Contents

1 Courcelle's Theorem

Our goal is to prove the following theorem, attributed to Bruno Courcelle.

Theorem 1 (Courcelle's Theorem)**.** *[1] For every $\omega \in \mathbb{N}$, there exists an algorithm which,*

- *upon an input consisting of a graph G, a binary tree-decomposition (T, β) of G, and an* MSO*-sentence φ over $\tau = \{\mathsf{edge}\}$,*
- *decides if $G \models \varphi$*

in time $f(\omega, \mathsf{qr}(\varphi)) \cdot n$.

Recall that there is an fpt-algorithm for approximating treewidth, parameterized by the treewidth of an input graph.

Theorem 2 (Computing treewidth)**.** *[2] There is an algorithm which, given an input consisting of a graph G and $\omega \in \mathbb{N}$, either outputs a tree-decomposition (T, β) of width at most $2\omega + 1$ or correctly decides that the treewidth of G is bigger than ω in time $2^{O(\omega)} \cdot n$.*

Combining Theorem 1 and Theorem 2, together with the fact that any tree-decomposition can be converted into a nice tree-decomposition whose width does not exceed the original one in linear time, we immediately obtain the following result.

Corollary 3. *There is an algorithm which, given an input consisting of a graph G of treewidth at most ω and an* MSO*-sentence φ over $\tau = \{\mathsf{edge}\}$, decides if $G \models \varphi$ in time $f(\omega, \mathsf{qr}(\varphi)) \cdot n$.*

Corollary 4. *Let $\mathcal{G}$ be an* MSO_2*-definable property. Then for any graph G of treewidth at most ω, one can decide if $G \in \mathcal{G}$ in time $f(\omega, \mathcal{G}) \cdot n$.*

Proof: Let φ be an MSO_2-sentence (over $\tau = \{\mathsf{inc}, V, E\}$ where V and E are unary predicates) such that any graph is a member of $\mathcal{G}$ if and only if it satisfies φ. Consider an arbitrary graph G and and recall that the treewidth of the incidence graph G' of G is at most ω. Note that the τ'-structure associated with G' has $V(G')(= V(G) \cup E(G))$ as the universe.

One can obtain an MSO-sentence φ' over the vocabulary $\{\mathsf{edge}\}$ which is equivalent to φ, i.e. $G \models \varphi$ if and only if $G' \models \varphi'$, as follows:

- let $\mathsf{edg}(e) := \exists x\, \mathsf{inc}(x, e)$
- let $\mathsf{vtx}(x) := \nexists y\, \mathsf{inc}(y, x)$
- every occurrence of $e \in E$ is replaced by $\mathsf{edg}(e)$
- every occurrence of $x \in V$ is replaced by $\mathsf{vtx}(x)$
- every occurrence of $\mathsf{inc}(x, e)$ is replaced by $\mathsf{edge}(x, e)$

Now applying Corollary 3 gives the claimed algorithm. □

2 Encoding of a $(\omega+1)$-colored graph with a tree-decomposition as a Σ-tree, and involution

The exposition presented in this lecture note is inspired by [3] (link) We fix a nice tree-decomposition (T, β) of G of width at most ω and the MSO-sentence to verify on G. We want to build a suitable alphabet $\Sigma := \Sigma_\omega$, a Σ-tree (T, ρ) with $\rho : N(T) \to \Sigma$ and an MSO-sentence φ_{tree} on Σ-trees, i.e. over the vocabulary $\tau_\Sigma = \{\mathsf{child}_1, \mathsf{child}_2, \} \cup \{P_a \mid a \in \Sigma\}$ such that

$$G \models \varphi \text{ if and only if } (T, \rho) \models \varphi_{tree}.$$

Here the underlying tree of the Σ-tree (T, ρ) will be actually the same as the tree in the tree-decomposition (T, β).

We shall investigate

- how to define Σ, and
- how to determine $\rho(t)$ for each node t of T from the given tree-decomposition (T, β) and G
- how to construct φ_{tree}.

Before we delve into details, let us contemplate on what we need in order to obtain such a sentence φ_{tree} on Σ-trees. Recall that atomic formulas on graphs are of the form: $x \in X$, $\mathsf{edge}(x, y)$ and $x = y$. So the first idea is that if you find a way to talk about "a vertex in the graph G, especially if two vertices are the same", if "a vertex is contained in a vertex set" *on the Σ-tree using the vocabulary τ_Σ* and "adjacency of two vertices in G again *on the Σ-tree using the vocabulary τ_Σ*, then we are halfway done. To be slightly more

specific: first, we want to encode the tree-decomposition (G, β) and G with a suitable Σ-tree, and secondly, under this encoding scheme, we want to have MSO-formulas on Σ-trees which tests if two vertices of G are the same, if a vertex is contained in some set and if two vertices of G are adjacent. We develop the first idea on how to come up with such an encoding and MSO-formulas.

Step 1. Fix a $\omega + 1$-coloring of G, giving a label from $\{0\} \cup [\omega]$ to each vertex of G. We use the fact that G of treewidth at most ω can be colored using $\omega + 1$ colors so that no two vertices that are contained in some bag together gets the same color. We fix such a coloring $c : V(G) \to [\omega + 1]$ and say that vertex v of G has *color* i if $c(v) = i$.

Step 2. Rooted graph and gluing operation. A k-*rooted* graph is a graph G equipped with

- a distinguished vertex subset $R \subseteq V(G)$ called the *root of* H, and
- a partial injective function $\iota : V(G) \to [k]$ such that $\iota(v)$ is defined for each vertex v in the root.

If $\iota(v)$ is defined, we say that v has label $\iota(v)$. A graph is seen as a trivial rooted graph which has an empty root and no label is assigned to any vertex. We often omit to specify the boundary and the labeling function.

Let (G_1, R_1, ι_1) and (G_2, R_2, ι_2) be two k-rooted graphs. The *gluing* $G_1 \oplus_k G_2$ of G_1 and G_2 is the k-rooted graph (G, R, ι) defined[1] when there is a label-preserving isomorphism between $G_2[R_2]$ and some induced subgraph of H_1; otherwise, the gluing operation is not defined, denoted as $G_1 \oplus G_2 = \bot$. When it is defined, the new rooted graph is defined from the disjoint union of H_1 and H_2 by

- *identifying* each vertex $v \in R_2$ with a vertex v' of $V(H_1)$ satisfying $\iota_1(v') = \iota_2(v)$,
- declaring R_1 as the root R, and
- $\iota := \iota_1$.

Notice that ι in the above is injective because ι_1 was injective.

A tree-decomposition (T, β) of G of width ω with a coloring c can be understood as a way to obtain G (as a trivial rooted graph) by repeated applying gluing operations on $\omega + 1$-rooted graphs. With each tree node t of the tree-decomposition, we associate the $\omega + 1$-rooted graph (H_t, R_t, ι_t) consisting of

- $H_t := G[\beta(t)]$,
- $R_t := \mathsf{adh}(t)$, and
- ι_t is the restriction of the coloring c on $\beta(t)$.

which we call the *node graph at node* t.

From the node graphs over all tree nodes, we shall obtain the graph G as follows. A *cone graph* at node t is the $\omega + 1$-rooted graph (G_t, R_t, ι_t), where R_t and ι_t defined as above, such that $G_t := G[\mathsf{cone}(t)]$. Clearly, the graph G is the cone graph at the root of the tree-decomposition. Also note that at a leaf node t, its cone graph is identical to its node graph. For an internal node t with two children t_1 and t_2, we can obtain the cone graph at node t as follows. Let (G_i, R_i, ι_i) be the cone graph of the child t_i for $i = 1, 2$.

[1]The gluing operation here is slightly different from the gluing operation often presented in the literature.

Lemma 5. *It holds that* $(G_t, R_t, \iota_t) = \big((H_t, R_t, \iota_t) \oplus (G_1, R_1, \iota_1)\big) \oplus (G_2, R_2, \iota_2)$.

Step 3. Fixing the alphabet Σ and labeling the tree T. The basic idea for constructing a Σ-tree (T, ρ) from the coloring c and (T, β) is to view the *node graph* at each tree node as a letter of Σ. A minor issue here is that a node graph itself maintain the original name of a vertex, so even though each node graph has bounded size (on at most $\omega + 1$ vertices), the number of such graphs can be arbitrarily large. This issue is resolved by ignoring the names of a vertex, and associate each node graph to a vertex on a fixed vertex set as a subset of $\{0\} \cup [\omega]$.

Now, t is uniquely associated with the rooted graph $(\bar{H}_t, \bar{R}_t, \bar{\iota}_t)$, which is precisely the graph obtained from the node graph (H_t, R_t, ι_t) by renaming each v by its label $\iota_t(v)$ and taking $\bar{\iota}_t$ as the identity function. Because ι_t defines the value of each vertex in a node graph, indeed the graph obtained in this way has the vertex set contained in $\{0\} \cup [\omega]$. As the labeling function ι is always an identity function, we omit it from the description of the rooted graph. Let

$$\Sigma = \{(H, R) : (H, R) \text{ is a rooted graph on a subset of } \{0\} \cup [\omega] \text{ as the vertex set}\},$$

and we give the rooted graph (H_x, R_x) as the label of node x, creating a Σ-tree (T, ρ). The Σ-tree (T, ρ) is called the *encoding of the triple G, coloring c and (T, β)*.

You may choose to encode a rooted graph $(H, R) \in \Sigma$ as a symbol (e.g. a constant-length string). However, we can maintain a look-up table which lists up the association of the rooted graphs and the letters in Σ. Therefore, we shall refer to a symbol from Σ as a rooted graph.

Step 4. Decoding the Σ-tree (T, ρ) to retrieve G, (T, β) and the coloring c. A Σ-tree (T, ρ) is said to be *legal* if for every node x and its parent y in T,

- $R_x \subseteq V(H_y)$, and
- the subgraphs induced by R_x coincide in H_x and H_y, i.e. $H_x[R_x] = H_y[R_x]$.

The next statement is trivial.

Lemma 6. *There is an* MSO*-sentence φ_{legal} over τ_σ such that $(T, \rho) \models \varphi_{legal}$ if and only if (T, ρ) is legal.*

When (T, ρ) is a legal Σ-tree, the construction of Lemma 5 is well-defined and one can retrieve a graph, its tree-decomposition and a coloring which give rise to the very Σ-tree (T, ρ).

For a node t of a legal (T, ρ), let (H_t, R_t, ι_t) be the graph obtained from $(H, R) \in \rho(t)$ by

- renaming each vertex i of H as $v(t, i)$,
- taking $R_t := R$, and
- setting $\iota_t(v(t, i)) = i$.

This is a rooted graph, in particular ι_t is injective whose domain is $V(H_t)$.

Let us associate the following rooted graph $(\bar{G}_t, \bar{R}_t, \bar{\iota}_t)$ to each node t of T recursively from bottom to top.

- When t is a leaf, we set $(\bar{G}_t, \bar{R}_t) := (H_t, R_t)$ and $\bar{\iota}_t := \iota_t|_{R_t}$.

- When t is an internal node with the two children of t be t_1 and t_2, let $(\bar{G}_i, \bar{R}_i, \bar{\iota}_i)$ be the rooted graph constructed for the child t_i for $i = 1, 2$. Then we obtain $(\bar{G}_t, \bar{R}_t, \bar{\iota}_t)$ from

$$\big((H_t, R_t, \iota_t) \oplus (\bar{G}_1, \bar{R}_1, \bar{\iota}_1)\big) \oplus (\bar{G}_2, \bar{R}_2, \bar{\iota}_2)$$

  by forgetting the label of each vertex in $\bar{G}_t$ not in the root $\bar{R}_t$ to create $\bar{\iota}_t$.

Lemma 7. *Let (T, ρ) is a legal Σ-tree. Then the rooted graph $(\bar{G}_t, \bar{R}_t, \bar{\iota}_t)$ is well-defined for each tree node t of T. Moreover, the domain of $\bar{\iota}_t$ is $\bar{R}_t$.*

Proof: For the leaf node, the statement trivially holds by construction. Consider an internal node t and we may assume that for each child t_1 and t_2, the statement holds with the associated rooted graph $(\bar{G}_j, \bar{R}_j, \bar{\iota}_j)$ for $j = 1, 2$.

We want to see that $(\bar{G}_t, \bar{R}_t, \bar{\iota}_t)$ is well-defined with $\bar{R}_t$ as the domain of $\bar{\iota}_t$. First, note that

$$(\bar{G}'_1, \bar{R}'_1, \bar{\iota}'_1) := (H_t, R_t, \iota_t) \oplus (\bar{G}_1, \bar{R}_1, \bar{\iota}_1)$$

is well-defined, where ι_t is the identity function on $V(H_t)$. Indeed, due to the legality assumption on (T, ρ) there is a label-preserving isomorphism between $\bar{G}_1[\bar{R}_1]$ and $H_t[\bar{R}_1]$, so the gluing operation is well-defined. Secondly, observe that $(\bar{G}'_1, \bar{R}'_1, \bar{\iota}'_1) \oplus (\bar{G}_2, \bar{R}_2, \bar{\iota}_2)$ is well-defined because the gluing operation which yields $(\bar{G}'_1, \bar{R}'_1, \bar{\iota}'_1)$ keeps (H_t, R_t, ι_t) intact in $(\bar{G}'_1, \bar{R}'_1, \bar{\iota}'_1)$. Therefore, the second gluing operation $(\bar{G}'_1, \bar{R}'_1, \bar{\iota}'_1) \oplus (\bar{G}_2, \bar{R}_2, \bar{\iota}_2)$ is well-defined due to the legality assumption. As $(\bar{G}_t, \bar{R}_t, \bar{\iota}_t)$ is obtained from $(\bar{G}'_1, \bar{R}'_1, \bar{\iota}'_1) \oplus (\bar{G}_2, \bar{R}_2, \bar{\iota}_2)$ by taking $\bar{\iota}_t$ as the restriction its partial labeling function onto the new root, i.e. $\bar{R}_t$, it follows that the domain of $\bar{\iota}_t$ is $\bar{R}_t$.. □

We remark that Lemma 7 holds even when the Σ-tree is not restricted to binary trees. Given a legal Σ-tree (T, ρ), the *decoding* of (T, ρ) is the triple $(G, c, (T, \beta))$ consisting of the rooted graph G, a coloring $c : V(G) \to \{0\} \cup [\omega]$ and a binary tree-decomposition (T, β) of G of width at most ω such that

I. $G := \bar{G}_r$ where r is the root of (T, ρ),

II. c is a mapping from $V(G)$ to $\{0\} \cup [\omega]$ so that $c(v) = i$ if $v = v(t, i)$ for some node t of (T, ρ)

III. $v \in \beta(t)$ for some node t of T if $v = v(t, i)$ for some i.

Lemma 8. *If Σ-tree (T, ρ) is legal, its decoding $(G, c, (T, \beta))$ is well-defined.*

Proof: The rooted graph $(\bar{G}_r, \bar{R}_r, \bar{\iota}_r)$ where r is the root of (T, ρ), is well-defined by Lemma 7 and it can be seen as a normal graph because $\bar{R}_r = \emptyset$. Let $G := \bar{G}_r$. For (II), it suffices to recall that we identify two vertices in the gluing operation only when their labels coincide.

Consider the pair (T, β) as defined in (III) and we claim that (T, β) is a tree-decomposition. As any vertex v of G is $v = (t, i)$ for some t and i by construction of G, the vertex coverage condition is met. The edge coverage condition is immediate from that G is obtained from the disjoint union of (H_t, R_t, ι_t) over all nodes t of (T, ρ) and during the gluing operation we do not create a new edge. To see the connectivity condition, choose an arbitrary vertex v of G and let U_v be the set of nodes of (T, ρ) such that $v = v(t, i)$. If U_v is not connected, let t_1 and t_2 be two nodes of (T, ρ) from distinct connected components of U. We choose t_1 and t_2 so that there distance on T is as small as possible. Then at least one of t_1 and t_2 is the

topmost node of a connected component of U; without loss of generality, t_2 is the topmost node of U_2, where U_2 is a connected component of $T[U]$. In order for $v(t_2, i)$ to be identified with $v(t_1, i)$, it must be contained in $\bar{R}_{t_2}$. Let t_2' be the parent of t_2 in (T, ρ). Due to the legality of (T, ρ), this means that there is a vertex $v(t_2', i)$ and it is identified with $v(t_2, i)$ during the construction of G. It follows that t_2' belongs to U, contradicting the choice of t_1 and t_2. □

The next lemma is straightforward.

Lemma 9. *A Σ-tree (T, ρ) is legal if and only if it is the encoding of some $(G, c, (T, \beta))$, i.e. a $\omega+1$-colored graph with a binary tree-decomposition of width at most ω. Moreover, such $(G, c, (T, \beta))$ is the decoding of (T, ρ).*

Proof: The backward implication is trivial. To see the forward direction, let $(G, c, (T, \beta))$ be the decoding of (T, ρ), which is well-defined (and thus unique) due to Lemma 8. Observe that $\beta(t) = V(H_t)$ holds for every tree nodes t of T and thus has at most $\omega + 1$ vertices. It is tedious to verify that (T, ρ) is the encoding of $(G, c, (T, \beta))$. The second statement is clear from the construction of $(G, c, (T, \beta))$ as the decoding of (T, ρ). □

3 Combinatorial properties

We explore the combinatorial properties of a tree-decomposition of $\omega + 1$-colored graph, as a key consequence of the encoding/decoding presented in the previous section, which can be readily used for queries on graph definable in MSO-formulas on Σ-trees.

Lemma 10. *Let (T, β) be a binary tree-decomposition of G of width at most ω and c is a $\omega + 1$-coloring on $V(G)$ so that no two vertices in the same bag have the same color. Then (U, i), where $U \subseteq V(G)$ and $i \in \{0\} \cup [\omega]$ is the defining pair of some vertex v of G if and only if it satisfies the following condition.*

(i) The node set U is connected in T.

(ii) For each node $t \in U$, there is some vertex of color i in $\beta(t)$

(iii) For every node $t \in U \setminus \mathsf{top}(U)$, there is a vertex of color i in $\mathsf{adh}(t)$

(iv) For $t := \mathsf{top}(U)$, v does not belong to $\mathsf{adh}(t)$ (i.e. $v \in \beta(t) \setminus \mathsf{adh}(t)$).

(v) For every node $t \notin U$ whose parent is in U, no vertex in $\mathsf{adh}(t)$ has color i.

Proof: If (U, i) is the defining pair of $v \in V(G)$, that is, $U = \textsc{bags}(v)$ and $i = c(v)$ then it is straightforward to see that (i)-(v) hold: (i) holds by the connectivity condition of a tree-decomposition and that v has color i; (ii) holds by the definition of a defining pair; (iii) holds because for every node $t \in \textsc{bags}(v)$ whose parent is also in $\textsc{bags}(v)$, we trivially have $v \in \beta(t) \cap \beta(\mathsf{parent}(t)) = \mathsf{adh}(t)$. To see (iv), if v is in the adhesion of $\mathsf{top}(\textsc{bags}(v))$, then this means that v is in the bag of the parent of $\mathsf{top}(\textsc{bags}(v))$, contradicting the construction of $\textsc{bags}(v)$. For (v), observe that for any such t, if there is a vertex $u \in \mathsf{adh}(t)$ of color i then $u \neq v$ because otherwise t should be also included in $\textsc{bags}(v)$. Now $c(u) = c(v)$ while both u and v belong to the bag of $\mathsf{parent}(t)$, contradicting the construction of coloring c.

To see the opposite direction, suppose that (U, i) meets (i)-(v). Let (T, ρ) be the encoding of (T, β) and observe that the vertices $v(t, i)$ from (T, ρ) are identified as a single vertex in the decoding of (T, ρ). Moreover, the conditions (iv)-(v) ensures that in the decoding $(\bar{G}, \bar{c}, (T, \bar{\beta})$ of (T, ρ), U is precisely the set of tree nodes whose bags contain v. Lastly, it remains to observe that $(G, c, (T, \beta))$ is precisely $(\bar{G}, \bar{c}, (T, \bar{\beta})$ thanks to Lemma 9. □

Thanks to Lemma 9 and Lemma 6, once a Σ-tree (T, ρ) satisfies the MSO-sentence φ_{legal}, we can talk about a vertex of G, its color and a bag of the tree-decomposition. Next, we shall see that one can construct MSO-formulas over τ_Σ that allow us to refer to a specific vertex, vertex sets as well as adjacency between vertices of G.

For a vertex v of G, let $\text{BAGS}(v) \subseteq N(T)$ be the set of tree nodes whose bag contains v. We associate each vertex v of G with the pair $(\text{BAGS}(v), i_v)$, called a *defining pair of* v, where i_v is the color of v. The next lemmas are straightforward and we omit their proofs.

Lemma 11. *If $u \neq v$, their defining pairs are distinct. Furthermore, Moreover, if $i_u = i_v$, then $\text{BAGS}(u) \cap \text{BAGS}(v) = \emptyset$ (and thus u and v are non-adjacent).*

Lemma 12. *uv is an edge of G if and only if there exists a tree node $t \in \text{BAGS}(u) \cap \text{BAGS}(v)$ such that the vertices $i_u, i_v \in \{0\} \cup [\omega]$ are adjacent in H_t*

We wish to query whether $(U, i) \in 2^{N(T)} \times (\{0\} \cup [\omega])$ is a defining pair or not using an MSO-formula over τ_Σ, but i is not a part of the universe of a Σ-tree. Instead, we shall use $\vec{U} := (U_0, \ldots, U_\omega) \in (2^{N(T)})^{\omega+1}$.

- a vertex $v \in V(G)$ of color i will be associated with $\vec{U} := (U_0, \ldots, U_\omega)$ such that

$$U_j = \begin{cases} \text{BAGS}(v) & \text{if } j = i \\ \emptyset & \text{otherwise.} \end{cases}$$

  We call such $(\omega + 1)$-tuple the *defining tuple of* v, and write as $\vec{U}^v$.

- a vertex subset $S \in V(G)$ will be associated with $\vec{U} := (U_0, \ldots, U_\omega)$ such that

$$U_i = \bigcup_{v \in S, c(v) = i} \text{BAGS}(v)$$

  We call such $(\omega + 1)$-tuple the *defining tuple of* S, and write as $\vec{U}^S$.

Note that the defining tuple of a vertex set S can be obtained as a coordinate-wise disjoint union of the defining tuples of all vertices in S. The disjointness of a union follows from Lemma 11.

Lemma 13. *Let (T, β) be a binary tree-decomposition of G of width at most ω and c is a $\omega + 1$-coloring on $V(G)$ so that no two vertices in the same bag have the same color. Then $(U_0, \ldots, U_\omega)$ the defining tuple of some vertex set S of G if and only if each U_i can be partitioned into sets $(U_i^j)_{j=1,\ldots}$ so that the following properties hold for each $U := U_i^j$.*

(i) The node set U is connected in T.

(ii) For each node $t \in U$, there is some vertex of color i in $\beta(t)$

(iii) For every node $t \in U \setminus \mathsf{top}(U)$, there is a vertex of color i in $\mathsf{adh}(t)$

(iv) For $t := \mathsf{top}(U)$, v does not belong to $\mathsf{adh}(t)$ (i.e. $v \in \beta(t) \setminus \mathsf{adh}(t)$).

(v) For every node $t \notin U$ whose parent is in U, no vertex in $\mathsf{adh}(t)$ has color i.

Proof: The forward implication is a corollary of Lemma 10 and Lemma 11. To see the opposite direction, suppose that $(U_0, \ldots, U_\omega)$ admits a partition for each U_i such that each part satisfies the conditions (i)-(v). By Lemma 10, each part of U_i is the defining pair (when the tuple $(\emptyset, \emptyset, \ldots, U_i, \ldots)$ obtained by replacing the i-th entry of all-$\emptyset$ tuple by U_i is construed as the pair (U, i)) of a vertex of color i. Let S be the set of all vertices which give rise to some defining pair in this way. As $(U_0, \ldots, U_\omega)$ is a coordinate-wise (disjoint) union of defining pairs over all vertices in S, it is the defining tuple of S. □

4 Converting MSO-sentence on graphs into an MSO-formula over Σ-trees

Step 4. Building MSO-formulas over τ_Σ for basic queries on graphs. Using the defining tuples, we can simulate the atomic formulas over graphs using formulas over τ_Sigma as follows.

Lemma 14. *There is an* MSO*-formula $\varphi_{\mathsf{vtx},i}(Z)$ over τ_Σ with a (single) free set variable such that $(T, \rho) \models \varphi_{\mathsf{vtx}}(U)$ if and only if (T, ρ) is legal and (U, i) is the defining pair of some vertex v (of color i) in the decoding of (T, ρ).*

Lemma 15. *There is an* MSO*-formula $\varphi_{\mathsf{vtx}}(\vec{Z})$ over τ_Σ with $\omega + 1$ free set variables such that $(T, \rho) \models \varphi_{\mathsf{vtx}}(\vec{U})$ if and only if (T, ρ) is legal and $\vec{U}$ is the defining tuple of some vertex v in the decoding of (T, ρ).*

Lemma 16. *There is an* MSO*-formula $\varphi_{\mathsf{set}}(\vec{Z})$ over τ_Σ with $\omega + 1$ free set variables such that $(T, \rho) \models \varphi_{\mathsf{set}}(\vec{U})$ if and only if (T, ρ) is legal and $\vec{U}$ is the defining tuple of some vertex subset S in the decoding of (T, ρ).*

Step 5. Converting an MSO-sentence about graphs into an MSO-sentence about Σ-trees.

Once such formulas are available, one can convert the given MSO-sentence φ over the graph vocabulary into a new MSO-sentence over tree vocabulary τ_Σ. We do not consider a universal quantifier separately as it holds that $\forall x\ \phi(x) = \nexists x\ \neg\phi(x)$.

The atomic formulas over graphs can be easily simulated by formulas over τ_σ.

Lemma 17. *There is an* MSO*-formula $\varphi_{\mathsf{edge}}(\vec{Z_1}, \vec{Z_2})$ over τ_Σ with $2(\omega + 1)$ free set variables such that $(T, \rho) \models \varphi_{\mathsf{edge}}(\vec{U_1}, \vec{U_2})$ if and only if (T, ρ) is legal and $\vec{U_i}$ is the defining tuple of some vertex v_i for $i = 1, 2$ and $v_1 v_2$ is an edge in the decoding of (T, ρ).*

Lemma 18. *There is an* MSO*-formula $\varphi_{equal}(\vec{Z_1}, \vec{Z_2})$ over τ_Σ with $2(\omega + 1)$ free set variables such that $(T, \rho) \models \varphi_{equal}(\vec{U_1}, \vec{U_2})$ if and only if (T, ρ) is legal and $\vec{U_i}$ is the defining tuple of some vertex v_i for*

$i = 1, 2$ *and* $v_1 = v_2$ *in the decoding of* (T, ρ).

Lemma 19. *There is an* MSO*-formula* $\varphi_{\mathsf{contain}}(\vec{Z}_1, \vec{Z}_2)$ *over* τ_Σ *with* $2(\omega + 1)$ *free set variables such that* $(T, \rho) \models \varphi_{\mathsf{contain}}(\vec{U}_1, \vec{U}_2)$ *if and only if* (T, ρ) *is legal and* $\vec{U}_1$ *is the defining tuple of some vertex* v*,* $\vec{U}_2$ *is the defining tuple of some vertex subset* X *and* $v \in S$ *in the decoding of* (T, ρ).

Using the MSO-formulas presented in Lemma 15, Lemma 16, Lemma 17, Lemma 18 and Lemma 19 we are now ready present the construction of φ_{tree} from φ.

- Each introduction of an individual variable x in φ, e.g. $\exists x\ \phi(x)$ or $\forall x\ \phi(x)$ is replaced by the $(\omega+1)$-tuple $\vec{Z^x} := (Z_0^x, \ldots, Z_\omega^x)$ of set variables followed by $\varphi_{\mathsf{vtx}}(\vec{Z^x})$. That is,

$$\exists x\ \phi(x) \qquad \Rightarrow \qquad \exists Z_0^x\ \exists Z_1^x \cdots\ \exists Z_\omega^x\ \varphi_{\mathsf{vtx}}(\vec{Z^x}) \wedge \phi'(\vec{Z^x}).$$

  The construction of ϕ' is explained later.

- Each introduction of a set variable X in φ, e.g. $\exists X\ \phi(X)$ or $\forall X\ \phi(X)$ is replaced by the $(\omega+1)$-tuple $\vec{Z^X} := (Z_0^X, \ldots, Z_\omega^X)$ of set variables followed by $\varphi_{\mathsf{set}}(\vec{Z^X})$. That is,

$$\exists X\ \phi(X) \qquad \Rightarrow \qquad \exists Z_0^X\ \exists Z_1^X \cdots\ \exists Z_\omega^X\ \varphi_{\mathsf{set}}(\vec{Z^X}) \wedge \phi'(\vec{Z^X}).$$

  The construction of ϕ' is explained later.

- Instead of each atomic formula $x = y$ in φ, we have

$$\varphi_{equal}(\vec{Z^x}, \vec{Z^y}),$$

  where $\vec{Z^x}$ (resp. $\vec{Z^y}$) is the $(\omega + 1)$-tuple of free set variables introduced for x (resp. y).

- Instead of each atomic formula $\mathsf{edge}(x, y)$ in φ, we have

$$\varphi_{\mathsf{edge}}(\vec{Z^x}, \vec{Z^y}),$$

  where $\vec{Z^x}$ (resp. $\vec{Z^y}$) is the $(\omega + 1)$-tuple of free set variables introduced for x (resp. y).

- Instead of each atomic formula $x \in X$ in φ, we have

$$\varphi_{\mathsf{contain}}(\vec{Z^x}, \vec{Z^X}),$$

  where $\vec{Z^x}$ (resp. $\vec{Z^X}$) is the $(\omega + 1)$-tuple of free set variables introduced for x (resp. X).

5 Completing the Proof

We use the following additional formula for defining queries on Σ-trees, which can be easily designed and left for the readers as exercise.

- $\mathsf{parent}(y, x)$; "y is the parent of x in T."
- $\mathsf{root}(x)$; "x is the root of T, i.e. it does not have a parent."

- $\mathsf{top}(x, X)$; "X is connected and x is the topmost node of X"
- $\mathsf{connected}(X)$; "X is connected in T."
- $\mathsf{dangle}(x, X)$; "$x \notin X$ and the parent of x is in X."

Proof of Lemma 6: The MSO-sentence φ_{legal} verifies whether for every node x of T with the parent y, the symbols (H_x, R_x) and (H_y, R_y) on the nodes x and y satisfy the two conditions for legality. Specifically, for every $a \in \Sigma$, let Σ_a be the subset of Σ such that for each $(H_t, R_t) \in \Sigma_a$, $R_a \subseteq V(H_t)$ and $H_a[R_a] = H_t[R_a]$. Intuitively, Σ_a is the subset of Σ which is *compatible* with a child having a as its label in ρ. Then we can construct an MSO-formula

$$\varphi_{local}(x, y) := \mathsf{parent}(y, x) \wedge \bigvee_{a \in \Sigma} \big(P_a(x) \wedge \bigvee_{b \in \Sigma_a} P_b(y)\big).$$

Clearly, $\varphi_{local}(x, y)$ holds on a Σ-tree (T, ρ) with an ordered pair $(\bar{x}, \bar{y}) \in N(T) \times N(T)$ if and only if $\bar{y}$ is the parent of $\bar{x}$ and its label in ρ is compatible with that of $\bar{x}$.

Finally we get

$$\varphi_{legal} := \forall x\ \mathsf{root}(x) \vee \exists y\ \varphi_{local}(x, y).$$

□

Note that the condition (ii) in Lemma 10 do not say that there is a vertex which appear in each bag. This property is rather an outcome of (ii) combined with (iii).

Proof of Lemma 14: For checking the legality, we use φ_{legal} in Lemma 6. Then the formula $\varphi_{\mathsf{vtx},i}(\vec{Z})$ verifies if $U = \text{BAGS}(v)$ for some vertex v color i in the decoding of (T, ρ). This is done by line-by-line translation of the conditions of Lemma 10 into MSO-formulas, which we recall below.

(i) U_i is connected;

(ii) for every node $x \in U_i$, the vertex set of the graph (H_x, R_x) contains $i \in \{0\} \cup [\omega]$,

(iii) for every node $x \in U_i \setminus \mathsf{top}(U_i)$, the vertex i belongs to R_x,

(iv) at the node $\mathsf{top}(U_i)$, i is *not in* $R_{\mathsf{top}(U_i)}$,

(v) for every node $x \notin U_i$ whose parent in U_i, i is *not in* R_x.

Each query can be implemented with MSO-formulas easily ; (i) is simply implemented by the formula $\mathsf{connected}(X)$, (ii) can be done by checking for each $x \in U$ if $P_a(x)$ holds over all all $a \in \Sigma$ which corresponds to a rooted graph (H_x, R_x) having i in R_x. (iii)-(iv) can be handled using the formula top and dangle.

Putting everything together,

$$\begin{aligned}\varphi_{\mathsf{vtx}}(\vec{Z}) := \varphi_{legal} \wedge & \\ & \mathsf{connected}(Z) \wedge \\ & \forall y \, \big(y \in Z \rightarrow \bigvee_{a \in \Sigma_i} P_a(y)\big) \wedge \\ & \forall z \, \big(z \in Z \rightarrow \bigvee_{a \in \Sigma_{i,root}} P_a(z) \vee \mathsf{top}(z, Z)\big) \wedge \\ & \forall t \, \big(\mathsf{top}(t, Z) \rightarrow \neg \bigvee_{a \in \Sigma_{i,root}} P_a(t)\big) \wedge \\ & \forall w \, \big(\mathsf{dangle}(w, Z) \rightarrow \neg \bigvee_{a \in \Sigma_{i,root}} P_a(w)\big)\end{aligned}$$

Here $\Sigma_i \subseteq \Sigma$ is the set of all (H_x, R_x) in which $i \in V(H_x)$ and $\Sigma_{i,root} \subseteq \Sigma$ is the set of all (H_x, R_x) in which $i \in R_x$.

It is straightforward to verify that $(T, \rho) \models \varphi_{\mathsf{vtx}}(U)$ if and only if (T, ρ) is legal and U satisfies the conditions (i)-(iv).

It remains to show that U satisfies (i)-(iv) in (T, ρ) if and only if it is the defining tuple of some vertex v of color i in the decoding of (T, ρ).

The backward direction is straightforward by Lemma 10. □

Proof of Lemma 15: It is straightforward to check that the following formula does the job.

$$\varphi_{\mathsf{vtx}}(\vec{Z}) := \varphi_{legal} \wedge \bigwedge_{0 \leq i \neq j \leq \omega} \big(\varphi_{\mathsf{vtx},i}(Z_i) \rightarrow (Z_j = \emptyset)\big) \wedge \exists z \bigvee_{0 \leq i \leq \omega} (z \in Z_i).$$

□

Proof of Lemma 16: Again we may assume that (T, ρ) is legal. Using Lemma 13, we express the query on a tuple $\vec{U} = (U_0, \ldots U_\omega)$ which tests the *decomposability* of each U_i. That is, for every node $t \in U_i$, the formula tests whether there exists a set $U \subseteq U_i$ containing t which satisfies the conditions of (i)-(v) in Lemma 13. As (i)-(v) in Lemma 13 is identical as the conditions (i)-(v) in Lemma 10, one can use the MSO-formula $\varphi_{\mathsf{vtx}}(\vec{Z})$ as

$$\forall t \big(t \in U_i \rightarrow \exists Z \, (Z \subseteq U_i) \wedge (t \in Z) \wedge \varphi_{\mathsf{vtx},i}(Z)\big)$$

□

The proofs of Lemma 17, Lemma 18 and Lemma 19 are left as exercises for the readers.

Proof of Theorem 1: We first fix a $(\omega + 1)$-coloring of G such that no two vertices of the same color is in the same bag of (t, β). Such a coloring c exists (Lemma from the previous lecture note) and can be determined greedily in linear time. Given the coloring c, the Σ-tree (T, ρ) can be constructed linear time from the given tree-decomposition (T, β).

From the input MSO-formula φ over $\{\mathsf{edge}\}$, one can construct an MSO-formula φ'_{tree} over τ_Σ in constant number of steps as presented in the previous section, where the constant depends on the length of φ and Σ

(thus on ω). Let

$$\varphi_{tree} := \varphi_{legal} \wedge \varphi'_{tree}$$

and note that $(T, \rho) \models \varphi_{tree}$ if and only if (T, ρ) is legal and satisfies φ'_{tree}, which holds if and only if $G \models \varphi$ by Lemma 15, Lemma 16, Lemma 17, Lemma 18 and Lemma 19. Therefore, it holds that

$$(\star)\ G \models \varphi \text{ if and only if } (T, \rho) \models \varphi_{tree}.$$

Due to the equivalence of MSO-definability and regularity of tree language (previous lecture note), there exists a deterministic finite tree automaton M which recognizes precisely the tree language consisting of Σ-tree which satisfies φ_{tree}. Therefore, by running this automaton on (T, ρ) (in linear time), one can determine if $(T, \rho) \models \varphi_{tree}$ or not. If M (does not) accepts (T, ρ), then we conclude $G \models \varphi$ ($G \not\models \varphi$). The correctness of the algorithm is immediate from $(\star)$. □

References

[1] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.

[2] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192, 2022.

[3] Mikołaj Bojańczyk. Interactive lecture note: Interpreting a graph in its tree decomposition. 2015.