CHENNAI MATHEMATICAL INSTITUTE

MASTERS THESIS

A Survey on Graph Automata and Tree-Width

Author:
Adwitee Roy

Supervisor:
Aiswarya Cyriac

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

in the

Computer Science Department
Chennai Mathematical Institute

Abstract

This thesis is a survey on some results of Graph Automata and tree-width. We introduce the concepts of graph automata and study the language emptiness and finiteness of it. We see that the language emptiness and finiteness of graph automata is undecidable. Then we move on to the class of nested words and we see that the language emptiness and finiteness is decidable for this class. For multiply nested word automata, the language emptiness and finiteness is undecidable in general but they become decidable for bounded split-width.

Acknowledgements

I would like to thank my supervisor, Aiswarya Cyriac, for introducing me to Graph Automata, and for all the helpful discussions that we have had. I also wish to thank Prof. Madhavan Mukund, B. Srivathsan, and M. Praveen for the exciting courses in Logic and Automata Theory that I took under them.

Contents

Chapter 1

Introduction

Trees has some very distinctive and fundamental properties. It is therefore legitimate to ask to what degree those properties can be transferred to more general graphs, graphs that are themselves not trees but 'tree-like' in some sense.

A tree-decomposition of a graph G is a way to find the 'tree-likeness' of the graph G. This helps in solving some problems in G. In chapter 2, we study the concept of tree decomposition and tree-width.

In the third chapter we study the model of Graph Automata, which generalizes the known models of automata on words, trees, and directed acyclic graphs. For specifying graph properties by finite acceptors we describe the idea of *local tests* by tiling. The language emptiness and finiteness of graph automata is undecidable as we can encode the halting problem of Turing machine to the emptiness and finiteness problem of graph automata.

In the next chapter we restrict the graph automata model on nested words. A nested word consists of a sequence of linearly ordered positions, augmented with nesting edges connecting calls to returns. Here we have a stack just like context free language but we can push anything in the stack. We study nested word automata (NWA) as acceptors of nested words for which we can easily give a graph automata. The resulting class of language for NWAs has all the nice theoretical properties that the regular languages of words and trees enjoy. We will see how similar the nested words are to the trees. We will also see that the decision problems such as language emptiness and finiteness are decidable for this class of language.

Then we move on to multiply nested words, where we can have finite number of stacks. We will see the similarities between this class of language and grids. The language emptiness and finiteness are undecidable here but they become decidable for bounded tree-width or bounded split-width.

Chapter 2

Tree-width

2.1 Tree Decomposition and Treewidth

Trees are graphs with some very distinctive and fundamental properties. It is therefore legitimate to ask to what degree those properties can be transferred to more general graphs, graphs that are themselves not trees but 'tree-like' in some sense. A tree-decomposition of a graph G is a representation of G in a tree-like structure. From this structure it is possible to deduce certain connectivity properties of G. Such information can be used to construct efficient algorithms to solve problems on G. Sometimes problems which are NP-hard in general are solvable in polynomial or even linear time when restricted to trees. Employing the tree-like structure of tree-decompositions these algorithms for trees can be adapted to graphs of bounded tree-width.

Definition 2.1.1 Let G be a graph, T a tree, and let $V' = (V_t)_{t \in T}$ be a family of vertex sets $V_t \subseteq V(G)$ indexed by the vertices t of T. The pair (T, V') is called a *tree-decomposition* of G if it satisfies the following conditions:

(**TD1**) $V(G) = \bigcup_{t \in V(T)} V_t$

(**TD2**) for every edge $e \in G$ there exists a $t \in T$ such that both ends of e lie in V_t

(**TD3**) $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$ whenever $t_1, t_2, t_3 \in T$ satisfy $t_2 \in t_1 T t_3$

We call the sets V_t, and sometimes also their induced subgraphs $G[Vt]$, the parts of the tree-decomposition (T, V'). For the rest of this section, let (T, V') be a tree-decomposition of G = (V, E) as defined above. Note that tree-decompositions are passed on to subgraphs and for contractions.

Lemma 2.1.2. Any complete subgraph of G is contained in some *part* of (T,V').

Lemma 2.1.3. For every $H \subseteq G$, the pair $(T, (V_t \cap V(H))_{t \in T})$ is a *tree-decomposition* of H.

Proof of the above two lemmas can be found in [4].

Definition 2.1.4. The *width* of a tree-decomposition (T, V') is the number,

$$max|V_t| - 1 : t \in T$$

The *tree-width* of a graph G is the minimum *width* over all possible tree-decompositions of G.

As one easily checks, trees themselves have tree-width 1. The following lemma shows that, the tree-width of a graph will never be increased by deletion or contraction.

Lemma 2.1.5. If H is a minor of a graph G then H has tree-width at most tree-width of G.

Proof. As can easily bee seen, deleting vertices or edges can not increase the tree-width of a graph. Now we examine contraction of an edge. Let (T, V') be a tree-decomposition of $G = (V, E)$ of width w and $e = uv \in E$ the edge to be contracted to a vertex v_e. Set W_i to be the part V_i, with the only difference that every occurrence of u or v is replaced by v_e and set $T' = T$. We now show that (T', W') is a tree decomposition of H with width at most w since replacing u and v by v_e can not increase the tree-width w.

Intuitively, it is clear that we have found a tree-decomposition of H. T_u and T_v were subtrees of T meeting in at least one vertex t. From this follows that ${T'}_{v_e}$ will be connected and since T' contains no cycles ${T'}_{v_e}$ will also be a subtree of T.

We prove **(TD1)-(TD3)** for (T', W). **(TD1)** is clear from $(TD1)$ for (T, V') and the replacement of u, v by v_e. For every edge wu, $w \in V$, there exists t' with $w, u \in {V'}_t$ by **(TD2)**, so W'_t contains wv_e. The same follows for edges $wv, w \in V$, and this implies **(TD2)** for (T', W'). Now let $t_1, t_2 \in E(T')$ with v_e in $W_{t_1} \cap W_{t_2}$. That means that either $u \in V_{t_1} \cap V_{t_2}$ or $v \in V_{t_1} \cap V_{t_2}$ or (without loss of generality) $u \in V_{t_1} \setminus V_{t_2}$ and $v \in V_{t_2} \setminus V_{t_1}$. The first two cases, together with **(TD3)** for (T, V'), imply $\forall t \in t_1 T t_2 : v_e \in W_t$ for (T', W'). Concerning the third case, we know that, by **(TD2)** and **(TD3)** for (T, V'), there has to exist $t \in t_1 T t_2$ such that both u, v are in V_t. **(TD3)** for (T, V') now tells us that u is a member of every set $V_{t'}$ between V_{t_1} and V_t, and v is contained in the other parts of the path $t_1 T t_2$. Thus v_e is included in every $W_{t''}$ with $t'' \in t_1 T' t_2$, which completes the proof. □

The above proof is from [4].

2.2 Other Representations of Tree Decomposition

2.2.1 Cops and Robbers

Here is a cops-and-robber game from [9], played on a finite, undirected graph G. The robber stands on a vertex of the graph, and can at any time run at great speed to any other vertex along a path of the graph. He is not permitted to run through a cop, however. There are k cops, each of whom at any time either stands on a vertex or is in a helicopter (that is, is temporarily removed from the game). The objective of the player controlling the movement of the cops is to land a cop via helicopter on the vertex occupied by the robber, and the robber's objective is to elude capture. The robber can see the helicopter approaching its landing spot and may run to a new vertex before the helicopter actually lands.

There are two forms of this game. In the first, the robber is invisible, and so to capture him the cops must methodically search the whole graph. If k cops can guarantee to catch the robber, then k cops can search the graph monotonely, that is, never returning to a vertex which a cop has previously vacated. Here we are concerned about the second form of the game, where the cops can see the robber at all times- the difficulty is just to corner him somewhere. Two cops suffice to catch a visible robber if G is a tree. Put cop 1 on some vertex v, see which component of $G \backslash v$ contains the robber, and transport cop 2 to the neighbour of v in that component. Now repeat with the cops exchanged. However, two cops may not

be able to catch an invisible robber. If k cops can catch a visible robber then again there is a monotone search strategy; and on the other hand, if a visible robber can guarantee to elude k cops, then there is an escape strategy with a particularly simple form.

Definition 2.1.6. Let $G\backslash X$ be the graph obtained from G by deleting X, where X may be a vertex or an edge, or a set of vertices or edges. The vertex set of a component of $G\backslash X$ is called an *X-flap*.

We denote by $[V]^{<k}$ the set of all subsets of V of cardinality $< k$. A position is a pair (X, R), where $X \in [V(G)]^{<k}$ and R is an *X-flap*. Here, X is the set of vertices currently occupied by cops. R tells us where the robber is- since he can run arbitrarily fast, all that matters is which component of $G\backslash X$ contains him. We set (X_0, R_0) to an initial position. In the normal game, $X_0 = \phi$ and the robber player chooses R_0 to be some component of G.

At the start of the i^{th} step we have a position (X_{i-1}, R_{i-1}). The cop player chooses a new set $X_i \in [V(G)]^{<k}$ such that either $X_{i-1} \subseteq X_i$ or $X_i \subseteq X_{i-1}$. Then the robber player chooses (if possible) an $X_i - flap\ R_i$ satisfying $R_i \subseteq R_{i-1}$ or $R_{i-1} \subseteq R_i$ respectively. If this choice is impossible, that is, if $V(R_{i-1}) \subseteq X_i$, the cop player has won, and otherwise the game continues with step $i+1$. The robber player thus cannot win; his objective is to stop the cop player winning.

If there is a winning strategy for the cop player, we say that "$< k$ cops can search the graph." If in addition the cop player can always win in such a way that the sequence $X_0, X_1, \ldots$ satisfies $X_i \cap X_i'' \subseteq Xi'$ for $i \leq i' \leq i''$, we say that $< k$ cops can monotonely search the graph." If $< k$ cops can search G then they can monotonely search it [9].

Two subsets $X, Y \subseteq V(G)$ touch if either $X \cap Y \neq \phi$ or some vertex in X has a neighbor in Y . Here is another, similar game, in which the cop player has slightly more power. We set (X_0, R_0) as before. At the start of the i^{th} step we have a position (X_{i-1}, R_{i-1}). The cop player chooses a new set $X_i \in [V(G)]^{<k}$ with no restriction on X_i. Then the robber player chooses (if possible) an $Xi - flap\ R_i$ which touches R_{i-1}. (In all other respects the game is unchanged.) We call this jump-searching; and define "$< k$ cops can jump-search G" etc. as before.

The following Theorem is due to Robertson and Seymour [9].
Theorem 2.1.7. Let G be a graph, and $k - 1$ an integer. Then the following are equivalent:

(i) $< k$ cops cannot jump-search G,

(ii) $< k$ cops cannot search G,

(iii) $< k$ cops cannot monotonely search G,

(iv) G has tree-width $k - 1$.

2.2.2 Bramble

Brambles are useful to deduce some information about a graph from the assumption that it has large tree-width. There is a theorem due to Seymour and Thomas [[9]] which identifies a canonical obstruction to small tree-width, a structural phenomenon that occurs in a graph if and only if its tree-width is large. Before stating the theorem we need some definitions.

Definition 2.4.1. Two sets $A, B \in V$ *touch* if they have a vertex in common or there exists an edge in G with end vertices in A and B. A *bramble* $\mathcal{B}$ is a set of mutually touching connected vertex sets. A *cover* of a bramble $\mathcal{B}$ is a set $S \subseteq V$ such that every set in $\mathcal{B}$ contains at least one vertex of S. The cardinality of a vertex-minimal cover of $\mathcal{B}$ is called the *order* of the bramble $\mathcal{B}$.

Lemma 2.4.2. Any set separating two covers of a bramble $\mathcal{B}$ is also a cover of $\mathcal{B}$.

Proof. Every set $B \in \mathcal{B}$ is connected and contains a vertex of each cover, therefore also a path between them. So every set separating the two covers contains at least one vertex of B. □

A typical example of a bramble is the set of crosses in a *grid*. The $k \times k$ *grid* is the graph on $\{1, 2, \ldots, k\}^2$ with edge set

$$\{(i,j)(i',j') : |i - i'| + |j - j'| = 1\}.$$

The *crosses* of this grid are the k^2 sets

$$C_{ij} := \{(i,\ell) | \ell = 1, 2, \ldots, k\} \cup \{(\ell, j) | \ell = 1, 2, \ldots, k\}.$$

Thus the cross C_{ij} is the union of the grid's i^{th}column and its j^{th} row. Clearly, the crosses of the $k \times k$ grid form a bramble of order K; they are covered by any row or column, while any set of fewer than k vertices misses both a row and a column, and hence a cross.

Definition 2.4.3. G has bramble number $\beta(G) = n$ if n is maximal such that G contains a bramble of order n.

Like complete subgraphs, the brambles contained in a given graph affect its tree-width. The following theorem is stated without a proof [9].

Theorem 2.4.4. Let $G = (V, E)$ be a graph, Then we have tree-width of G is equal to $\beta(G) - 1$.

The following theorem due to Seymour and Thomas [9] is sometimes called *tree-width duality theorem*.

Theorem 2.4.5. Let $k \geq 0$ be an integer. A graph has tree-width $\geq k$ iff it contains a bramble of order $> k$.

2.3 Coloring

A legal k-coloring of a graph colors the vertices with k colors such that adjacent vertices receive different colors. The chromatic number of a graph is the minimum k that allows a legal k-coloring. Computing the chromatic number is NP-hard.

Proposition 2.5.1 Every graph with tree-width p has chromatic number at most $p + 1$.

2.4 Graph Algebra

[2] has given algebraic characterization of tree-width. He we give the syntax from [3]. Let C be a finite set of colors. Then C-tree-expressions are given by:

$$te ::= \quad x \quad | \quad x\mathsf{E}y \quad | \quad te_1 || te_2 \quad | \quad \mathsf{rnm}_{x<->y}(te) \quad | \quad \mathsf{fg}_x(te)$$

where $x, y \in C$ and E is an edge relation. Each C-tree-expression defines an edge labeled graph (up to isomorphism) as described below:

- The expression x denotes the graph with a single vertex colored x.
- The expression $x\mathsf{E}y$ (with $x \neq y$, considering graphs without self-loops) denotes the graph with two vertices colored x and y and these vertices are connected by an edge E.
- The expression $te_1 || te_2$ (parallel composition) denotes the disjoint union of the graphs defined by the expressions te_1 and te_2, where the nodes with the same label are fused.
- The expression $\mathsf{rnm}_{x<->y}(te)$ (renaming) denotes the graphs obtained by re-coloring the vertices colored x and y in the graph denoted by te with y and x.
- The expression $\mathsf{fg}_x(te)$ (forget color) denotes the graph obtained by removing the color of the vertices colored x in the graph denoted by te.

There can be at most one vertex with color x for each color x, since the parallel composition fuses nodes with the same color. Also once the color of a vertex is forgotten, that vertex cannot be colored later.
The tree-width of a graph is at most $|C| - 1$ if there is a C-tree-expression denoting it [3].

2.5 Some Classes of Graph

A number of NP-complete graph problems become polynomial-time solvable if their inputs are restricted to particular classes of graphs such as those of trees, of series-parallel graphs, of planar graphs etc. For many of these classes, in particular for trees, almost trees (with parameter k), partial k-trees, series-parallel graphs, outerplanar graphs and cographs, the efficient algorithms take advantage of certain hierarchical structures of the input graphs. Because of these structures,these graphs are somehow close to trees. We discuss tree-like nature of Series Parallel graph in the following subsection.

2.5.1 Series Parallel Graph

Definition 2.7.1. A series-parallel graph is a graph obtained from an independent set using the following operations:

1. Add a new node and connect it to an existing node by an edge.
2. Add a self loop.
3. Add an edge in parallel to an existing edge.

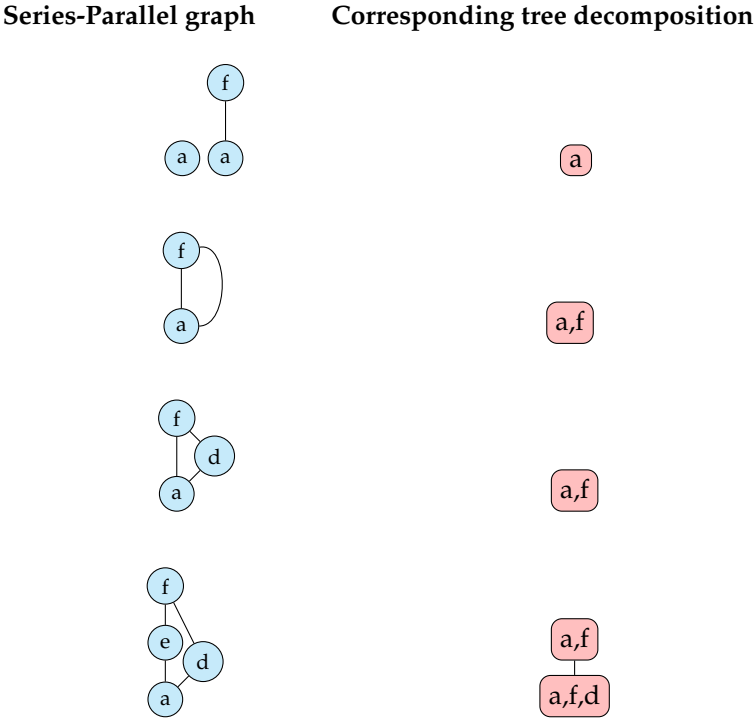
4. Subdivide an edge by creating a node in the middle.

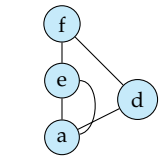
Proposition 2.7.2. G has tree-width at most 2 iff it is a series-parallel graph.

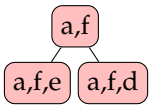
Proof. Let G be a series-parallel graph. We build a tree decomposition for it inductively, without ever exceeding tree-width 2. We do this by following the inductive construction of G.
For step 1, suppose v is added to an existing node u. Before adding v, the tree decomposition included a vertex i such that $u \in B_j$. Create a new node labeled by $\{u, v\}$ and connect it to i. For step 2 and 3 the tree decomposition does not change. The most involved step is step 4 in the construction of series parallel graphs. Suppose edge (u, v) was subdivided, introducing a vertex w. Prior to the subdivision, the tree decomposition included a vertex j such that $u, v \in B_j$. Create a new node labeled by $\{u, v, w\}$ and connect it to j. Conversely, assume a tree decomposition T with tree-width at most 2. We show that this corresponds to a series-parallel graph. We do this by removing a node from the tree decomposition, show that this corresponds to removing a vertex from the graph G, and then show that G can be obtained by adding back the vertex using one of the rules of Definition 2.7.1. Again we illustrate only one case as the other cases are easy to see.
Take a leaf l of T. Assume that it is labeled by three vertices u, v, w (it cannot be labeled by more than three vertices). Then without loss of generality, the neighbor of l in T does not contain w. Hence the only possible neighbors of w in G are u and v. If both are indeed neighbors, then w could have been obtained by subdividing an edge (u, v), and the graph prior to subdividing also had tree width at most 2 (by removing w from l). □

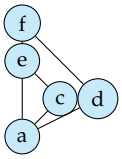
2.5.2 Example of series-parallel graph and its tree decomposition

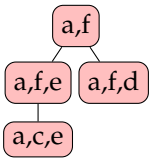
Below is an example of the construction and tree decomposition of a series parallel graph:

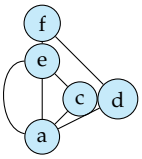


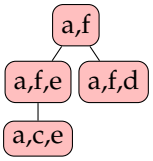

f
e
d
a

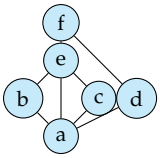

a,f
a,f,e
a,f,d

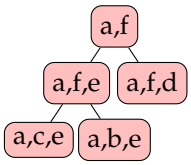

f
e
c
d
a


a,f
a,f,e
a,f,d
a,c,e


f
e
c
d
a


a,f
a,f,e
a,f,d
a,c,e


f
e
b
c
d
a


a,f
a,f,e
a,f,d
a,c,e
a,b,e

Chapter 3

Graph Automata

3.1 Introduction

[10] has given the model of Graph Automata, which generalizes the known models of automata on words, trees, and directed acyclic graphs. The idea of "local tests" in graph automata are realised in the form of "tiling by transitions".

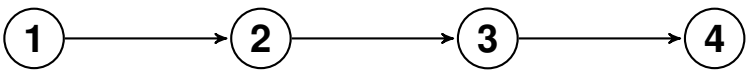
3.2 Graph Recognizability by Tiling

We consider directed graphs whose vertices and edges are labelled with symbols of finite alphabets A and B, respectively. Formally, these graphs are structures $G = (VE, \beta, \alpha)$ where V is a nonempty and finite set, $E \subseteq V \times V$, and $\beta : E \to B$ and $\alpha : V \to A$ are the valuations. We represent these graphs as relational structures, in the form $(V, (E_b)_{b\in B}, (P_a)_{a\in A})$ where the E_b are pairwise disjoint binary relations over V and the P_a are unary predicates which form a partition of V.

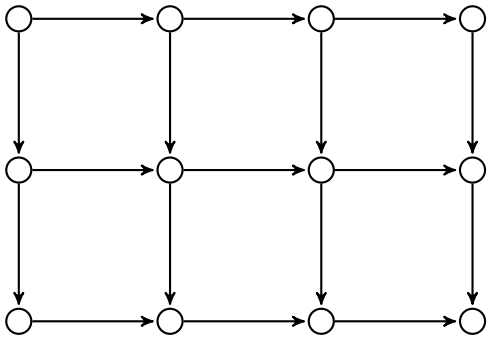
For specifying graph properties by finite acceptors we describe the idea of *local tests* by transitions. A transition associates states with the vertices of a *local neighbourhood* in a graph, given by at least one vertex together with its adjacent vertices. If the admitted graph acceptors are finite, they involve only finitely many transitions each of which is a finite object. If there is no bound to the degree of the vertices, some vertices may have any number of adjacent vertices. But we can check the local neighbourhoods only when the degree of the vertices are upper bounded by the maximum degree in the set of transitions. So, we consider only graphs whose degree is uniformly bounded by some constant k.

Let $DG_k(A, B)$ be the class of finite directed graphs $(V, (E_b)_{b\in B}, (P_a)_{a\in A})$ as described above, where for each vertex x there are at most k vertices y with $(x, y) \in E_b$ or $(y, x) \in E_b$ for some $b \in B$. We consider graphs of degree k.
Now we give some examples of graphs as relational structures.

1. **Words over an alphabet A**: A nonempty word $w = a_1, \ldots, a_n$ can be represented by the graph $(\{1, \ldots, n\}, S, (P_a)_{a\in A})$ where S is the successor relation on $\{1, \ldots, n\}$ and $P_a = \{i | a_i = a\}$.



2. **Grids**: These are graphs of the form $G_{m,n} = (\{1, \ldots, m\} \times \{1, \ldots, n\}, E_l, E_2)$ where the edge sets E_1 and E_2 are given by $((x, y), (x + 1, y)) \in E_1$ for $I \leq x < m, 1 \leq y \leq n$, $((x, y), (x, y + 1)) \in E_2$ for $1 \leq x \leq m, 1 \leq y < n$ and we assume a trivial vertex valuation.



Now we describe Graph Acceptor [10].

Definition 3.1.1 A finite graph acceptor over $DG_k(A, B)$ is a triple $\mathcal{A} = (Q, \Delta, C)$ where,

- Q is a finite set of states,
- Δ is a finite set of connected graphs in $DG_k((Q \times A) \cup Q, B)$, called the set of transitions (or "tiles"),
- and C, called constraint, is a boolean combination of conditions of the form "there are $\geq n$ copies of transition τ" (where $\tau \in \Delta$).

The graph acceptor (Q, Δ, C) accepts a graph $G = (V, E, \beta, \alpha)$ if G can be *tiled coherently* by transitions from Δ obeying the constraint C. Here *coherence* means that the A- and B-values of the transitions agree with the valuation of the underlying graph G, and the transitions overlap such that only one state is associated with each vertex of G. We can also say that the tiling define some *run* $\rho : V \to Q$.

Now we describe *coherent tilings* precisely. Fix $G = (V, E, \beta, \alpha)$ and let $\rho : V \to Q$. Define the corresponding extended vertex valuation $\rho \times \alpha : V \to Q \times A$ by $\rho \times \alpha(x) = (\rho(x), \alpha(x))$. We denote by G_ρ the graph $(V, E, b, \rho \times \alpha)$. A subgraph of $G_\rho = (V, E, \rho \times \alpha)$ is a graph $G' = (V', E', \beta', (\rho \times \alpha)')$ where $V' \subseteq V$, $E' = E \cap (V' \times V')$, and β', $(\rho \times \alpha)'$ are the restrictions of β, $\rho \times \alpha$ to V'. Vertex $x \in V'$ is called a *border* vertex if there is an edge (x, y) or (y, x) in E with $y \notin V'$. The *core* of G' is the set of vertices of G' which are not border vertices. We write $[G']$ for the graph which results from G' by erasing the A-values for the border vertices. Thus $[G']$ has a vertex valuation in $(Q \times A) \cup Q$. Let us say that G' matches the transition τ if $[G']$ and τ are isomorphic (via a bijection preserving vertex and edge labels, hence mapping core to core and border to border). We say that G_ρ satisfies the condition "there are $\geq n$ copies of τ" if there are $\geq n$ distinct occurrences of graphs $[G']$ isomorphic to τ within G. Applied to boolean combinations of such conditions,this fixes the meaning of the statement "G_ρ satisfies the constraint C".

The run $\rho : V \to Q$ of $A = (Q, \Delta, C)$ on G is called successful if,

1. every local neighbourhood bounded by the maximum size of the transitions is in the core of a subgraph of G_ρ which matches some transition of Δ,
2. G_ρ satisfies the constraint C.

Let us say that $\mathcal{A}$ accepts G if there is a successful run of A on G. Given a class $\mathcal{G}$ of graphs in $DG_k(A, B)$ and a graph set $\mathcal{L} \subseteq \mathcal{G}$, $\mathcal{A}$ recognlizes L relative to $\mathcal{G}$ if for any graph $G \in \mathcal{G}$, we have $G \in \mathcal{L}$ iff $\mathcal{A}$ accepts G. Then $\mathcal{L}$ is called recognizable by tilings (or short *t-recognialzable*) relative to $\mathcal{G}$. We call a graph acceptor *elementary* if it contains only transitions with just

one core vertex and only edges which start or end in this vertex, excepting the transition with empty border. We call a set of graphs recognized by an elementary graph acceptor *e-recognizable*.

Theorem 3.1.2. [10] Any *t-recognizable* set of graphs in $DG_k(A,B)$ is definable in monadic second-order logic (using the signature with equality, the binary predicate symbols E_b for $b \in B$, and the unary predicate symbols P_a for $a \in A$).

Proof. MSO formulas over $DG_k(A,B)$ graphs are given by the following syntax.

$$\varphi = P_a(x) \mid x = y \mid E_b(x,y) \mid x \in X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \forall x\varphi \mid \exists X\varphi \mid \forall X\varphi$$

where x, y, x_1, x' etc are first order variables that vary over vertices of the graph, X, Y etc are set variables that vary over subsets of vertices of the graph, $a \in A$ and $b \in B$. The semantics is as expected.

A formula without any free variables is a sentence. Each MSO sentence defines a language of $DG_k(A,B)$ graphs. A formula is said to be in *existential* MSO if it is of the form $\exists X_1 \exists X_2 \ldots \exists X_m \varphi$ where φ does not contain subformulas of the form $\exists X\varphi'$ or $\forall X\varphi'$.

Let $A = (Q, \Delta, C)$ be the the graph acceptor. We will in fact give an *existential* MSO formula recognising the set of graphs accepted by A. Let $Q = \{q_1, \ldots, q_\ell\}$ be the set of states of A. We will use ℓ set variables $X_1, \ldots, X_\ell$ quantified existentially representing a run, that is, labelling of the vertices by states. A vertex v belongs to the set X_i if and only if the run assigns the states q_i to v. Thus the set variables will form a partition of the set of vertices of the graph. The required MSO formula will be of the form $\exists X_1 \ldots \exists X_\ell \varphi$. The formula φ will be a conjunction of several formulas, which we describe below.

1. First of all, we need to state that $X_1, \ldots, X_\ell$ forms a partition of the vertices.

$$\varphi_1 \equiv \forall x \bigvee_{i \in \{1,\ldots,\ell\}} x \in X_i \wedge \bigwedge_{i,j \in \{1,\ldots,\ell\} : i \neq j} \neg(x \in X_i \wedge x \in X_j)$$

Before giving the further formulas, we will use some macros for convenience. For each tile $\tau = (\{u_1, \ldots u_m\}, E, \beta, \alpha)$ we define a macro $\mathsf{subiso}_\tau(x_1, \ldots, x_m)$. The formula $\mathsf{subiso}_\tau(x_1, \ldots, x_m)$ will be true if an only if the induced subgraph on the interpretations of $x_1, \ldots, x_m$, under the state labelling as guessed by $X_1, \ldots, X_\ell$, is isomorphic to τ.

$$\begin{aligned}\mathsf{subiso}_\tau(x_1, \ldots, x_m) \equiv \quad & \bigwedge_{i,j \in \{1\ldots m\} : (u_i,u_j) \in E_b} E_b(x_i, x_j) \wedge \\ & \bigwedge_{i,j \in \{1\ldots m\} : (u_i,u_j) \notin E_b} \neg E_b(x_i, x_j) \wedge \\ & \bigwedge_{i \in \{1\ldots m\} : u_i \in P_{a,q_j}} P_a(x_i) \wedge x_i \in X_j \wedge \\ & \bigwedge_{i \in \{1\ldots m\} : u_i \in P_{q_j}} x_i \in X_j\end{aligned}$$

We also define a macro for a weaker version of the subiso_τ where the state labels are discarded.

$$\mathsf{wsubiso}_\tau(x_1,\ldots,x_m) \equiv \bigwedge_{i,j\in\{1\ldots m\}:(u_i,u_j)\in E_b} E_b(x_i,x_j) \;\wedge \bigwedge_{i,j\in\{1\ldots m\}:(u_i,u_j)\notin E_b} \neg E_b(x_i,x_j) \;\wedge \bigwedge_{i\in\{1\ldots m\}:u_i\in P_{a,q_j}} P_a(x_i)$$

2. Next we state that every vertex is indeed covered by a some tile.

$$\varphi_2 \equiv \forall x \bigvee_{\tau=(\{u_1,\ldots u_m\},E,\beta,\alpha)\in\Delta} \exists x_1,\ldots,x_m \bigvee_{i\in\{1,\ldots,m\}} x = x_i \;\wedge\; \mathsf{subiso}_\tau(x_1,\ldots,x_m)$$

3. Next we state that every induced subgraph that is isomorphic to a tile (discarding state labels) must actually match a tile from Δ.

$$\varphi_3 \equiv \bigwedge_{\tau=(\{u_1,\ldots u_m\},E,\beta,\alpha)\in\Delta} \forall x_1 \ldots \forall x_m \left(\mathsf{wsubiso}_\tau(x_1,\ldots,x_m) \implies \bigvee_{\tau'=(u_1,\ldots u_m,E,\beta,\alpha')} \mathsf{subiso}_{\tau'}(x_1,\ldots,x_m)\right)$$

4. Next we need to state that the constraint C is satisfied. For each constraint of the form *there are* $\geq n$ *copies of the transition* $\tau = (\{u_1,\ldots u_m\},E,\beta,\alpha) \in \Delta$, we have a formula as follows.

$$\varphi_4 \equiv \exists x_1^1 \ldots \exists x_m^1 \; \exists x_1^2 \ldots \exists x_m^2 \ldots \exists x_1^n \ldots \exists x_m^n \bigwedge_{i\in\{1\ldots n\}} \mathsf{subiso}_\tau(x_1^i,\ldots,x_m^i) \bigwedge_{i,j\in\{1\ldots n\}:i\neq j} \bigvee_{k\in\{1\ldots m\}} x_k^i \neq x_k^j$$

The formula essentially says that there are n induced subgraphs which are isomorphic to τ. Further, they are pairwise distinct: they must differ in at least one component. The boolean combinations of the constraints is then expressed by performing the corresponding boolean combinations on the respective formulas. Let φ_5 describe the boolean combinations required for the constraint C.

The required existential MSO formula describing the language of A is

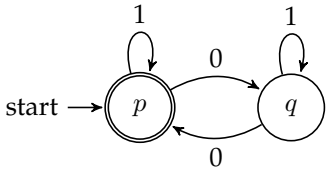
$$\exists X_1 \ldots \exists X_\ell \;\; \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_5$$

. □

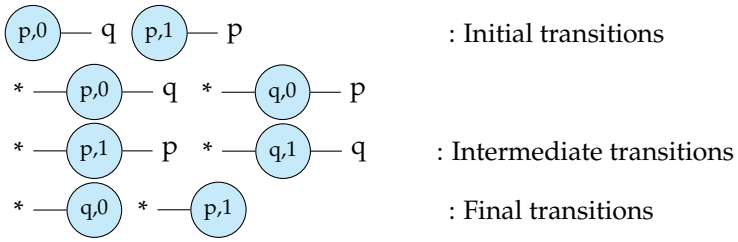
3.3 Example of a Graph Automata

Consider the finite state automaton $A = (Q,\Sigma,\delta,q_{in},F)$ where Q is the set of states, $\Sigma = \{0,1\}$, q_{in} is the initial state and $F \subseteq Q$ is the set of final states and δ is the transition function as

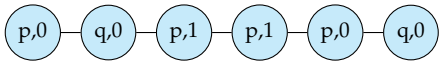
given in the diagram below.



We can give the a graph automata $\mathcal{A} = (Q', \Delta, C)$ which recognizes the set of words accepted by the above DFA, seen as a set of line graphs, where $Q' \subseteq (Q \times \Sigma) \cup Q$, C is the trivial constraint (i.e., true) and Δ is the following set of transitions:



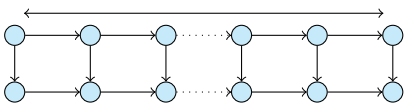
The word 001100 (seen as a line graph) can be tiled as,



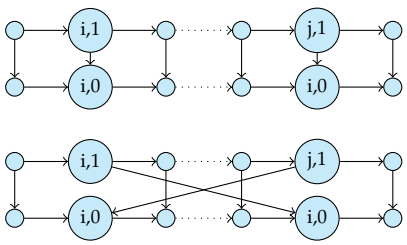
3.3.1 Graph Automata for Grids

The set of Grids $G_{m,2}$ (Ladders) is not *e-recognizable*

Assume there is an *elementary graph acceptor* $\mathcal{A} = (Q, \Delta, C)$ which accepts exactly the set of grids $G_{m,2} = (\{1, \ldots, m\} \times \{0, 1\}, E_1, E_2)$, where the edge sets E_1 and E_2 are given by $((x, y), (x + 1, y)) \in E_1$ for $1 \leq x < m$, $0 \leq y \leq 1$, $((x, 0), (x, 1)) \in E_2$ for $1 \leq x < m$. Degree of all vertices in $G_{m,2}$ are 3 except of the corner vertices which has degree 2. So, the number of possible tiles except the corner tiles in Δ with single core are, $|Q|^4$. The number of possible pair of tiles are, $(|Q|^4)^2$. If we choose $m > (|Q|^4)^2 + 2$, a successful run of $\mathcal{A}$ on $G_{m,2}$ will have the same pair of tiles on a vertex pair $((i, 0), (i, 1))$ and on $((j, 0), (j, 1))$. Now we change the considered grid by modification of two edges. We replace the edge $((i, 0), (i, 1))$ by $((i, 0), (j, 1))$, and replace $((j, 0), (j, 1))$ by $((j, 0), (i, 1))$. This new graph is not isomorphic to a grid $G_{m,2}$ but still accepted by $\mathcal{A}$, which is a contradiction to the fact that $\mathcal{A}$ accepts exactly the set of grids $G_{m,2}$. [10]

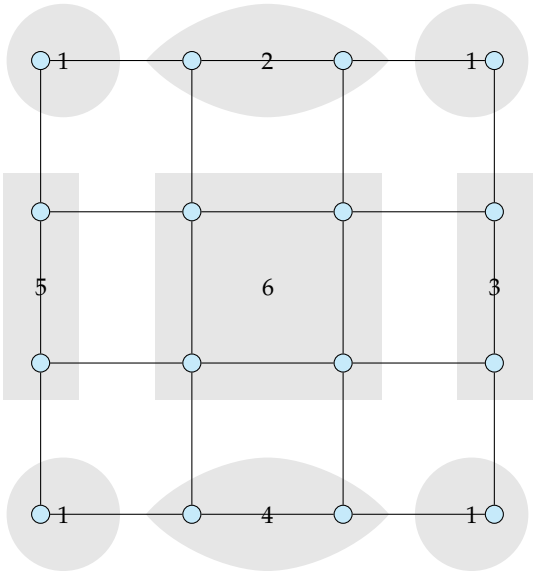


For $m > (|Q|^4)^2 + 2$,

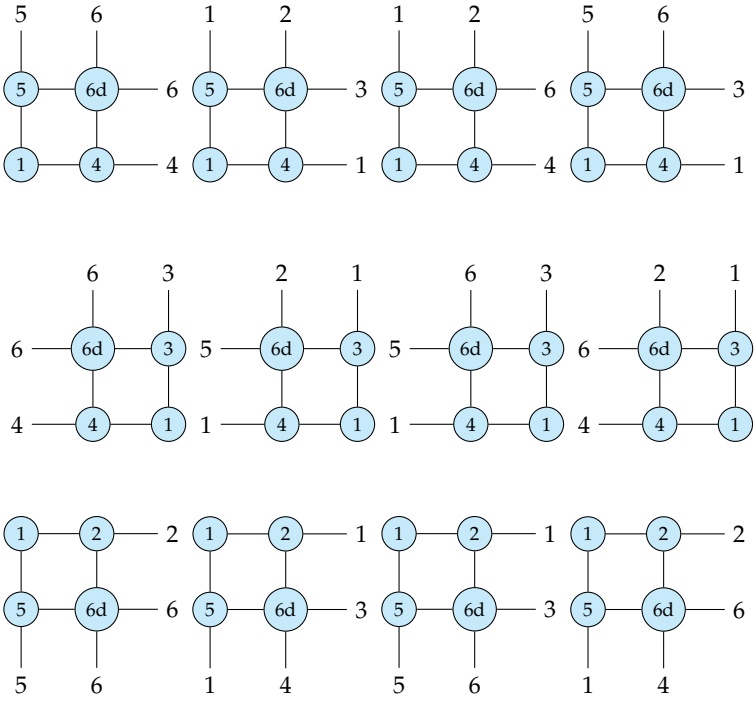


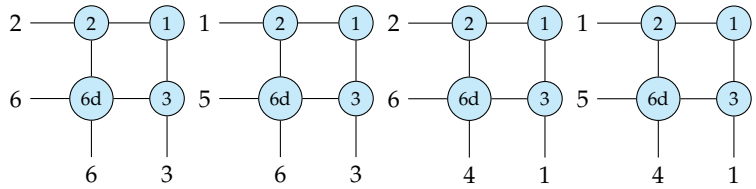
The set of all grids are *t-recognizable*

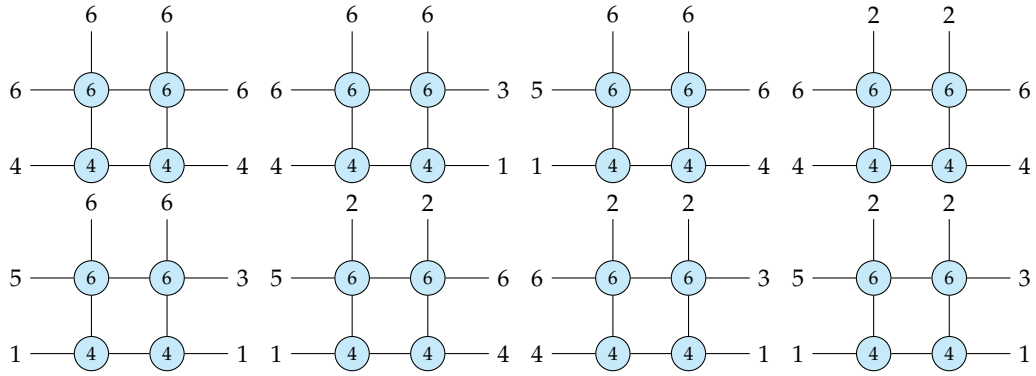
We consider the grids $G_{m,n}$ with $m, n \geq 2$. We can give a graph automaton with 11 states, where 6 states to be used respectively for the corners, top vertices, right vertices, bottom vertices, left verices and all remaining vertices inside and 5 more states to maintain the diagonal information [[10]]. The required constraint is : "each corner tile should appear ≥ 1 time". Below the state assignments are shown in a 4×4 grid (the diagonal informations are not shown here).

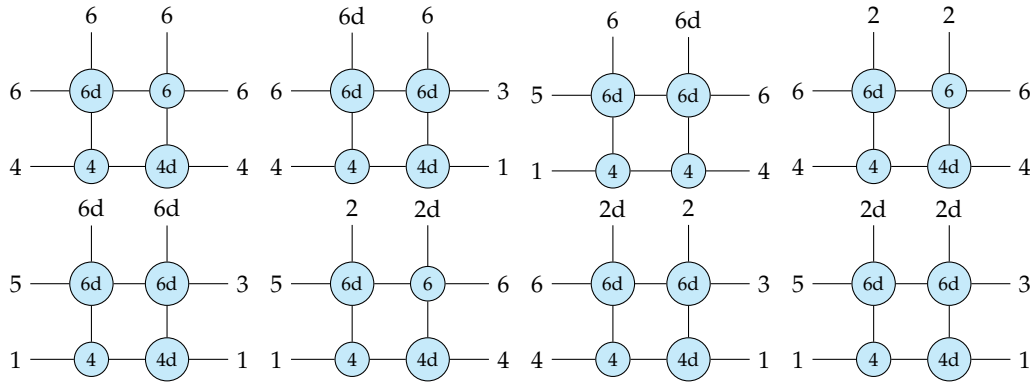


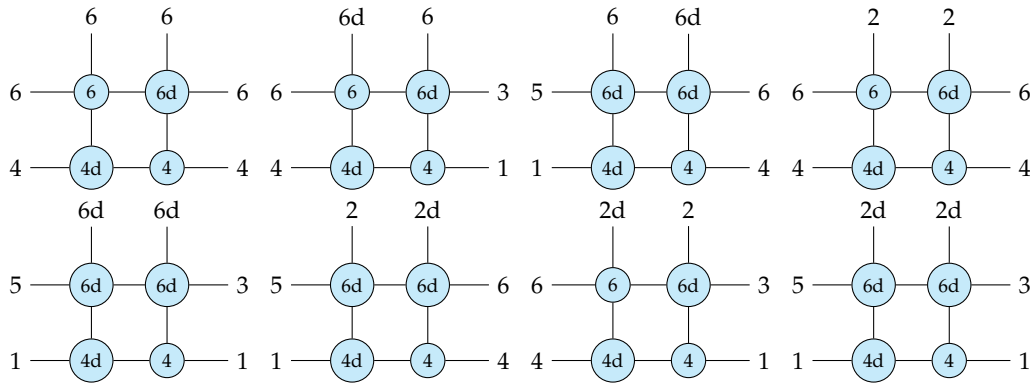
The core of each transition is a square of four vertices. Formally, the graph automaton is given as, $\mathcal{A} = (Q, \Delta, C)$, where $Q = \{$**1,2,3,4,5,6,2d,3d,4d,5d,6d**$\}$, $C = \phi$ and the transitions Δ are given below (We have not considered the diagonal conditions while showing the tiles, as there will be many; we have shown them for some of the tiles only):

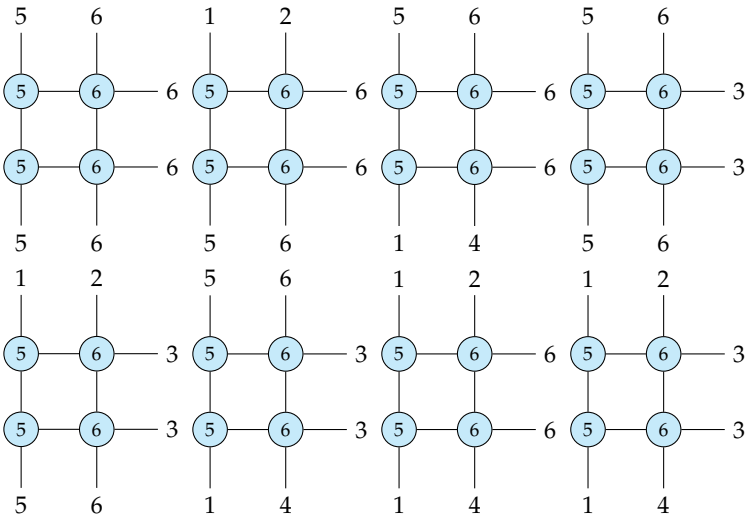


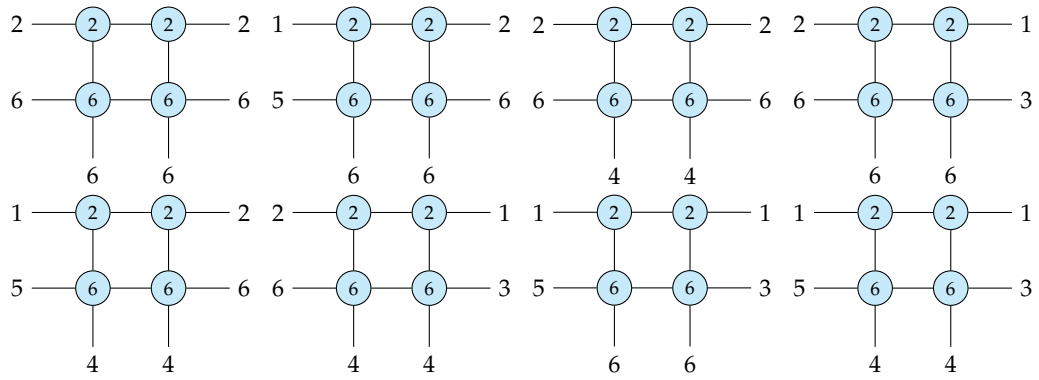

2 2 1 1
6 6d 3 5
6 3
1 2 1 2
5 6d 3 6
6 3
2 2 1 1
6 6d 3 5
4 1
1 2 1 1
5 6d 3 5
4 1

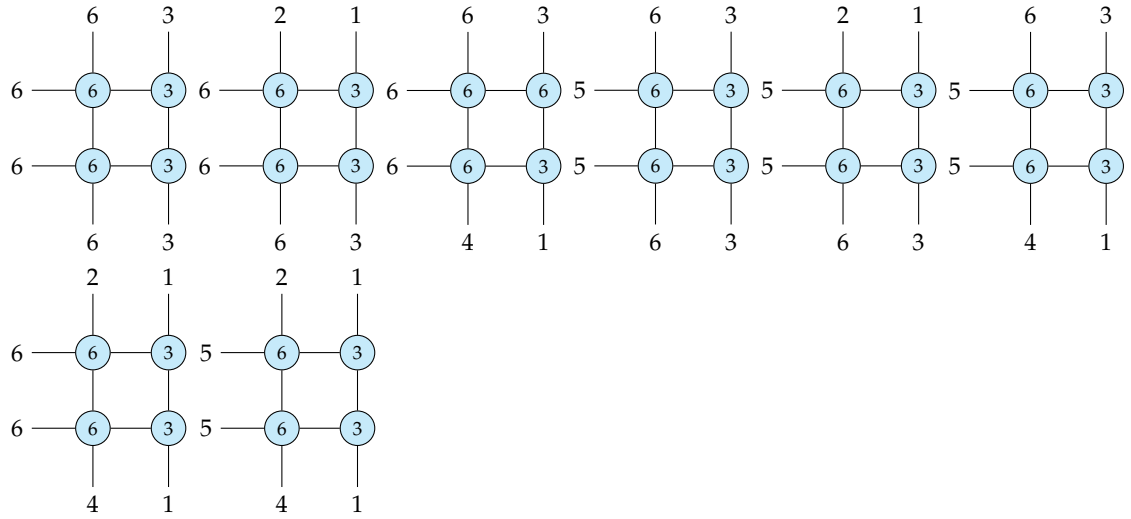

6 6 6 6
6 6 6 6 6 6 6 6
4 4 4 4
6 6 6 6
6 6 6 3 5 6 6 6
4 4 4 4
6 6 6 6
1 4 4 4
2 2 2 2
6 6 6 6
4 4 4 4
6 6 2 2
5 6 6 3 5 6 6 6
1 4 4 1 1 4 4 4
2 2 2 2
6 6 6 3 5 6 6 3
4 4 4 1 1 4 4 1
2 2 2 2
6 6 6 3 5 6 6 3
4 4 4 1 1 4 4 1

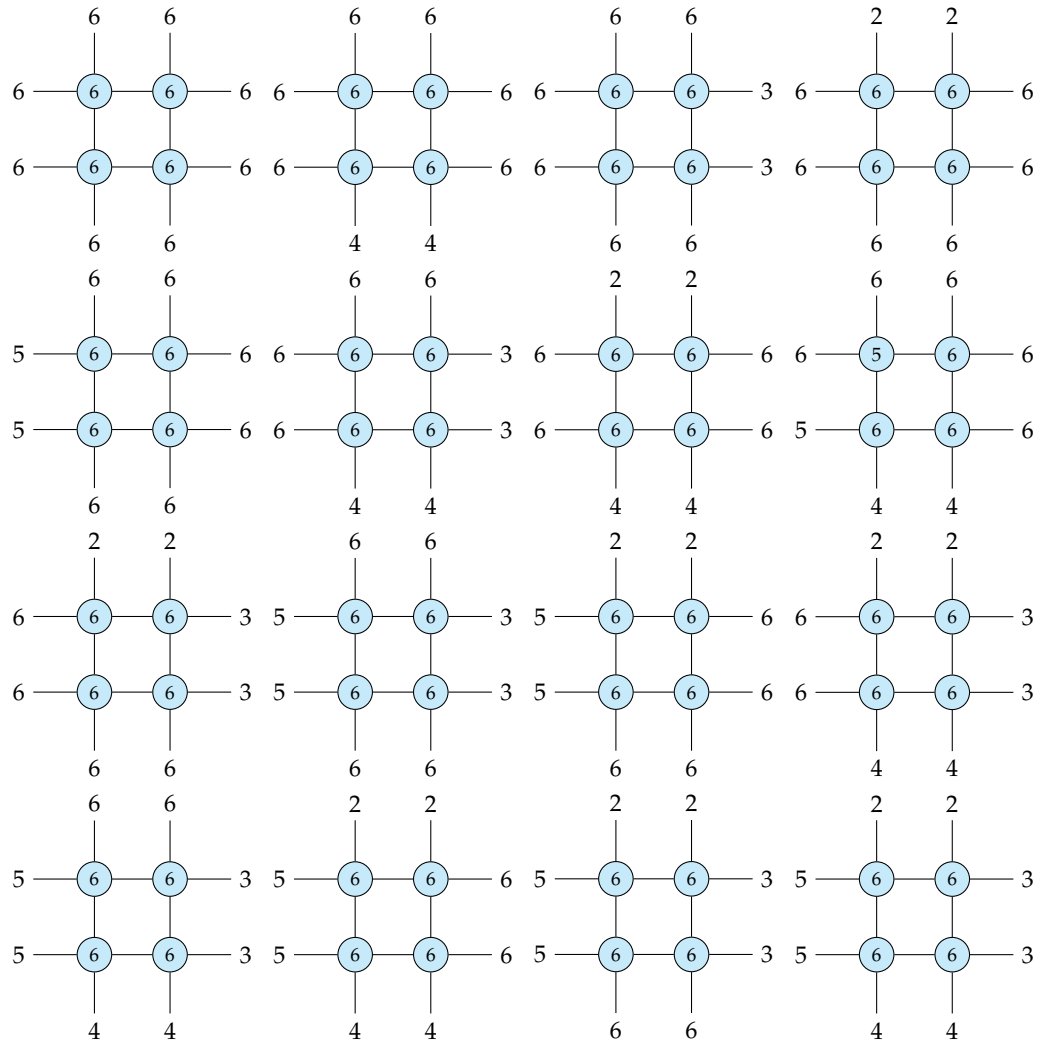

6 6 6d 6
6 6d 6 6 6d 6d 3
4 4 4d 4 4 4 4d 1
6 6 6d 6
6 6d 6 6 6d 6d 3 5 6d 6d 6
4 4 4d 4 4 4 4d 1
6d 6d 2 2d
5 6d 6d 3 5 6d 6 6
1 4 4d 1 1 4 4d 4
2 2 2 2
6 6d 6 6
4 4 4d 4
2d 2 2d 2d
6 6d 6d 3 5 6d 6d 3
4 4 4d 1 1 4 4d 1

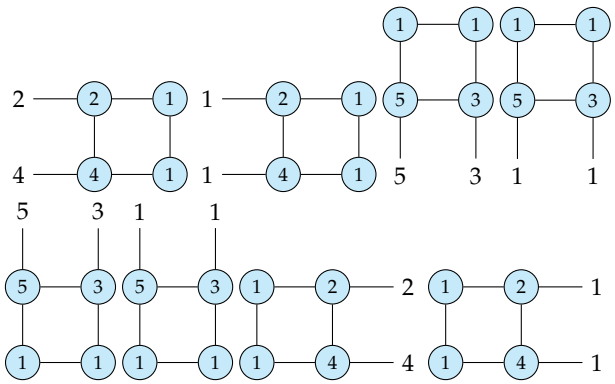

6 6 6d 6
6 6 6d 6 6 6d 6d 3
4 4d 4 4 4 4d 4 1
6 6 6d 6
6 6d 6 6 6d 6d 3 5 6d 6d 6
4 4d 4 4 4 4d 4 1
6d 6d 2 2d
5 6d 6d 3 5 6d 6 6
1 4d 4 1 1 4d 4 4
2 2 2 2
6 6d 6 6
4 4d 4 4
2d 2 2d 2d
6 6d 6d 3 5 6d 6d 3
4 4d 4 1 1 4d 4 1

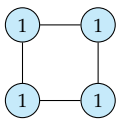

5 6 1 2 5 6 5 6
5 6 6 5 6 6 5 6 6 5 6 3
5 6 6 5 6 6 5 6 6 5 6 3
5 6 5 6 1 4 5 6
1 2 5 6 1 2 1 2
5 6 3 5 6 3 5 6 6 5 6 3
5 6 3 5 6 3 5 6 6 5 6 3
5 6 1 4 1 4 1 4


2 2 2 1 2 2 2 1
6 6 6 6 5 6 6 6 6 6 6 6 6 6 6 3
6 6 6 6 4 4 6 6
1 2 2 2 2 1 1 1
5 6 6 6 6 6 6 3 5 6 6 3 5 6 6 3
4 4 4 4 6 6 4 4


6 3 2 1 6 3 6 3 2 1 6 3
6 6 3 6 6 3 6 6 6 5 6 3 5 6 3 5 6 3
6 6 3 6 6 3 6 6 3 5 6 3 5 6 3 5 6 3
6 3 6 3 4 1 6 3 6 3 4 1
2 1 2 1
6 6 3 5 6 3
6 6 3 5 6 3
4 1 4 1





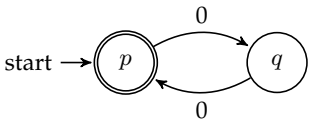


We show that $\mathcal{A}$ accepts exactly the set of grids by induction (Ref. [10]).

Proof hint: First we show that all grids $G_{m,n}$ are accepted by $\mathcal{A}$ by induction on m and n.
Base case: $G_{2,2}$ is accepted by $\mathcal{A}$ as one of the last tile from above matches it.
Inductive case: Suppose $G_{k,k'}$ is accepted by $\mathcal{A}$, i.e., there is a valid tiling for $G_{k,k'}$. We show that there is a valid tiling for $G_{k+1,k'}$ and $G_{k,k'+1}$. From the tiling of $G_{k,k'}$ we replace the tiles with core $(6,4,6,4)$ by suitable tiles with core $(6,6,6,6)$ which has 4 as non-core vertices. Similarly, we replace the tiles of the top-right and bottom right with suitable tiles and match the previously removed tiles again. This is a valid tiling for $G_{k,k'+1}$. In the same way we cane give a valid tiling for $G_{k+1,k'}$. We do not consider the diagonal informations here. Similar proof can be given by considering the diagonal information as well.
The reverse direction can also be proved by induction on number of tiles. We omit the detailed proof here as it will be very long and can be easily done by the reader..

3.3.2 More Examples on Grids

1. **Grids of even height :**
   Suppose the graph automaton $\mathcal{A} = (Q, \Delta, C)$ accepts grids and the following finite state automaton $\mathcal{A}' = (\Sigma, Q', \delta, q_{in}, Q_f)$ accepts words of even length, where $\Sigma = \{0\}$, $Q' = \{p, q\}$, $q_{in} \in Q'$ and $Q_f \subseteq Q'$.

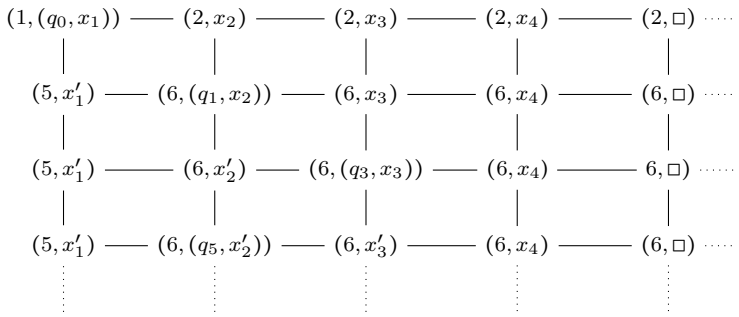
   We give a graph automata $\mathcal{A}'' = (Q'', \Delta', C')$ where $Q'' \subseteq Q \times Q'$ and Δ' is same as Δ except the states are tuples where the first component is same as Δ and the second component alternates in each rows, i.e., if states of the ith row is (x, p) where $x \in \Delta$, the states of the $(i+1)$th row is (y, q) where $y \in \Delta$. The first row of of the grid should have states (x, p) where $x \in \{1, 2\}$ and the last row should have states (y, q) where $y \in \{1, 4\}$.

2. **Grids where each row belong to some regular language on unary alphabet :**
   Suppose the graph automaton $\mathcal{A} = (Q, \Delta, C)$ accepts grids and the following finite state automaton $\mathcal{A}' = (\Sigma, Q', \delta, q_{in}, Q_f)$ accepts some regular language, where $\Sigma = \{0\}$, $q_{in} \in Q'$ and $Q_f \subseteq Q'$. We give a graph automata $\mathcal{A}'' = (Q'', \Delta', C')$ where $Q'' \subseteq Q \times Q'$ and Δ' is same as Δ except the states are tuples where the first component is same as Δ and the second component follows the transitions of $\mathcal{A}'$, i.e., for all i, j, if states of the vertex (i, j) of the grid is (x, y) where $x \in Q$ and $y \in Q'$, the states of the vertex $(i, j+1)$ is (x', y') where $x' \in Q$ and $y' = \delta(y, 0)$. The first column of of the grid should have states (x, y) where $x \in \{1, 5\}$ and $y = q_{in}$ and the last column should have states (w, z) such that $w \in \{1, 3\}$ and $z \in Q_f$.

3.4 Emptiness and Finiteness of Graph Automata is undecidable

The halting problem for Turing machines can be coded in the emptiness problem for grids. An $m \times n$ grid $G_{m,n}$ can encode the run of a Turing machine, where m is the space required by the Turing machine and n represents the runtime of the Turing machine. Suppose we have a graph automata $\mathcal{A} = (Q, \Delta, C)$ that accepts grids and a Turing machine $M = (Q', \Sigma, \mathcal{T}, \delta, q_i n, q_{acc}, q_{rej})$. We can give a graph automata $\mathcal{A}' = (Q'', \Delta'', C'')$ that encode the run of the Turing machine M in a grid $G_{m,n}$. $Q'' \subseteq (Q \times Q' \times (\Sigma \cup \mathcal{T})) \cup (Q \times (\Sigma \cup \mathcal{T}))$. Each row of the grid represents one configuration of the Turing machine. Exactly one node in a row will have state $x = (a, b, c)$ such that $x \in Q \times Q' \times (\Sigma \cup \mathcal{T})$. This node represents the head position of the Turing machine. All other nodes in that row will be labelled by states $y = (p, q)$ such that, $\{y \in Q \times (\Sigma \cup \mathcal{T})\}$. If $x = (a, b, c)$ is the (i, j)th node of the grid then the node $y = (a', b', c')$ at $(i+1, j-1)$ belongs to $Q \times Q' \times (\Sigma \cup \mathcal{T})$ if the tape head of M goes to

left after reading c at state b where $b' = \delta(b, c)$, a and a' are the states according to the transitions of the grid. The node $z = (p, q)$ at $(i + 1, j)$ belongs to $Q \times (\Sigma \cup \mathcal{T})$ where q is the alphabet written by M after reading c at b and p is according to the transitions of the grid. Similarly, the node $y = (a'', b'', c'')$ at $(i + 1, j + 1)$ should belong to $Q \times Q' \times (\Sigma \cup \mathcal{T})$ if the tape head goes to right after reading c at state b where $b'' = \delta(b, c)$, a and a'' are the states according to the transitions of the grid. The node $w = (r, s)$ at $(i + 1, j)$ belongs to $Q \times (\Sigma \cup \mathcal{T})$ where s is the alphabet written by M after reading c at b and r is according to the transitions of the grid. All other nodes of the row $(i + 1)$ is same as that of the row i. Below is an example to illustrate this.



So, halting problem can be reduced to the emptiness problem of the graph automata that accepts grids. Hence, emptiness problem for graph automata is undecidable. Similarly, finiteness problem for Turing machines can be reduced to the finiteness problem of the graph automata that accepts grids. Hence, finiteness problem for graph automata is undecidable.

Chapter 4

Nested Words

4.1 Introduction

In this chapter we restrict the graph automata model on nested words and multiply nested words. A nested word consists of a sequence of linearly ordered positions, augmented with nesting edges connecting calls to returns. The edges do not cross creating a properly nested hierarchical structure. We study finite-state automata as acceptors of nested words. A nested word automaton (NWA) is similar to a classical finite state word automaton, and reads the input from left to right according to the linear sequence. At a call, it can propagate states along both linear and nesting outgoing edges, and at a return, the new state is determined based on states labeling both the linear and nesting incoming edges. [1] has shown that the resulting class of regular languages of nested words has all the appealing theoretical properties that the regular languages of words and trees enjoy. The deterministic nested word automata are as expressive as their nondeterministic counterparts. The class is closed under union, intersection, complement, concatenation, Kleene-$*$, and prefix-closure [1]. The class is also closed under nesting-respecting language homomorphisms, which can model tree operations. We will see that the decision problems such as emptiness and finiteness are decidable. In [1] they have considered nesting words with pending edges as well but here we consider only complete nested words.

4.2 Nested Word

Given a linear sequence, we add hierarchical structure using edges that are well nested (that is, they do not cross).

Definition 4.2.1 A *matching relation* $\rightsquigarrow$ of length ℓ, for $\ell \geq 0$, is a subset of $\{1, 2, \ldots \ell\} \times \{1, 2, \ldots, \ell\}$ such that

1. Nesting edges go only forward: if $i \rightsquigarrow j$ then $i < j$;
2. No two nesting edges share a position: for $1 \leq i \leq \ell$, $|\{j \mid i \rightsquigarrow j\}| \leq 1$ and $|\{j \mid j \rightsquigarrow i\}| \leq 1$;
3. Nesting edges do not cross: if $i \rightsquigarrow j$ and $i' \rightsquigarrow j'$ then it is not the case that $i < i' \leq j < j'$.

When $i \rightsquigarrow j$ holds, for $1 \leq i \leq \ell$, the position i is called a call position, and the unique position j such that $i \rightsquigarrow j$ is called its *return-successor*. Similarly, when $i \rightsquigarrow j$ holds, for $i \leq j \leq \ell$, the position j is called a *return position*, and the unique position i such that $i \rightsquigarrow j$ is called its *call-predecessor*. Our definition requires that a position cannot be both a call and a return. A position $1 \leq i \leq \ell$ that is neither a call nor a return is called *internal*. For $1 \leq i < \ell$, there is a linear edge from i to $i + 1$. Note that a call has indegree 1 and outdegree 2, a return has indegree 2 and outdegree 1, and an internal has indegree 1 and outdegree 1. For matched call positions i, there is a *nesting edge* from i to its return-successor. We call such graphs corresponding to matching relations as *nested sequences*. A *nested word* n over an alphabet Σ is a pair $(a_1 \ldots a_\ell, \rightsquigarrow)$, for $\ell \geq 0$, such that a_i, for each $1 \leq i \leq \ell$, is a symbol in Σ, and $\rightsquigarrow$ is a matching relation of length ℓ. In other words, a *nested word* is a *nested sequence* whose positions are labelled with symbols in alphabet Σ. Let us denote the set of all nested words over as $NW(\Sigma)$. A *language* of nested words over Σ is a subset of $NW(\Sigma)$.

4.3 Nondeterministic Nested Word Automata (NWA)

Now we define finite-state acceptors over nested words as given by [1].

Definition 4.3.1 A nondeterministic nested word automaton (NWA) A over an alphabet Σ is a structure $(Q, Q_0, Q_f, \delta_c^\ell, \delta_i, \delta_r)$ consisting of

- a finite set of (linear) states Q,
- a set of linear initial states $Q_0 \subseteq Q$,
- a set of linear final states $Q_f \subseteq Q$,
- a call-transition function $\delta_c^\ell \subseteq Q \times \Sigma \times Q \times Q$,
- an internal-transition function $\delta_i \subseteq Q \times \Sigma \times Q$, and
- a return-transition function $\delta_r \subseteq Q \times Q \times \Sigma \times Q$.

The automaton A starts in some initial state, and reads the nested word from left to right according to the linear order. The state is propagated along the linear edges as in case of a standard word automaton. However, at a call, the nested word automaton can propagate a state along the outgoing nesting edge also. At a return, the new state is determined based on the states propagated along the linear edge as well as along the incoming nesting edge. The run is accepting if the final linear state is accepting.

The language $L(A)$ of an NWA A is the set of nested words it accepts. The notion of regularity is defined using acceptance by finite-state automata: A language L of nested words over Σ is regular if there exists a NWA A over Σ such that $L = L(A)$.

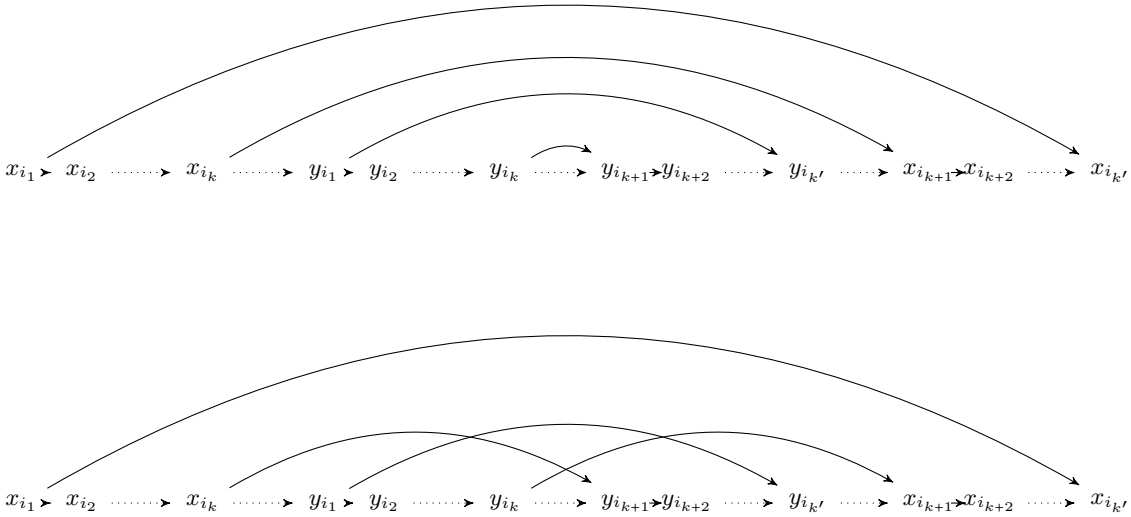
A *run* r of A over a nested word $n = (a_1 \dots a_\ell, \rightsquigarrow)$ is a sequence $q_i \in Q$, for $0 \le i \le \ell$, of states corresponding to linear edges, and a sequence $p_i \in P$, for calls i, of hierarchical states corresponding to nesting edges, such that $q_0 \in Q_0$, and for each position $1 \ge i \ge \ell$,

- if i is a call matched with return at j, then $(q_{i-1}, a_i, q_i, q_j) \in \delta_c$;
- if i is an internal, then $(q_{i-1}, a_i, q_i) \in \delta_i$;
- if i is a matched return with call-predecessor j then $(q_{i-1}, q_j, a_i, q_i) \in \delta_r$.

The run is accepting if $q_\ell \in Q_f$. The automaton A accepts the nested word n if A has some accepting run over n. The language $L(A)$ is the set of nested words it accepts. [1] has shown that given a nondeterministic linearly-accepting NWA A, one can effectively construct a deterministic linearly-accepting NWA B such that $L(B) = L(A)$.

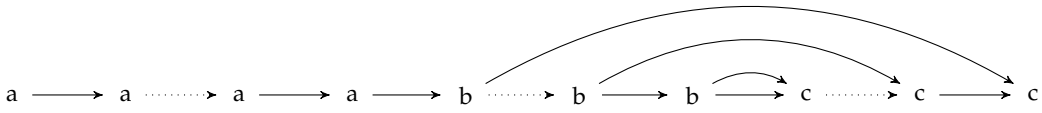
4.4 There Exists no GA that Accepts Exactly the Set of Nested Words

Suppose we can give a graph automaton $\mathcal{A} = (Q, \Delta, C)$ which accepts the set of nested words. Let the maximum number of nodes in any transitions of $\mathcal{A}$ be p. So, number of possible transitions are $|Q|^p$. If we choose $n > 2|Q|^p$, a successful run of $\mathcal{A}$ on a nested word w will have the same pair of transitions on the vertices $x_{i_1}, \dots, x_{i_k}, x_{i_{(k+1)}}, \dots, x_{i_{k'}}$ and $x_{j_1}, \dots, x_{j_k}, x_{j_{(k+1)}} \dots, y_{j_{k'}}$, where $k \le p$ and $i_k < j_1 < j_{k'} < i_{(k+1)}$. Suppose there is an edge incident on $x_{i_{(k+1)}}$, so there is another edge incident on $y_{i_{(k+1)}}$. Now we change the considered nested word by replacing the edge incident on $x_{i_{(k+1)}}$ by the edges incident on $y_{i_{(k+1)}}$ and by replacing the edges incident on $y_{i_{(k+1)}}$ by the edges incident on $x_{i_{(k+1)}}$. This new graph is not isomorphic to a nested word as it has crossings but still accepted by $\mathcal{A}$, which is a contradiction to the fact that $\mathcal{A}$ accepts L. This is elaborated in the following diagram.

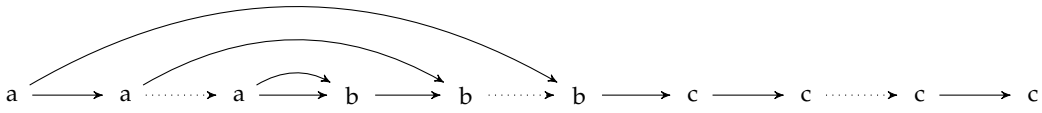


4.5 How NWL is Different from CFL

Given a context free languae L, it can be expressed as nested word language by homomorphism. But nested word languages are closed under intersection ([]) but the context free languages are not. Consider the two context free languages $L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$ and $L_2 = \{a^m b^n c^n \mid n, m \geq 0\}$. It is clear that $L = L_1 \cap L_2$ is not context free. Here L is non-empty. If we see them as nested words we get some extra information about the structure. The family of nested words L_1' corresponding to L_1 is the following :



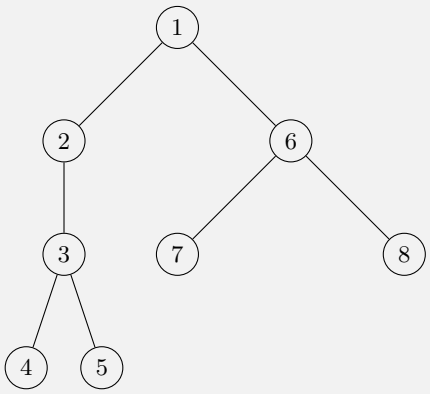
Which means, we push a symbol on a and pop a symbol on b and on c there are internal transitions. Similarly, the family of nested words L_2' corresponding to L_2 is the following :

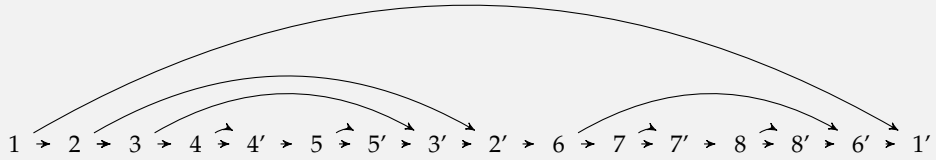


Which means, there are internal transitions on a and we push a symbol on b and pop a symbol on c. The intersection of L_1' and L_2' is ϕ, which is again a nested word.

OBSERVATIONS

Nested words are very similar to trees. We can map all trees as nested words. And there can be many tree interpretations of a nested word. One mapping from tree to nested word is illustrated below with an example.





Each node x of the tree corresponds to two nodes x and x' in the nested word, where x and x' are connecting by nesting edge. If a node x in the tree has children $y_1,\ldots,y_k$ then, in the nested word, $i_x < i_{y_1} < i_{y_1'} < \cdots < i_{y_k} < i_{y_k'} < i_{x'}$. Thus the parent-child relation and order among the siblings in tree are maintained in the nested word. Ordered trees, Binary trees, ranked trees, unranked trees etc. can be represented by trees. Word operations such as prefixes, suffixes, concatenation, reversal, as well as tree operations referring to the hierarchical structure, can be defined naturally on nested words.

4.6 Emptiness and finiteness of 1-NW is decidable

Nonempty nested words can be generated by the following context free grammar:

$$S ::= a \quad | \quad a \overset{\frown}{\longrightarrow} b \quad | \quad a \overset{\frown}{\longrightarrow} S \longrightarrow b \quad | \quad S \longrightarrow S$$

We can check if this grammar generates finite language or not. The only way a grammar with no useless productions can generate an infinite set of words is by repetition. We start at S and keep expanding until we hit a non-terminal that we have already expanded. So now we can copy the subtree corresponding to that non-terminal and paste it. We can do this forever so it must be that the grammar generates infinitely many words.
We can give a directed graph to represent a context-free grammar, where the non-terminals of the grammar are the vertices of the directed graph. There is an edge from some node X to some node Y

in the directed graph iff there is a production rule in the grammar where X is in the l.h.s. and Y is in the r.h.s. Now, the grammar generates an infinite set if and only if the graph is cyclic.

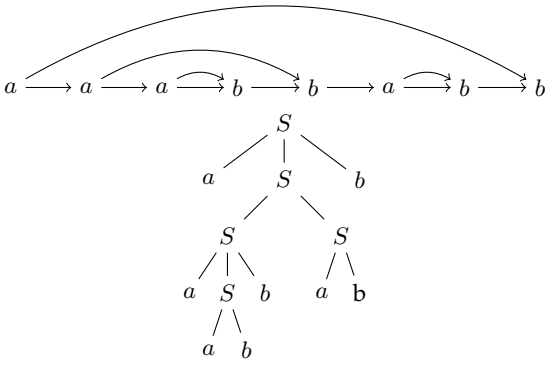
4.7 Finiteness of 1-NW is decidable : Tree automata approach

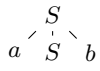
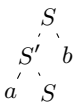
Here we give another approach to show that the finiteness of 1-NWs is decidable.
We can give a tree-interpretation of 1-Nested Words using the grammar described in the above section as the following.

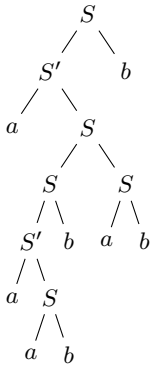
Tree Interpretation : Parse Trees of NWs

- If x is a node in the NW, x is a leaf node in the tree.
- If x and y are connected by a nesting edge in the NW, x and y have the same parent in the tree.
- If x and y are connected by a linear edge in the NW, y is the nearest successor in the inorder traversal of the tree which is not labeled by S.

An example of nested word and the corresponding tree interpretation is shown below :



We can change the parse trees to binary trees by replacing each  by 
So, the modified parse tree of the above example is,



We can give a bottom-up tree automaton $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ which accepts exactly the modified parse trees.

- $Q = \{q_a, q_b, q_{acc}, q'_{acc}\}$
- $\Sigma = \{a, b, S, S'\}$

- $Q_f = \{q_{acc}\}$
- Δ is a set of transition rules of the following type:

$$\{a \to q_a, b \to q_b,$$
$$S(q_a, q_b) \to q_{acc}, S'(q_a, q_{acc}) \to q'_{acc}, S(q'_{acc}, q_b) \to q_{acc}, S(q_{acc}, q_{acc}) \to q_{acc}\}$$

Given a MPDS with 1 stack $\mathcal{M}$, we can also obtain a tree automata $\mathcal{A}_\mathcal{M}$ over parse trees of $\mathcal{M}$.

- A leaf node a is labeled by the state (q_1, q_2) if and only if there is a transition $(q_1, a, q_2, q_3, 1) \in \Delta_{push}$.
- A leaf node b is labeled by the state (q_1, q_2) if and only if there is a transition $(q_1, q_3, b, q_2, 1) \in \Delta_{pop}$.
- A leaf node x is labeled by the state (q_1, q_2) if and only if there is a transition $(q_1, x, q_2) \in \Delta_{int}$.
- A binary internal node S at a level above leaves, is labeled by the state (q_1, q_2) if and only if there is a transition $(q_1, a, q, q'_1, 1) \in \Delta_{push}$ and $(q'_1, q, b, q_2, 1) \in \Delta_{pop}$.
- A 3-ary internal node S is labeled by the state (q_1, q_2) if and only if its children from left to right are labeled by (q_1, q'_1), (q'_1, q'_2) and (q'_2, q_2) and there is a transition $(q_1, a, q, q'_1, 1) \in \Delta_{push}$ and $(q'_2, q, b, q_2, 1) \in \Delta_{pop}$.
- A binary internal node S is labeled by the state (q_1, q_2) if and only if its children from left to right are labeled by (q_1, q'_1) and (q'_1, q_2).

We can get similar tree automata $\mathcal{A}_\mathcal{M}$ over modified parse trees also.

From our constructions we can see that for a nested word we may have several parse trees, but number of possible parse trees for a nested word is finite. SO, finitely many NWs implies finitely many parse trees. Given an MPDS $\mathcal{M}$ and an integer k, we can construct a tree automaton $\mathcal{A}_{1,\mathcal{M}} = \mathcal{A} \times \mathcal{A}_\mathcal{M}$ over modified parse trees, such that $\mathcal{A}_{1,\mathcal{M}}$ accepts all the valid parse trees of 1-NWs which have an accepting run in $\mathcal{M}$. Then we can check the finiteness of the tree automata using **Algorithm 1**, which is described below. Hence, finiteness of 1-NWs is decidable.

Deciding language finiteness of tree automata

The finiteness problem for tree automata can be computed in linear-time by fix-point computation. We compute the set of reachable states. If a state is repeated, then the language of the automaton is not finite, otherwise it is finite. We present the following algorithm to check the finiteness of tree automata. All states that occur in a leaf transition are reachable, and if there is a transition $\tau = (q_1, \ldots, q_k, a, q)$ such that $q_1, \ldots, q_k$ are already in the reachable set, then q can be added to the reachable set. We do this by first defining the set $pre(\tau) = q_1, \ldots, q_k$. The algorithm is given in the next page.

Result: **True** if the language of a nondeterministic tree automata is finite otherwise **False**

Input: Nondeterministic tree automata $A = (Q, \Sigma, Q_f, \Delta)$;
$R = \{q \in Q | \exists a \in \Sigma : (a, q) \in \Delta\}$
// All states that occur in leaf transitions are reachable
forall the $\tau = (q_1, \ldots, q_k, a, q) \in \Delta$ **do**
    $pre(\tau) = \{q_1, \ldots, q_k\}$;
    $post(\tau) = q$;
end
while $(R \neq Q) \wedge (f_1 =$ ***True***$)$ **do**
    $f_1 =$ **False**;
    // f_1 is true iff R changes
    forall the $\tau = (q_1, \ldots, q_k, a, q) \in \Delta$ **do**
        if $pre(\tau) \subseteq R$ **then**
            if $post(\tau) \notin R$ **then**
                // If $post(\tau)$ is not introduced before, we add $post(\tau)$ in R
                $R := R \cup post(\tau)$;
                $f_1 =$ **True**;
            else
                $Temp := R$;
                // When a state repeats in R, we need to check if we can reach to an accepting state or not; if we can reach some accepting state that means the language of a nondeterministic tree automata is infinite.
                while $(Temp \cap Q_f = \varnothing) \wedge (f_2 =$ ***True***$)$ **do**
                    $f_2 =$ **False**;
                    // f_2 is true iff R changes
                    forall the $\tau = (q_1, \ldots, q_k, a, q) \in \Delta$ **do**
                        if $pre(\tau) \subseteq Temp$ **then**
                            if $post(\tau) \notin Temp$ **then**
                                $Temp := Temp \cup post(\tau)$;
                                $f_2 =$ **True**;
                            end
                        end
                    end
                end
                if $Temp \cap Q_f \neq \varnothing$ **then**
                    return **True**;
                    // If some accepting state is reached, we return True
                end
            end
        end
    end
end
return **False**;

Algorithm 1: Finiteness of nondeterministic tree automata

Chapter 5

Multiply Nested Words

5.1 Multi-Pushdown System

An MPDS is a finite state of system with finite number of stacks. A transition may push onto a stack, pop from a stack or leave the stack untouched. However, in one transition an MPDS can touch at most one stack. Moreover, the push transition and the pop transition are disjoint.

Definition 5.1.1 A MPDS $\mathcal{M}$ with $s \in \mathbb{N}$ number of stacks is a tuple $(Q, \Sigma, \iota, \Delta, F)$ where,

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

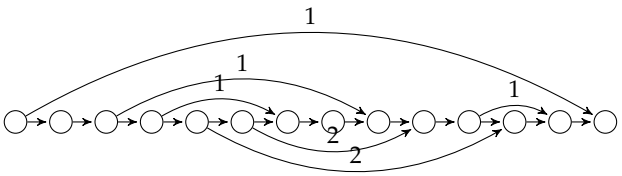
The language of a Multi-pushdown system is a set of multiply nested words, which we define formally below.

Let S be a set. For a binary relation $\mathcal{R} \subseteq S \times S$, we define *support* of $\mathcal{R}$, denoted $\mathsf{supp}(\mathcal{R})$ to be $\{x \in S |$ there is some $y \in S$ such that $(x, y) \in S$ or $(y, x) \in S\}$.

Definition 5.1.2 A *multiply nested word* (MNW) w over Σ is a structure $w = (dom(w), \lambda, <, \rightsquigarrow_1, \ldots, \rightsquigarrow_s)$ where,

- $dom(w)$ is the set of positions
- $\lambda : dom(w) \to \Sigma$ is a node labeling function
- $<$ is a total order on $dom(w)$
- For each $i \in [s]$, where $[s]$ denotes the set $\{1, \ldots, s\}$ for $s \in \mathbb{N}$, $\rightsquigarrow_i \subseteq <$ is a binary relation such that,
  - For $i \neq j$, $\mathsf{supp}(\rightsquigarrow_i) \cap \mathsf{supp}(\rightsquigarrow_i) = \varnothing$
  - For all $i \in [s]$, $x \rightsquigarrow_i y \Rightarrow (\forall z(x \rightsquigarrow_i z \Rightarrow z = y) \wedge (z \rightsquigarrow_i y \Rightarrow z = x))$
  - For all $i \in [s]$, there do not exists $x < x' < y < y'$ such that $x \rightsquigarrow_i y$ and $x' \rightsquigarrow_i y'$

Here is an example of multiply nested word with two stacks:

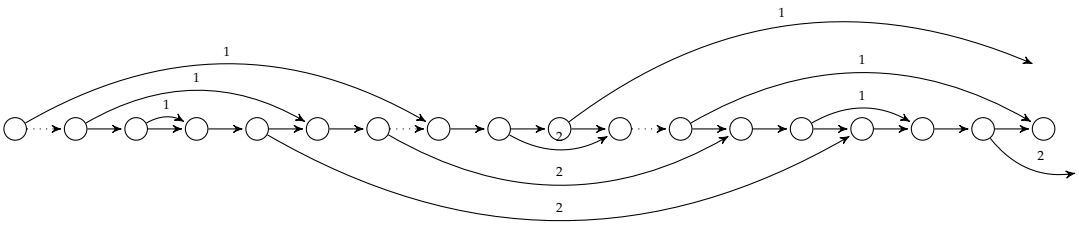


5.2 Emptiness and finiteness of MPDS is undecidable

We can embed a grid inside a multiply nested word. As the emptiness and finiteness of graph automata that accept grids is undecidable, the emptiness and finiteness of multi-pushdown system is also undecidable. Suppose we have a grid $G_{m,n}$ as defined earlier and a multiply nested word $N = (V, \rightarrow, \{\rightsquigarrow_j\}_{j\in\{1,2\}})$, where

- V is the set of vertices.
- $\rightarrow \subseteq V \times V$ is the linear edge relation.
- $\rightsquigarrow_j \subseteq V \times V$ is a nested relation, for every $j \in \{1, 2\}$.

There are $2mn$ number of vertices in N. First m items are pushed in the first stack, then in the next $2m$ vertices one item is popped from the first stack and one item is pushed in the second stack. Again in the next $2m$ vertices one item is popped from the second stack and one item is pushed in the first stack and this continues for n times.
In $G_{m,n}$ if we go from (i, j) to $(i, j + 1)$ that is similar to going from the $(m \times i + m)$th vertex to $(m \times i + m + 2)$th vertex by taking two linear edges in N, (i.e., the path $(m \times i + m) \rightarrow\rightarrow (m \times i + m + 2)$).
In $G_{m,n}$ if we go from (i, j) to $(i + 1, j)$ that is similar to going from the $(m \times i + m)$th vertex to $(m \times (i + 1))$th vertex by taking the path $\rightarrow\rightsquigarrow_1\rightarrow\rightsquigarrow_2$ in N. Similarly,iIn $G_{m,n}$ if we go from $(i + 1, j)$ to (i, j) or $(i, j + 1)$ to (i, j), we can take the corresponding reverse paths. The below image illustrates the nested word.



So, a grid can be embedded in a multiply nested word. Hence, the emptiness and finiteness of multi-pushdown system is undecidable.

5.3 Emptiness of MPDS is decidable when the accepted MNWs have bounded tree-width/split-width

[2] has shown that emptiness of graph automata is decidable, when evaluated over graphs that are definable in MSO and are also of bounded tree-width.

Theorem 5.3.1 [8] The problem of checking, given $k \in \mathbb{N}$ and $\phi \in MSO$ over Σ-labeled graphs, whether there is a Σ-labeled graph G of tree-width at most k that satisfies ϕ, is decidable.

From the **Theorem 3.1.2** we can see that set of graphs accepted by a graph automata is definable in MSO.

So, from Seese's theorem and the **Theorem 3.1.2** by [10], we can state the following corollary : **Corollary 5.3.1** Let C be a class Of MSO definable Σ-labeled graphs. The problem of checking, given $k \in \mathbb{N}$ and a graph automaton GA, whether there is some $G \in C$ of tree-width at most k that is accepted by GA, is decidable.

[6] has given uniform proofs of decidability of emptiness for different under approximations of MPDS, such as bounded context-switching multi-stack automata [7], bounded phase multi-stack automata [5] by showing that they have bounded tree-width. [3] has shown similar results when the set of graphs has bounded split width. To show that they have introduced the notion of *split-width*, which we will define next.

5.3.1 Split-Width

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

Definition 5.3.1 Given an MNW $w = (dom(w), \lambda, <, \rightsquigarrow_1, \ldots, \rightsquigarrow_s)$, an *m-split* of w is a $\bar{w} = (dom(w), \lambda, \rightarrow, \dashrightarrow, \rightsquigarrow_1, \ldots, \rightsquigarrow_s)$, where $\rightarrow \cap \dashrightarrow = \emptyset$, $\rightarrow \cup \dashrightarrow = <$ and $|\dashrightarrow| = m-1$.

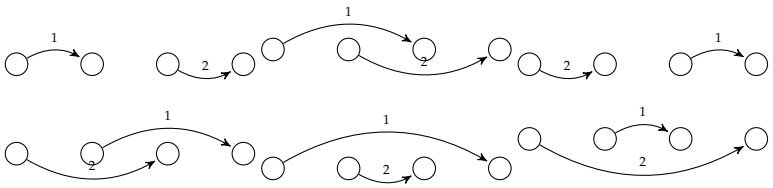
Definition 5.3.2 A *split multiply nested word* (SMNW) is an m-split $\bar{w}$ of some MNW w and some m. We say that $\bar{w}$ is an m-SMNW. An entire multiply nested word is always an 1-SMNW.

Now we define two operations *shuffle* and *merge*.

Definition 5.3.3 *Shuffle :* Let $\bar{u} = (dom(u), \lambda, \rightarrow, \dashrightarrow, \rightsquigarrow_1, \ldots, \rightsquigarrow_s)$ be an m-SNMW and $\bar{v} = (dom(v), \lambda, \rightarrow, \dashrightarrow, \rightsquigarrow_1, \ldots, \rightsquigarrow_s)$ be an n-SNMW. shuffle$(\bar{u}, \bar{v})$ is a set of $(m+n)$-SNMWs, if and only if,

- $dom(w) = dom(u) \uplus dom(v)$
- $\lambda_w = \lambda_u \uplus \lambda_v$
- $\rightarrow_w = \rightarrow_u \cup \rightarrow_v$
- $\rightsquigarrow_w = \rightsquigarrow_u \cup \rightsquigarrow_v$

Below is an example of *shuffle*:



Definition 5.3.4 *Merge :* Let $\bar{(}u) = dom(u), \lambda, \rightarrow, \dashrightarrow, \rightsquigarrow_1, \ldots, \rightsquigarrow_s)$ be an m-SMNW. merge$(\bar{u})$ is a set of n-SNMW obtainted by replacing some $\dashrightarrow$ by $\rightarrow$ in $\bar{u}$.

The following image illustrates the merge operation. $a \overset{\frown}{\dashrightarrow} b \;\rightarrow\; a \overset{\frown}{\longrightarrow} b$

Definition 5.3.5 *k-bounded splits (k-BS) :* The class k-BS is the smallest set of SMNWs closed under the following operations,

- $a \in k$-BS
- $a \overset{\frown}{\dashrightarrow} b \;\in k$-BS
- If $\bar{u}$ is an m-SNMW in k-BS and $\bar{v}$ is an n-SMNW in k-BS, and if $m+n \leq k$, then *shuffle*$(\bar{u}, \bar{v}) \subseteq k$-BS
- If $\bar{u}$ is an m-SNMW in k-BS then *merge*$(\bar{u}) \subseteq k$-BS.

5.4 Finiteness of MPDS is decidable when it has bounded split width

To solve the finiteness problem for MPDS $\mathcal{M}$ with k-bounded split-width, we give a tree representation (proof trees) of multiply nested words. These proof trees can be accepted by a bottom up tree automata

$\mathcal{A}_k$. We can also obtain a tree automata $\mathcal{A}_{\mathcal{M}}$ over proof trees such that it accepts a proof tree iff the corresponding MNW has k-bounded split-width and is accepted by $\mathcal{M}$. The automata $\mathcal{A}_{k,\mathcal{M}} = \mathcal{A}_k \times \mathcal{A}_{\mathcal{M}}$ accepts all the valid proof trees of MNWs in k-bounded split-width which have an accepting run in $\mathcal{M}$. We have already seen in **Algorithm 1** that we can check the language finiteness of tree automata. By applying the algorithm on $\mathcal{A}_{k,\mathcal{M}}$ we can check the language finiteness of multiply nested words with bounded split-width.
First we describe the proof trees.

Proof trees : Let $\bar{w}$ be an SMNW in k-BS.

- the root is labeled by $\bar{w}$.
- leaves are labeled by atomic MNWs.
- if an internal node labeled $\bar{u}$ has only one child labeled $\bar{v}$, then $\bar{u} \in \mathit{merge}(\bar{v})$.
- if an internal node labeled $\bar{u}$ has two children labeled $\bar{x}$ and $\bar{y}$, then $\bar{u} \in \mathit{shuffle}(\bar{x}, \bar{y})$.

Internal nodes of the *proof-tree* are labeled by one of,

$\rightsquigarrow_i$, denotes nesting edge for stack i.

mrg$(m, i_1, \ldots, i_{n-1}, i_n)$, where $1 \le i_1 < \cdots < i_{n-1} < i_n$. This denotes a merge operation on an SMNW with m components resulting in an SMNW with n components.

shf(m, n, f_ℓ, f_r), where $1 \le m < m + n \le k$ and $f_\ell : [m] \to [m + n]$ is a monotone map that describes the positions in shuffled word of the m components coming from the left child and similarly for f_r and the n components coming from the right child.

We can construct a tree automaton $\mathcal{A}_k$ over proof trees of k-BS, where the relations are the states and it checks few properties which are local to a node and its children to check proper shuffle and merge.

This tree automata will have $2^{s \times k^2}$ states.
Given an MPDS $\mathcal{M}$, we can obtain tree automata over proof trees $\mathcal{A}_{\mathcal{M}}$ in the following way:

- All leaves are marked by pair of states (p, q) iff it can take p to q in the MPDS
- A node v labeled $\rightsquigarrow_i$ has a left child v_1 labeled a and a right child v_2 labeled b and the run labels v_1 with (q_2, q_2') and v_2 with (q_1, q_1'). v can be labeled (q_1, q_1', q_2, q_2') iff there exists a state q is a transition $(q_1, a, q_1', q, i) \in \Delta_{push}$ and $(q_2, q, b, q_2', i) \in \Delta_{pop}$ of the MPDS $\mathcal{M}$.
- For a shuffle node the start and end states of each components are inherited from the corresponding child.
- For a merge node v with child v', if components p and $p+1$ from v' are merged, we check that the end state of p equals the start state of $p + 1$ and start state of p and end state of $p + 1$ are inherited to v.

We can construct the above described tree automata with $|Q|^{2k}$ number of states.

Each of the MNW may have several proof trees but the number of proof trees corresponding to an MNW is finite. So, finite number of MNWs implies finite number of proof trees.

Theorem 5.4.1 [3] Given an MPDS $\mathcal{M}$ and an integer k, we can construct in exponential time a tree automaton $\mathcal{A}_{k,\mathcal{M}} = \mathcal{A}_k \times \mathcal{A}_{\mathcal{M}}$ over proof trees, such that $\mathcal{A}_{k,\mathcal{M}}$ accepts all the valid proof trees of MNWs in k-BS which have an accepting run in $\mathcal{M}$.

We can check the language finiteness of the tree automata by **Algorithm 1**. So, the language finiteness of the tree automata is decidable.

Chapter 6

Conclusion

There are many interesting problems connecting graph automata and tree-width that we can look into in future. In this thesis we have mainly studied that given a graph automata (or a multiply nested word automata) the language accepted by it is empty or not and the language accepted by it is finite or not. We may extend this to more complicated graph properties for the accepted set of graphs. One of the open problem is, given a graph automata if we can say the set of graphs accepted by the graph automata has bounded tree-width or not.

Bibliography

[1] Rajeev Alur and P. Madhusudan. "Adding Nesting Structure to Words". In: (2006), pp. 1–13. DOI: 10.1007/11779148_1. URL: https://doi.org/10.1007/11779148_1.

[2] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Vol. 138. Encyclopedia of mathematics and its applications. Cambridge University Press, 2012. ISBN: 978-0-521-89833-1. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

[3] Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. "MSO Decidability of Multi-Pushdown Systems via Split-Width". In: (2012), pp. 547–561. DOI: 10.1007/978-3-642-32940-1_38. URL: https://doi.org/10.1007/978-3-642-32940-1_38.

[4] Reinhard Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2012. ISBN: 978-3-642-14278-9.

[5] Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. "A Robust Class of Context-Sensitive Languages". In: (2007), pp. 161–170. DOI: 10.1109/LICS.2007.9. URL: https://doi.org/10.1109/LICS.2007.9.

[6] P. Madhusudan and Gennaro Parlato. "The Tree Width of Auxiliary Storage". In: POPL '11 (2011), pp. 283–294. DOI: 10.1145/1926385.1926419. URL: http://doi.acm.org/10.1145/1926385.1926419.

[7] Shaz Qadeer and Jakob Rehof. "Context-Bounded Model Checking of Concurrent Software". In: (2005), pp. 93–107. DOI: 10.1007/978-3-540-31980-1_7. URL: https://doi.org/10.1007/978-3-540-31980-1_7.

[8] D. Seese. "The structure of the models of decidable monadic theories of graphs". In: *Annals of Pure and Applied Logic* 53.2 (1991), pp. 169 –195. ISSN: 0168-0072. DOI: http://dx.doi.org/10.1016/0168-0072(91)90054-P. URL: http://www.sciencedirect.com/science/article/pii/016800729190054P.

[9] Paul D. Seymour and Robin Thomas. "Graph Searching and a Min-Max Theorem for Tree-Width". In: *J. Comb. Theory, Ser. B* 58.1 (1993), pp. 22–33. DOI: 10.1006/jctb.1993.1027. URL: https://doi.org/10.1006/jctb.1993.1027.

[10] Wolfgang Thomas. "On Logics, Tilings, and Automata". In: (1991), pp. 441–454. DOI: 10.1007/3-540-54233-7_154. URL: https://doi.org/10.1007/3-540-54233-7_154.