u(\mathcal{G}^k)$. Clearly, κ will fulfill condition (3). □

Proposition 11 *Let $(G, \kappa) \in \mathcal{L}^u(\mathcal{G}^k)$ and $G = (N, E, n_b, n_e)$. Let for every $n \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, \widetilde{r}\}$, $\widehat{\tau}_{n,h} \in \Sigma^*$ be a trace and $\widehat{\pi}_{n,h} \in N^*$ be a path such that $\widehat{\pi}_{n,h} \in \mathsf{Paths}(G, \widehat{\tau}_{n,h})$.*

Then, there exists a bijection $g : \{1, \ldots, m\} \times \{1, \ldots, \widetilde{r}\} \to \{1, \ldots, t\}$ such that if for each $i \in \{1, \ldots, t\}$ we define $\tau_i = \widehat{\tau}_{g^{-1}(i)} = \sigma_{i,1} \cdots \sigma_{i,f_i}$ and $\pi_i = \widehat{\pi}_{g^{-1}(i)} = n_{i,0}, \ldots, n_{i,f_i}$, then there exists a word $\widetilde{\tau} = \widetilde{\sigma}_1 \cdots \widetilde{\sigma}_f \in \mathcal{L}(\mathcal{P}_t)$ that satisfies the following requirements.

(1) For each $i \in \{1, \ldots, t\}$, it holds that $\tau_i = \left(\widetilde{\tau}_i|_{\Sigma \times \{0,\ldots,k\}^2 \times \{1,\ldots,t\}}\right)|_\Sigma$.

(2) For all $i', i'' \in \{1, \ldots, t\}$, $j \in \{1, \ldots, f\}$, $j' \in \{1, \ldots, \widetilde{f}_{i'}\}$, $j'' \in \{1, \ldots, \widetilde{f}_{i''}\}$ we have:

- *If $\widetilde{\sigma}_{i',j} = (\sigma', j'_1, j'_2, l')$, $\widetilde{\sigma}_{i'',j} = (\sigma'', j''_1, j''_2, l'')$, n and n' are the nodes in $\pi_{i'}$ corresponding to the position of the respective occurrence of σ' in $\tau_{i'}$, and n'' and n''' are the nodes in $\pi_{i''}$ corresponding to the position of the respective occurrence of σ'' in $\tau_{i''}$, then $l' = l''$ implies $n = n''$ and $n' = n'''$.*
- *If $\widetilde{\sigma}_{i',j'} = (\sigma', j'_1, j'_2, l')$, $\widetilde{\sigma}_{i'',j''} = (\sigma'', j''_1, j''_2, l'')$ and $l' \neq l''$, then $n_{i',j'} = n_{i'',j''}$ iff there is $\widetilde{\sigma}_{\widetilde{j}'''} = (\widetilde{\sigma}_{1,j'''}, \ldots, \widetilde{\sigma}_{t,j'''}) \in \{1, \ldots, t\}^t$ with $\widetilde{\sigma}_{i',\widetilde{j}'''} = \widetilde{\sigma}_{i'',\widetilde{j}'''}$, and*
  - *$\widetilde{j}''' \leq \widetilde{j}'$ and $\widetilde{j}''' \leq \widetilde{j}''$, where $\widetilde{j}'$ is the index in $\widetilde{\tau}$ corresponding to the index j' in $\tau_{i'}$ and $\widetilde{j}''$ is the index in $\widetilde{\tau}$ corresponding to j'' in $\tau_{i''}$,*
  - *for each $\widetilde{j}''' \leq \widetilde{j}'''' \leq \widetilde{j}'$ it holds that $\widetilde{\sigma}_{i',\widetilde{j}''''} \notin \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$,*
  - *for each $\widetilde{j}''' \leq \widetilde{j}'''' \leq \widetilde{j}''$ it holds that $\widetilde{\sigma}_{i'',\widetilde{j}''''} \notin \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$.*
- *If $\widetilde{\sigma}_{i',j'} = (\sigma', j'_1, j'_2, l')$, $\widetilde{\sigma}_{i'',j''} = (\sigma'', j''_1, j''_2, l'')$ and $l' \neq l''$, then $n_{i',j'+1} = n_{i'',j''+1}$ iff there is $\widetilde{\sigma}_{\widetilde{j}'''} = (\widetilde{\sigma}_{1,j'''}, \ldots, \widetilde{\sigma}_{t,j'''}) \in \{1, \ldots, t\}^t$ such that $\widetilde{\sigma}_{i',\widetilde{j}'''} = \widetilde{\sigma}_{i'',\widetilde{j}'''}$ and:*
  - *$\widetilde{j}''' \geq \widetilde{j}'$ and $\widetilde{j}''' \geq \widetilde{j}''$, where $\widetilde{j}'$ is the index in $\widetilde{\tau}$ corresponding to the index j' in $\tau_{i'}$ and $\widetilde{j}''$ is the index in $\widetilde{\tau}$ corresponding to j'' in $\tau_{i''}$,*
  - *for each $\widetilde{j}''' \geq \widetilde{j}'''' \geq \widetilde{j}$ it holds that $\widetilde{\sigma}_{i',\widetilde{j}''''} \notin \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$,*
  - *for each $\widetilde{j}''' \geq \widetilde{j}'''' \geq \widetilde{j}''$ it holds that $\widetilde{\sigma}_{i'',\widetilde{j}''''} \notin \Sigma \times \{0, \ldots, k\}^2 \times \{1, \ldots, t\}$.*

(3) For each $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, f_i\}$, if $\widetilde{\sigma} = (\sigma_{i,j}, j_1, j_2, l)$ is the letter of $\widetilde{\tau}_i$ corresponding to the letter $\sigma_{i,j}$ of τ_i, then $j_1 > 0$ iff $\kappa(n_{i,(j-1)})$ and $j_2 > 0$ iff $\kappa(n_{i,j})$.

Proof (*Proof sketch*) The bijection g can be defined by ordering the traces $\tau_{n,h}$ such that the order matches the derivation of the graph G. Since the graph G contains at most k registers positioned at nodes marked by a mapping κ, we can assign to each of those nodes unique identifiers from the set $\{1, \ldots, k\}$ and assign 0 to all other nodes. Let us denote with $\widehat{\kappa}(n)$ the identifier assigned to node n in this way.

We extend the traces τ_i using the symbol $\flat$ in order to align the start and end points of parallel traces. We then construct the word $\widetilde{\tau}$ by first replacing each $\sigma_{i,j}$ by $(\sigma, \widehat{\kappa}(n_{i,j-1}), \widehat{\kappa}(n_{i,j}), t)$ and each symbol $\flat$ by $(\flat, 0, 0, t)$, where each t is chosen such that in each position of $\widetilde{\tau}$ the same path index is given to all rows that share the same edge in G and the assignment is monotonic in the number of the row. Then we insert letters from the set $\{1, \ldots, t\}^t$ to mark the positions where parallel paths branch off or come together. This insertion is done in a way that satisfies the last two parts of condition (2) of the proposition. Thus, the construction of $\widetilde{\tau}$ is done in a way that fulfills the conditions of the proposition.

We can construct an accepting run of $\mathcal{P}$ on $\widetilde{\tau}$ by simulating the derivation of the graph G by $\mathcal{G}^k$ and using the paths π_i to resolve the nondeterministic choices in the automaton transitions corresponding to parallel composition rules. Similarly, the choice of placement of the symbol $\flat$ in $\widetilde{\tau}$ determines the application of the transition relation δ''_p. The positions of the letters from $\{1, \ldots, t\}^t$ at points where traces are branching off guarantees that they can be correctly matched by symbols put on $\mathcal{P}$'s stack.

Since the register indices in the letters $\widetilde{\tau}$ are obtained using $\widehat{\kappa}$ we have that no index is used for multiple nodes in the graph. Furthermore by the construction of $\widetilde{\tau}$, all letters that share a node in the graph also share the corresponding register index. This guarantees that the conditions of the transition relation δ_{r} of $\mathcal{A}_{\mathsf{r}}$ are satisfied and thus we can construct an accepting run of $\mathcal{A}_{\mathsf{r}}$ on $\widetilde{\tau}$. □

Proposition 12 *Let $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}^*$. Then $\widetilde{\tau} \in \mathcal{L}(\widetilde{\mathcal{A}_n})$ iff $\widetilde{\tau}$ satisfies the following conditions.*

(1) For each $1 \leq h \leq \widetilde{r}$ there exist $1 \leq l'_h \leq l''_h \leq f$ such that:

- *for every l such that $1 \leq l < l'_h$ or $l''_h < l \leq f$ it holds that $p_{n,h,l} = 0$,*
- *for every l such that $l'_h \leq l \leq l''_h$ it holds that $p_{n,h,l} > 0$,*
- *if h is odd and $h < \widetilde{r}$, $l''_h = l''_{h+1}$; if h is even and $h > 1$, $l'_h = l'_{h-1}$.*

(2) For each $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$ we define $\widehat{\xi}_{n,h,l}$ as follows:

- *If $\widetilde{\sigma}_{t_{n,h,l},l} = (\sigma, c, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, $p_{n,h,l} > 0$ and h is odd, then let $\widehat{\xi}_{n,h,l} = (\sigma, b_{n,h,l}, b'_{n,h,l}) \cdot q'_{n,h,l}$,*
- *If $\widetilde{\sigma}_{t_{n,h,l},l} = (\sigma, c, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, $p_{n,h,l} > 0$ and h is even, then let $\widehat{\xi}_{n,h,l} = q'_{n,h,l} \cdot (\sigma, b_{n,h,l}, b'_{n,h,l})$.*
- *In all other cases, let $\widehat{\xi}_{n,h,l} = \varepsilon$ be the empty word.*

Let $\widehat{\xi}_{n,h} = \widehat{\xi}_{n,h,1} \cdot \cdots \cdot \widehat{\xi}_{n,h,f}$ if h is odd, and $\widehat{\xi}_{n,h} = \widehat{\xi}_{n,h,f} \cdot \cdots \cdot \widehat{\xi}_{n,h,1}$ otherwise. The sequence $\xi_n = q^0 \cdot \widehat{\xi}_{n,1} \cdot \cdots \cdot \widehat{\xi}_{n,\widetilde{r}}$ is an execution of $\mathcal{M}$.

Proof (*Proof sketch*) The automaton $\widetilde{\mathcal{A}_n}$ is a two way automaton that reverses its head only at both of the word's ends and performs exactly r reversals, thus making $\widetilde{r} = r + 1$ passes through the input word. It simulates the execution of the nth machine $\mathcal{M}$ described by the part of $\widetilde{\tau}$ corresponding to the execution on the corresponding paths described by the trace part of the input word.

The sequences ξ_n are defined considering the letters of $\widetilde{\tau}$ where the block number in the corresponding row of the execution word for machine n is positive and the symbol in the corresponding row of the trace word is from the set $\Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$.

Suppose that $\widetilde{\tau} \in \mathcal{L}(\widetilde{\mathcal{A}}_n)$. The fact that $\widetilde{\tau}$ fulfills condition (1) of the statement of the proposition is ensured by conditions (3) in the definition of $\widetilde{\delta}_n$, which verify that the encoding of the reversals of machine m in the word $\widetilde{\tau}$ is correct. This means that the positions in the execution word in which the block number is 0 agree on the respective rows where the head of $\widetilde{\mathcal{A}}_n$ is moving to the (front or back) end of the word and back to the reversal point of the machine.

The transition relation $\widetilde{\delta}_n$ simulates the execution of machine n described by the execution word. This is done in condition (1) which checks that the letter in τ corresponds to a transition of $\mathcal{M}$ and that the state of the machine remains unchanged when the block index is 0 (that is, the machine is inactive for this part of the trace). Thus the automaton verifies that the sequence ξ_n is an execution of $\mathcal{M}$.

Now, consider a word $\widetilde{\tau}$ that satisfies the conditions of the proposition. Since ξ_n is an execution of $\mathcal{M}$ we can construct a run of $\widetilde{\mathcal{A}}_n$ on $\widetilde{\tau}$ by updating the index i in the state of the automaton that indicates the current row of the trace word according to ξ_n. As each letter of $\widetilde{\tau}$ corresponds to a transition step of $\mathcal{M}$, at each letter we can satisfy condition (1) of $\widetilde{\delta}_n$. The run is chosen such that it satisfies condition (2) of the definition of $\widetilde{\delta}_n$. The third condition must also be satisfied by the run since $\widetilde{\tau}$ meets condition (1) of the proposition. Since all states are accepting the constructed run is accepting. □

Proposition 13 *Consider a word $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}^*$. Fix $n \in \{1, \ldots, m\}$, $h \in \{1, \ldots, \widetilde{r}\}$ and $i \in \{1, \ldots, \widetilde{f}\}$ Consider $\widetilde{\eta}_{n,h,i} = (p_{n,h,i}, b_{n,h,i}, b'_{n,h,i}, q'_{n,h,i}, t_{n,h,i})$. If $\widetilde{\sigma}_{t_{n,h,i}} = (\sigma, l, j_1, j_2) \in \Sigma \times \{1, \ldots, t\} \times \{0, \ldots, k\}^2$, then we let $j_{n,h,i} = j_1$ if h is odd and $j_{n,h,i} = j_2$ if h is even. In the other cases we let $j_{n,h,i} = 0$.*

Now, let $\widehat{\eta}_{n,h,i} = (p_{n,h,i}, j_{n,h,i}, b_{n,h,i}, h_{n,h,i})$. Then we define $\widehat{\eta}_{n,h} = \widehat{\eta}_{n,h,1}, \ldots, \widehat{\eta}_{n,h,f}$ if h is odd and $\widehat{\eta}_{n,h} = \widehat{\eta}_{n,h,f}, \ldots, \widehat{\eta}_{n,h,1}$ if h is even.

Let us also define $\widehat{\eta}_n = \big(\widetilde{\eta}_{n,1} \cdot \cdots \cdot \eta_{n,\widetilde{r}}\big)|_{\left(\{1,\ldots,u\} \times \{0,\ldots,k\} \times \mathbb{B}^2\right)}$.

The word $\widetilde{\tau}$ is accepted by $\widetilde{\mathcal{A}_{\mathsf{c}}}$ iff it satisfies the two conditions below.

(1) Let $u = \max\big\{p_{n,h,l} \mid n \in \{1, \ldots, m\}, h \in \{1, \ldots, \widetilde{r}\}, l \in \{1, \ldots, f\}\big\}$.

- *For every $1 \leq v \leq u$ there exists exactly one $n \in \{1, \ldots, m\}$ such that there exist $h \in \{1, \ldots, \widetilde{r}\}$ and $l \in \{1, \ldots, f\}$ with the property $p_{n,h,l} = v$.*
- *For all $n \in \{1, \ldots, m\}$ and $0 \leq l_1 \leq l_2 < |\widehat{\eta}_n|$ where $\widehat{\eta}_n[l_1] = (p_1, j_1, b_1, b'_1)$ and $\widehat{\eta}_n[l_2] = (p_2, j_2, b_2, b'_2)$ it holds that $p_1 \leq p_2$.*

(2) Let $\widehat{\eta}$ be the unique permutation of $\widehat{\eta}_1 \cdot \cdots \cdot \widehat{\eta}_m$ that is such that

- *the elements of each $\widehat{\eta}_n$ appear in $\widehat{\eta}$ in the same order as in $\widehat{\eta}_n$,*
- *for each $0 \leq l_1 \leq l_2 < |\widehat{\eta}|$, where $\widehat{\eta}[l_1] = (p_1, j_1, b_1, b'_1)$ and $\widehat{\eta}[l_2] = (p_2, j_2, b_2, b'_2)$ it holds that $p_1 \leq p_2$.*

Then, the sequence $\eta = \widehat{\eta}|_{(\{0,\ldots,k\} \times \mathbb{B}^2)}$ is a read–write sequence that is valid w.r.t. initial valuation $\beta_0 : \{1, \ldots, k\} \to \mathbb{B}$, where $\beta_0(j) = 0$ for each $j \in \{1, \ldots, k\}$.

Proof (*Proof sketch*) The automaton $\widetilde{\mathcal{A}_{\mathsf{c}}}$ is a two way automaton that reverses its head only at both of the word's ends and performs exactly r reversals, thus making $\widetilde{r} = r + 1$ passes through the input word. On each pass it traverses m rows of the execution word (one for each of the machines) in the same direction as the machines traverse the graph in that pass.

Suppose that $\widetilde{\tau} \in \mathcal{L}(\mathcal{A}_{\mathsf{c}})$. Condition (1) from the definition of $\widetilde{\delta_{\mathsf{c}}}$ verifies that the positive block numbers for each machine are nondecreasing and that each block number belongs to at most one machine. The requirement that there are no gaps in the block numbers is ensured by the definition of accepting states where it is checked that P contains all numbers less or equal than the maximum.

Condition (2) of the transition relation $\widetilde{\delta_{\mathsf{c}}}$ guarantees that $\widetilde{\tau}$ encodes a valid read–write sequence. This is done as follows. The transition relation checks that all read operations except those at the beginning of a block read the current value of the registers that is stored in the automaton's state. The value read at the beginning of a block is guessed, and this guess is added to the set A of assumptions. The values from write operations in $\widetilde{\tau}$ are used to update the current value for the respective machine and are stored in the set G of guarantees at the end of each block. The condition that the read–write sequence η is valid is ensured by requiring that in an accepting state the set A must be empty, that is, all assumptions in A have been discharged by a matching guarantee from G.

Thus, since $\widetilde{\tau}$ is accepted it must satisfy the conditions of the proposition.

Suppose now that $\widetilde{\tau}$ satisfies the conditions of the proposition. Condition (1) ensures that for each letter of $\widetilde{\tau}$ condition (1) of the transition relation $\widetilde{\delta_{\mathsf{c}}}$ is satisfied. For constructing a run of $\widetilde{\mathcal{A}_{\mathsf{c}}}$ on $\widetilde{\tau}$ we guess at each step for each machine register values equal to the ones from the preceding block as given in $\widetilde{\tau}$. Since these guesses will obviously be matched by the respective guarantee, the word $\widetilde{\tau}$ will be accepted by $\widetilde{\mathcal{A}_{\mathsf{c}}}$. □

Proposition 14 *Consider a word* $\widetilde{\tau} = \widetilde{\sigma_1} \cdots \widetilde{\sigma_f} \in \widetilde{\Sigma}^*$ *and let* $P_n = \{p_{n,h,l} \mid 1 \leq h \leq \widetilde{r}, 1 \leq l \leq f\}$. *The word* $\widetilde{\tau}$ *is accepted by* $\mathcal{A}_{\mathsf{s}}$ *iff there exists a tuple* (c', c'') *occurring in* $\widetilde{\tau}$, *set* $M \subseteq \{1, \ldots, m\}$ *and a function* $f_M : M \to \{1, \ldots, \widetilde{r}\}$ *which satisfy the predicate* NodeProperty$(S_{final}, A, c', c'', M, f_M)$ *and are such that for every* $n \in M$ *it holds that* $p_n(c', c'', M, f_M) = \max P_n$.

Proof (*Proof sketch*) $\mathcal{A}_{\mathsf{s}}$ is an NFA, reading simultaneously all $\widetilde{r}$ rows in the execution word of the input word τ and the respective rows of the trace word, and checking for the occurrence of final configurations.

Suppose that $\widetilde{\tau} \in \mathcal{L}(\mathcal{A}_{\mathsf{s}})$, and thus there exists an accepting run of $\mathcal{A}_{\mathsf{s}}$ on $\widetilde{\tau}$.

Since the run of $\mathcal{A}_{\mathsf{s}}$ on $\widetilde{\tau}$ is accepting, when reading the right endmarker of the word, either we have $final = 1$ in the current state, which means that a potential final configuration was already identified, or the current state of the automaton encodes a potential final configuration. $\mathcal{A}_{\mathsf{s}}$ verifies the existence of an actual final configuration by checking that for each machine n of the ones involved in this configuration, the block number p_n in that configuration is maximal in the set P_n at the end of the run, which contains all block numbers for n encountered in $\widetilde{\tau}$. This, in turn, guarantees that all involved machines are simultaneously located at the respective node of the graph, in their respective local states. Thus, the existence of an accepting run implies the property stated in the proposition.

For the other direction, suppose that $\widetilde{\tau}$ satisfies the condition stated in the proposition. This allows us to provide an accepting run of $\mathcal{A}_{\mathsf{s}}$ on $\widetilde{\tau}$ as follows. The components p_n of the automaton's state are updated when the tuple (c', c'') that exists according to condition (1) is encountered in $\widetilde{\tau}$. By assumption, the requirement that for each involved machine n it holds that $p_n = \max P_n$ will be satisfied. Thus, the run constructed in this way will be accepting. □

References

1. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004—Concurrency Theory, 15th International Conference, London, UK, 31 Aug–3 Sept 2004, Proceedings, Lecture Notes in Computer Science, vol. 3170, pp. 35–48. Springer (2004). doi:10.1007/978-3-540-28644-8_3
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: CONCUR'99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, 24–27 Aug 1999, Proceedings, Lecture Notes in Computer Science, vol. 1664, pp. 114–129. Springer (1999). doi:10.1007/3-540-48320-9_10
3. Baldan, P., Corradini, A., König, B.: A static analysis technique for graph transformation systems. In: Proceedings of CONCUR'01, LNCS, vol. 2154, pp. 381–395. Springer (2001). doi:10.1007/3-540-44685-0_26
4. Baldan, P., Corradini, A., König, B., Lluch-Lafuente, A.: A temporal graph logic for verification of graph transformation systems. In: WADT, LNCS, vol. 4409, pp. 1–20. Springer (2006). doi:10.1007/978-3-540-71998-4_1
5. Bertrand, N., Delzanno, G., König, B., Sangnier, A., Stückrath, J.: On the decidability status of reachability and coverability in graph transformation systems. In: RTA, LIPIcs, vol. 15 (2012). doi:10.4230/LIPIcs.RTA.2012.101
6. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic—A Language-Theoretic Approach, vol. 138. Cambridge University Press (2012). doi:10.1017/CBO9780511977619
7. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific (1995)
8. Dimitrova, R., Majumdar, R.: Reachability analysis of reversal-bounded automata on series-parallel graphs. In: Esparza, J., Tronci, E. (eds.) Proceedings Sixth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2015, Genoa, Italy, 21–22 Sept 2015. EPTCS, vol. 193, pp. 100–114 (2015). doi:10.4204/EPTCS.193.8
9. Drewes, F., Kreowski, H.-J., Habel, A.: Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific Publishing Co., Inc., pp. 95–162 (1997). doi:10.1142/9789812384720_0002
10. Engelfriet, J., Rozenberg, G.: In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, chap. Node Replacement Graph Grammars, pp. 1–94. World Scientific Publishing Co., Inc. (1997). doi:10.1142/9789812384720_0001
11. Esparza, J., Ganty, P., Majumdar, R.: A perfect model for bounded verification. In: LICS 2012, pp. 285–294. IEEE Computer Society (2012). doi:10.1109/LICS.2012.39
12. Esparza, J., Ganty, P., Poch, T.: Pattern-based verification for multithreaded programs. ACM Trans. Program. Lang. Syst. **36**(3), 9:1–9:29 (2014). doi:10.1145/2629644
13. Gurari, E., Ibarra, O.: The complexity of decision problems for finite-turn multicounter machines. J. Comput. Syst. Sci. **22**(2), 220–229 (1981). doi:10.1016/0022-0000(81)90028-3
14. Gurari, E., Ibarra, O.: Two-way counter machines and Diophantine equations. J. ACM **29**(3), 863–873 (1982). doi:10.1109/SFCS.1981.52
15. Heußner, A., Leroux, J., Muscholl, A., Sutre, G.: Reachability analysis of communicating pushdown systems. Log. Methods Comput. Sci. (2012). doi:10.2168/LMCS-8(3:23)2012
16. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, 2nd edn. Addison-Wesley, Boston (2000)
17. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. ACM **25**(1), 116–133 (1978). doi:10.1145/322047.322058
18. Ibarra, O.H.: Automata with reversal-bounded counters: a survey. In: DCFS 2014, pp. 5–22. Springer (2014). doi:10.1007/978-3-319-09704-6_2
19. König, B., Kozioura, V.: Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In: TACAS, LNCS, vol. 3920, pp. 197–211. Springer (2006). doi:10.1007/11691372_13
20. Lodaya, K., Weil, P.: Series-parallel posets: Algebra, automata and languages. In: Morvan, M., Meinel, C., Krob, D. (eds.) STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, 25–27 Feb 1998, Proceedings, Lecture Notes in Computer Science, vol. 1373, pp. 555–565. Springer (1998). doi:10.1007/BFb0028590
21. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: Ball, T., Sagiv, M. (eds.) Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, 26–28 Jan 2011, pp. 283–294. ACM (2011). doi:10.1145/1926385.1926419
22. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959). doi:10.1147/rd.32.0114

23. Rensink, A.: Explicit state model checking for graph grammars. In: Concurrency, Graphs and Models, LNCS, vol. 5065, pp. 114–132. Springer (2008). doi:10.1007/978-3-540-68679-8_8
24. Rosenberg, A.L.: On multi-head finite automata. In: 6th Annual Symposium on Switching Circuit Theory and Logical Design, pp. 221–228. IEEE Computer Society (1965). doi:10.1109/FOCS.1965.19
25. Vardi, M.: From löwenheim to PSL and SVA. In: Language, Culture, Computation. Computing—Theory and Technology—Essays Dedicated to Yaacov Choueka on the Occasion of His 75th Birthday, Part I, Lecture Notes in Computer Science, vol. 8001, pp. 78–102. Springer (2014). doi:10.1007/978-3-642-45321-2_5