Expressive Power of Graph Languages

Timos Antonopoulos

University of Cambridge
Computer Laboratory
St. Catharine's College

This dissertation is submitted for
the degree of Doctor of Philosophy

Abstract

Finite model theory is a field of logic that originated from computer science and studies properties of finite logical structures. Algorithmic properties of problems are interpreted over structures and inspected using logical languages, and thus connections are made between the latter and computational models. The expressive power of such logical languages is examined and matched with classes of problems and algorithmic classifications.

In this thesis, we look into the expressive power of Graph Logic and also its extension with a recursion operator, that was introduced by Cardelli, Gardner and Ghelli, and later systematically investigated by Dawar, Gardner and Ghelli. Graph Logic was introduced as a query language on labelled directed graphs and is an extension of first-order logic with a spatial connective that allows one to express that a graph can be decomposed into two subgraphs, and reason thereafter about the properties definable over the two subgraphs in isolation from each other. It was known that Graph Logic is contained in monadic second-order logic over graphs and we show that the containment is strict. Furthermore, we investigate the expressive power of the logic over restricted classes of graphs.

In addition, the expressive power of Graph Logic with the recursion operator, which over many classes of graphs exceeds the one of monadic second-order logic, is studied and compared to the expressive power of other related languages.

Preface

First of all, I would like to thank my supervisor Dr. Anuj Dawar, without whose continued support and guidance none of this work would have been possible. I am also very grateful for his encouragement and patience throughout my studies.

I am especially thankful to Dr. Jerzy Marcinkowski for a series of valuable discussions and sharing his insight with me. I would also like to thank Prof. Wolfgang Thomas and Prof. Martin Lange for kindly discussing specific aspects relating to my work. Furthermore, I would like to acknowledge Dr. Philippa Gardner for her advice prior to my taking up my PhD studies.

Finally, I would like to dedicate my thesis to my father, mother and sister, without whom I could have never accomplished any of this work.

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. Parts of Chapter 2 and Chapter 3 are based on [AD09]. This dissertation does not exceed 60,000 words, including footnotes.

Contents

Chapter 1

Introduction

A connection between Mathematical Logic and Automata Theory was first established by Büchi and Elgot's result ([Büc60],[Elg61]) stating that the Monadic fragment of Second Order Logic (the one where second order variables range over subsets of the universe of the structure) can define exactly the same languages over finite words as the ones recognized by finite word automata. This result was among the first that established a connection between a logic and a class of problems decidable by some theoretical machine model, such as automata or Turing machines, which is one of the fundamental issues of Finite Model Theory, along with the search for the exact expressive power of logical languages. Other similar connections made between a machine model and a logic include Fagin's famous result showing that $\exists$SO logic, the existential fragment of Second-Order Logic, describes exactly the problems in NP ([Fag74]), as well as Immerman and Vardi's result on LFP and P over ordered structures ([Imm86],[Var82]).

It has been observed that Monadic Second-Order Logic (MSO), over some classes of graphs which include words and trees, has linear data complexity. In other words, evaluating a fixed MSO formula on some arbitrary structure contained in one of these classes, takes linear time with respect to the size of the structure. This is possible due to a translation of MSO formulae to finite automata. Due to this connection between MSO and finite automata, MSO has been the standard to which many other logics are usually compared, especially when it comes to structures such as words and trees.

Graph Logic (GL), introduced by Cardelli et al. in [CGG02], is a spatial logic for querying graphs, with its characteristic operator being a spatial connective that allows for decomposing a graph into two edge-disjoint components, akin to how Separation Logic ([Rey02],[IO01]) is designed to work on memory heaps and other structures used for reasoning in programs with direct memory management through pointer variables. In [CG00], Cardelli et al. introduced Ambient Logic that allow for reasoning on the way the properties of processes change through time as well as spatially. In other words, through the logic, one can express what are the allowable changes on the spatial properties of a process, which are represented as a graph algebra, and for this, Ambient Logic makes use of a composition operator, similar to the one in GL. On the other hand, logics in finite model theory are traditionally used to reason on what is expressible over

certain classes of structures. The version of GL introduced in [CGG02] allows for expressing properties about graphs, and in addition includes operators allowing for querying graphs, and as a result creating new graphs from input graphs. Furthermore, one can also define transducers through the logic, which are mappings from graphs to graphs.

A fragment of GL introduced in [CGG02], which includes only the connectives used for expressing properties of graphs, was further studied in [DGG07] in order to resolve questions regarding the expressiveness of the logic as well as the complexity of the graphs definable in it and the complexity of evaluating the formulae. Two versions where investigated, one equipped with a recursion operator similar to the one in [CGG02], and the other version being without it. The two versions were termed GL_μ and GL respectively.

The characteristic spatial connective in the version of Graph Logic considered in [DGG07], is a composition operator that allows for reasoning about disjoint subgraphs in a graph. It is essentially a restricted form of second-order quantification overs edges, where in particular, one can express that a graph can be decomposed into two edge disjoint subgraphs, and then reason in the same way about definable properties of the subgraphs in isolation from each other. This spatial connective can be simulated in monadic second-order logic with quantification over edges, as shown in [DGG07], and hence all properties definable in Graph Logic are definable in MSO as well. The opposite direction of the containment is the one of interest.

Dawar et al. observed in [DGG07] that GL is of equal expressive power to MSO when restricted to the class of words, and hence captures the regular word languages. Despite this result, it was conjectured that it is strictly less expressive than MSO over graphs in general. In [Mar06], Marcinkowski showed that this is indeed the case for the GL version studied in [DGG07] and a corresponding extension of MSO that allows for quantification over a countable set of labels which label the edges of the structures. The two logics were termed GL+ and MSO+ in [Mar06], to avoid confusion with the version of GL and MSO that lacks this extra quantification over edge labels. The question whether GL is strictly less expressive than MSO was thus left open.

Graph Logic with an additional operator for recursion (GL_μ) was also introduced in [CGG02] and the expressive power of a version of it was studied further in [DGG07]. The latter operator increases the expressive power of GL, and in some classes of graphs such as trees and words, it is shown that it becomes more expressive than MSO. On the other hand, over graphs in general it is believed that there are MSO definable properties, such as 3-colourability, that are not definable in GL_μ. Despite the fact that the expressive power is increased with the use of the recursion operator, and the fact that PSPACE-complete problems are expressible in GL_μ, model checking as well as evaluating a fixed formula on an arbitrary graph, both remain in PSPACE. This is interesting since not many logics have data complexity and model-checking complexity that are both complete for the same complexity class. Least Fixed-Point Logic for example can only define problems in P but its model-checking complexity is EXPTIME-complete and it should be noted that model-checking of Graph Logic without the recursion operator is also PSPACE-complete. In this thesis, we study the expressive power of GL with and without the recursion

operator.

1.1 Preliminaries

We proceed by giving a few definitions for notions that will be used in this thesis. For an introduction to most of these notions we refer the reader to [EF99] and [Lib04].

Structures: A signature σ is defined as a tuple of the form $\langle c_1 \ldots, c_m, R_1^{r_1}, \ldots, R_n^{r_n} \rangle$, where the c_i are symbols of constants and the $R_j^{r_j}$ are relation symbols, of arity r_j. The signatures we consider are purely relational and do not have function symbols. Unless otherwise stated, they do not contain constants either.

A finite structure $\mathfrak{A}$ of signature σ is a tuple $\langle A, c_1 \ldots, c_m, R_1^{r_1}, \ldots, R_n^{r_n} \rangle$, where A is a finite set of elements and denotes the universe of the structure and $c_1, \ldots, c_m \in A$ are the constants, namely fixed elements of the structure and each R_i, for $1 \leq i \leq n$ is a relation of arity r_i, or in other words a subset of A^{r_i}. The interpretation of the symbols of a signature $\sigma = \langle c_1, \ldots, c_m, R_1, \ldots, R_n \rangle$ in a structure $\mathfrak{A}$ will be denoted by $c_1^{\mathfrak{A}}, \ldots, c_m^{\mathfrak{A}}, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}}$ when it is not clear, and similarly the universe of $\mathfrak{A}$ will be denoted by $A^{\mathfrak{A}}$. Finally, the class of all finite structures of signature σ is denoted as STRUC$[\sigma]$.

If $\mathfrak{A}$ and $\mathfrak{B}$ are two structures of the same signature without constants, then $\mathfrak{A} \uplus \mathfrak{B}$ denotes the structure that is the disjoint union of $\mathfrak{A}$ and $\mathfrak{B}$. In other words, if $\mathfrak{A} = \langle A, R_1^{r_1}, \ldots, R_n^{r_n} \rangle$ and $\mathfrak{B} = \langle B, R_1^{r_1}, \ldots, R_n^{r_n} \rangle$, then $\mathfrak{A} \uplus \mathfrak{B}$ is the structure with universe $A \uplus B$, and for all $1 \leq i \leq n$, $R_i^{\mathfrak{A} \uplus \mathfrak{B}} = R_i^{\mathfrak{A}} \uplus R_i^{\mathfrak{B}}$. Similarly if $k \in \mathbb{N}$, then $k \times \mathfrak{A}$ is the structure that is a disjoint union of k copies of $\mathfrak{A}$. Furthermore, if $\mathfrak{A}$ and $\mathfrak{B}$ are two structures with a tuple of elements $\bar{a}$ common to both, then $\mathfrak{A} \oplus_{\bar{a}} \mathfrak{B}$, denotes the disjoint union of the two structures $\mathfrak{A}$ and $\mathfrak{B}$, while identifying $\bar{a}$. For a set of structures $\mathfrak{A}_i$, for $n \leq i \leq n'$, we denote the disjoint union of these structures using the notation $\bigoplus_{i=n}^{n'} \mathfrak{A}_i$.

Queries: Let $\mathcal{C}$ be a class of structures. Then, an r-ary query Q on $\mathcal{C}$ is a mapping $Q : \mathcal{C} \to \{S \mid S \subseteq (A^{\mathfrak{A}})^r \text{ for } \mathfrak{A} \in \mathcal{C}\}$ that maps each structure $\mathfrak{A} \in \mathcal{C}$ to an r-ary relation on its universe $A^{\mathfrak{A}}$, satisfying the condition that if $f : \mathfrak{A} \to \mathfrak{B}$ is an isomorphism, then $Q(\mathfrak{B}) = f(Q(\mathfrak{A}))$. When a query Q is 0-ary, it is called a boolean query.

Notation: If $f : X \to Y$ is a function, mapping elements of a set X to elements of a set Y, then, for $x_1 \in X, y_1 \in Y$, $f[x_1 \mapsto y_1]$ is a function that maps x_1 to y_1, and for all x in the domain of f such that $x \neq x_1$, the function $f[x_1 \mapsto y_1]$ maps x to $f(x)$.

The lowercase letters $x, y, z, \ldots$ will usually denote variables and $\bar{x}$ denotes a tuple $(x_1, \ldots, x_n)$ for some $n \in \mathbb{N}$, in which case we also write $|\bar{x}| = n$. Similarly, the uppercase letters $X, Y, Z, \ldots$ are used for second-order variables, with $\bar{X}$ denoting a tuple of such variables.

Monadic Second Order Logic and First Order Logic will be denoted MSO and FO respectively. Similarly, we use GL for Graph Logic and GL$_\mu$ for Graph Logic with recursion. When it

is not clear, the satisfaction symbol '$\models$' will be subscripted with the appropriate logic.

Model Checking: We refer the reader to [Pap94] for a formal introduction to complexity classes. Considering the complexity of evaluating a formula φ of some logic $\mathcal{L}$ on some structure $\mathfrak{A}$, depends on which of the two is given as input to the problem and what is fixed. Vardi defined the following three classifications for these problems in [Var82]. First, we assume that a natural encoding exists for both formulae and structures, denoted by enc(φ) and enc($\mathfrak{A}$) respectively. Such an encoding and the definition of the different classifications that are presented below can be found in [Lib04] and [GKL$^+$07].

Definition 1.1.1. *Let $\mathcal{C}$ be some complexity class and $\mathcal{L}$ a logic. then*

- *The* data complexity *of $\mathcal{L}$ is $\mathcal{C}$ if for every sentence φ of $\mathcal{L}$, the language $\{enc(\mathfrak{A}) \mid \mathfrak{A} \models \varphi\}$ is in $\mathcal{C}$.*
- *The* expression complexity *of $\mathcal{L}$ is $\mathcal{C}$ if for every finite structure $\mathfrak{A}$, the language $\{enc(\varphi) \mid \mathfrak{A} \models \varphi\}$ is in $\mathcal{C}$.*
- *The* combined complexity *of $\mathcal{L}$ is $\mathcal{C}$ if the language $\{(enc(\mathfrak{A}), enc(\varphi)) \mid \mathfrak{A} \models \varphi\}$ is in $\mathcal{C}$.*

If for some sentence φ of $\mathcal{L}$, the language $\{\text{enc}(\mathfrak{A}) \mid \mathfrak{A} \models \varphi\}$ is $\mathcal{C}$-hard, and also $\mathcal{C}$, then the data complexity of $\mathcal{L}$ is $\mathcal{C}$-complete. Similarly for the expression and combined complexities. Throughout this thesis, model checking of some logic $\mathcal{L}$ refers to the combined complexity of $\mathcal{L}$.

1.1.1 Graphs

A graph $G = (V, E)$, is a set of vertices or nodes V, and a set of edges E that is usually considered as a binary relation defined as a subset of $V \times V$. The binary relation denotes edges connecting the vertices in the graph. Thus we consider only simple graphs, which are graphs that no two vertices have more than one edge connecting them, and furthermore we restrict to graphs such that $\forall v \in V$, $E(v, v)$ does not hold, or in other words E is irreflexive. Graphs are in general classified as directed and undirected. In the former case, if for two vertices $v_1, v_2 \in V$, $E(v_1, v_2)$ holds then $E(v_2, v_1)$ does not necessarily hold, to the contrary of undirected graphs, where the relation E is symmetric. For any graph $G = (V, E)$, we define $V(G)$ to be its set of vertices V and $E(G)$ to be its set of edges E.

As structures therefore, graphs are represented in mainly two ways in the literature ([Lib04], [GKL$^+$07], [Cou08]). A graph can be represented either using a universe for the vertices and a binary relation on vertices for the edges, or having a universe for both the vertices and the edges, and a binary relation, to describe which edges are adjacent to which vertices, where the binary relation is defined differently for directed and undirected graphs. Although a single binary relation can be used for directed graphs as well, we stick to a formulation similar to the one presented in [Cou08], and is as shown below.

More formally, let $G = (V, E)$ be a directed graph. Then let $\lfloor G \rfloor$ be the structure with universe $V_{\lfloor G \rfloor} = V$, and a binary relation $E_{\lfloor G \rfloor}$, such that for any two vertices $v_1, v_2 \in V_{\lfloor G \rfloor}$, $E_{\lfloor G \rfloor}(v_1, v_2)$ if and only if there is an edge in G, from v_1 to v_2. Similarly, let $\lceil G \rceil$ be the structure whose universe contains an element for each vertex and each edge of G. Let $V_{\lceil G \rceil}$ and $E_{\lceil G \rceil}$ denote these sets respectively, and let the signature of $\lceil G \rceil$ comprise two binary relations $\mathrm{inc}_1(e, x)$ and $\mathrm{inc}_2(e', x')$, such that for any two vertices $v_1, v_2 \in V_{\lceil G \rceil}$, $(v_1, v_2) \in E$ if and only if for the element $e \in E_{\lceil G \rceil}$, representing the edge (v_1, v_2), $\mathrm{inc}_1(e, v_1)$ and $\mathrm{inc}_2(e, v_2)$. Furthermore, for any $e \in E_{\lceil G \rceil}$,$v_1 \in V_{\lceil G \rceil}$, if $\mathrm{inc}_1(e, v_1)$, there exists a unique v_2 different from v_1 such that $\mathrm{inc}_2(e, v_2)$, and vice versa. The signature of the structures in the first case is denoted by $\sigma_{\lfloor G \rfloor}$ and by $\tau_{\lceil G \rceil}$ in the second case. The case for undirected graphs is similar, where in particular one relation symbol $\mathrm{inc}(e, v)$ is used for the representation of incidence graphs. The above notation is as the one used in [Cou08].

In the sections below, we will need to deal with graphs, whose edges are labelled. We therefore define the following alternative notion of a graph. Let Σ be some finite alphabet, and let $G = (V, E)$ be a graph, where $E = (E_a)_{a \in \Sigma}$, and where the E_a, for each $a \in \Sigma$, partition the set E. The signatures $\sigma_{\lfloor G \rfloor}$ and $\tau_{\lceil G \rceil}$ are extended appropriately to reflect this change in the way edges are represented. In particular, the signature $\sigma_{\lfloor G \rfloor}$ comprises a binary relation E_a for each $a \in \Sigma$, and the signature $\tau_{\lceil G \rceil}$ contains two binary relations inc_1^a and inc_2^a for each $a \in \Sigma$. We denote the modified versions of these signatures as $\sigma_{\Sigma, \lfloor G \rfloor}$ and $\tau_{\Sigma, \lceil G \rceil}$ respectively. Graphs with labelled edges will only be used for the representation of words and trees, and for reasons that will be explained in Section 1.2.1, there is no need to consider both graph representations for words and trees, and we will only consider the representation of graphs under the signature $\sigma_{\Sigma, \lfloor G \rfloor}$.

We define two properties of graphs that will be used later. We refer the reader to [Pap94] for more details. A Hamiltonian Cycle of a graph G, is a cycle that visits each vertex exactly once. A matching of a graph G is a set of non-adjacent edges, and a perfect matching is a matching were every vertex is adjacent to an edge of the matching.

In some results stated below, the notion of tree-width is briefly encountered and we proceed by presenting an informal introduction to this concept. For further details, we refer the reader to [Die05]. Tree decompositions and the tree-width of a graph were introduced by Robertson and Seymour ([RS86]) and independently by Halin ([Hal76]) and have been used extensively in Computer Science since. There are problems that are intractable over graphs in general, but when restricted to classes of graphs of bounded tree width they become tractable. Essentially, the tree-width of a graph is a measure for how much the graph resembles a tree. In a tree decomposition T of a graph G, vertices of the graph are grouped together, in such a way that each vertex corresponds to a connected subtree of the tree T, and each vertex of T corresponds to a set of vertices of G. Hence, for many problems, an algorithm can be built that travels through

the tree T in a top-down or bottom-up manner, to compute the required result. This is essential for dynamic programming algorithms that are based on finding solutions to subproblems of the given problem. A graph is of tree width k, if there is a tree decomposition T of G, such that each vertex of T corresponds to a set of vertices of G of size at most $k-1$.

On classes of graphs that contain only graphs of tree-width less than k, for some fixed $k \in \mathbb{N}$, several problems become easier. Such a class of graphs is said to be of *bounded tree-width*. One such result presented below in Section 1.2.1, regarding classes of graphs of bounded tree-width, is that the two versions of MSO that differ on the ability of quantifying over sets of edges, have the same expressive power on classes of graphs of bounded tree-width.

1.1.2 Words

A word is a sequence of letters from some fixed alphabet Σ and they are interchangeably called strings both in the literature in general and in this thesis. Throughout this thesis we only consider finite words. We present some of the ways words are represented as structures in the literature. For words over some alphabet Σ the two signatures $\tau_s = \langle \mathrm{succ}^2, (P_a^1)_{a\in\Sigma}\rangle$ and $\tau_o = \langle <^2, (P_a^1)_{a\in\Sigma}\rangle$ are used. For example, in [Tho96], structures of signature τ_o are considered, with an additional successor relation succ as the one in signature τ_s, which can be defined in FO using the order relation $<$. In [Lib04] and [EF99], words are represented as structures of signature τ_o. A word structure $\mathfrak{W} \in \mathrm{STRUC}[\tau_s]$, with universe W, is such that for any $v \in W$, exactly one of the unary relations P_a holds, and succ is a binary relation representing the partial successor function. Similarly a word structure $\mathfrak{W} \in \mathrm{STRUC}[\tau_o]$ is as for τ_s, but with the successor relation being replaced with a linear order over the elements of the word, representing the order of the elements as they appear in the word.

Example. The word *aabcb* over the alphabet $\Sigma = \{a, b, c\}$, can be represented as a word structure $\langle \{1,2,3,4,5\}, <^2, P_a^1, P_b^1, P_c^1\rangle$, where $<$ is the usual linear order on natural numbers restricted to $\{1,2,3,4,5\}$ and $P_a(1),\ P_a(2),\ P_b(3),\ P_c(4),\ P_b(5)$ hold. If the successor relation is used instead of the linear order, then $\mathrm{succ}(i, i+1)$ holds for $1 \leq i \leq 4$.

The second way of representing words, and the one which is used throughout this thesis, is the one where words are seen as connected, directed graph structures. Namely, for words over some alphabet Σ, the signature for word structures is defined as $\langle (E_a^2)_{a\in\Sigma}, \rangle$, where for each $a \in \Sigma$, E_a is a binary relation defined to hold for any two vertices that are connected with an edge labelled with a.

Example. The word in the example above is represented as a graph by the structure $G = \langle V, E_a^2, E_b^2, E_c^2\rangle$ where $V = \{1,2,3,4,5,6\}$, and $E_a = \{(1,2),(2,3)\},\ E_b = \{(3,4),(5,6)\}$ and $E_c = \{(4,5)\}$.

We saw above, that two different signatures are used for representing graphs. For reasons to be explained below, there is no need to consider both these representations, and unless otherwise specified a word is seen as a graph under the signature $\sigma_{\lfloor G \rfloor}$.

1.1.3 Trees

Trees are in general acyclic simple graphs and the first classification they undergo in the literature is based on whether the neighbours of each vertex are ordered. For the case where the neighbours of each vertex are ordered the trees are called ordered trees and are defined as follows. Let a prefix of a sequence s be an initial subsequence of s. The set of sequences formed from elements of some set S, is denoted as S^*. A set $Z \subseteq S^*$ is prefix-closed if for any sequence $s \in Z$, the set of prefixes of s is a subset of Z. A tree T over an alphabet Σ, is a prefix-closed domain $D \subseteq \mathbb{N}^*$ together with a labelling function $f : D \to \Sigma$. A class of rank-k ordered trees, for $k \in \mathbb{N}$, is a class of trees all of whose domains are subsets of S^*, where S is equal to $\{1, \ldots, k\}$. Otherwise, when the domain is a subset of $\mathbb{N}^*$, the trees are unranked.

On the other hand, unordered trees over an alphabet Σ are defined as $T = (V, <, f)$ where $<$ is a partial order on the set V, such that for any $v \in V$, the set $\{v' \mid v' < v\}$ is linearly ordered. A vertex v_2 is a child of another vertex v_1 if and only if $v_1 < v_2$ and there exists no v_3 such that $v_1 < v_3 < v_2$. Finally, $f : V \to \Sigma$ is the function labelling the vertices of the tree. There are a few widespread ways of representing trees such as the ones described above, which we consider below.

Ranked trees: In general, for ranked ordered trees of maximum rank k, for $k \in \mathbb{N}$, over some alphabet Σ, the signature $\tau_{r,o} = \langle \prec^2, (\text{child}_i^2)_{i \leq k}, (P_a^1)_{a \in \Sigma} \rangle$ is widely used (for example, in [Lib04] and [Tho96]). In a tree structure $\mathfrak{T}$ of signature $\tau_{r,o}$, the relation child_i contains (v_1, v_2), for $v_1, v_2 \in T$, if v_2 is the i-th child of v_1, and for each $a \in \Sigma$, P_a contains all vertices that are labelled with a. The binary relation $\prec$ defines a partial order on the vertices of $\mathfrak{T}$ and is defined as the transitive closure of the child relations. For ranked trees that are not ordered, the set of relations $(\text{child}_i^2)_{i \leq k}$ is replaced by a single binary relation, child^2, defining the child relation, and the resulting signature is denoted as $\tau_{r,u}$.

Unranked trees: For unranked trees over some alphabet Σ, one commonly used signature in the literature is $\tau_{u,o} = \langle \prec^2, \text{first-child}^2, <^2, \text{next-sibling}^2, (P_a^1)_{a \in \Sigma} \rangle$, where $\prec$ is as above and $<$ is a linear order ordering the children of each vertex. Similarly, first-child holds for any two vertices v_1, v_2 when v_2 is the first child of v_1, according to the order of the children, and the binary relation next-sibling contains (v_1, v_2), if v_2 is the successor of v_1 in the order of the children. This is the signature used for unranked ordered trees in [Lib04], omitting the relations first-child and next-sibling, which are FO-definable from the order relations in the signature. For unordered unranked trees, the simple signature $\tau_{u,u} = \langle \prec^2, \text{child}^2, (P_a^1)_{a \in \Sigma} \rangle$ can be used.

In all the above signatures, the binary relation $\prec$ is sometimes omitted. This does not make a difference on the expressiveness of Monadic Second-Order Logic, but it does so for First-Order Logic.

Trees as graphs: As graphs, trees are defined to be acyclic directed or undirected connected graphs, where in the case of directed graphs each vertex has in-degree at most 1. In this

thesis, unless otherwise specified, trees are represented as directed graphs, with the edges being labelled instead of the vertices. In other words, instead of the signature $\sigma_{\lfloor G \rfloor}$, labelled trees are represented using graph structures of signature $\sigma_{\Sigma,\lfloor G \rfloor}$, that is defined in Section 1.1.1. In particular, for trees with set of labels Σ, there is a binary relation E_a for each $a \in \Sigma$ that holds for any two vertices v_1, v_2 for which there is an edge from v_1 to v_2 labelled with a. Notice that since trees are represented as graphs, the signature contains no order relation, as in some of the cases described above.

As with words, by Theorem 1.2.4 that is stated below, there is no need to consider both ways defined in Section 1.1.1 for representing graphs. For the logics we are considering the expressive power remains the same regardless of the representation of trees as graphs structures used.

Notice that there is a natural way of mapping trees in the traditional sense, to trees with labels on the edges, since each vertex can be associated with the edge connecting it to its parent vertex, which in any case can be at most one. Therefore, all results can be seen to hold for trees defined in the traditional way.

Rank-k trees are also called k-ary and in particular binary trees are rank-2 trees with out-degree at most 2. Rank-k trees where all vertices have either exactly k children or are leaves, are called *proper*.

A forest is a collection of trees, or in other words an acyclic directed graph, possibly disconnected, where each vertex has in-degree at most 1. The vertices of the graph with in-degree 0, are called roots of the respective connected component they belong to, which is a tree.

1.2 Monadic Second-Order Logic

Monadic second-order logic (MSO) is a fragment of second-order logic, where quantification only of set variables is allowed instead of relation variables of arbitrary arity. It is a well-studied logic, that corresponds exactly to finite automata over trees and words. The data complexity of MSO over classes of graphs of bounded tree-width has been shown by Courcelle in [Cou90] to be linear with respect to the size of the structures, and the algorithm is bounded in time by $f(|\varphi|) \cdot p(|\mathfrak{A}|)$, with p a linear function. On trees and words this result can be established by translating MSO formulae into automata. The translation, however, is not computationally easy and it has been proved by Frick et al., in [FG04], that if there is a model checking algorithm with input an MSO formula φ and a word $\mathfrak{A}$ that is bounded in time by $f(|\varphi|) \cdot p(|\mathfrak{A}|)$ for p a polynomial and f elementary, then P collapses to NP. For a more in-depth introduction to MSO we refer the reader to [Cou08] and [Lib04].

MSO Syntax: Monadic second-order logic is an extension of FO with second-order quantification over second-order variables of arity 1. The syntax in MSO extends the one for FO appropriately. Suppose first that there are infinitely many second-order variables $X_1, X_2, \ldots$, ranging over sets. Since the semantics extend the ones for FO, first-order variables and quantification are allowed in MSO and $\varphi(\bar{x}, \bar{X})$ denotes that the MSO formula φ has $\bar{x}$ as its free

first-order variables and $\bar{X}$ as its free second-order variables. Terms are defined as in the FO case (first-order variables and constants) and MSO allows for all the atomic formulae of FO, is closed under boolean connectives and first-order quantification, and allows additionally for the following formation rules of formulae.

- If X is a monadic second-order variable and t a term, then $X(t)$ is an atomic MSO formula.
- If $\varphi(\bar{x}, \bar{X}, Y)$ is an MSO formula then $\exists Y\ \varphi(\bar{x}, \bar{X}, Y)$ and $\forall Y\ \varphi(\bar{x}, \bar{X}, Y)$ are MSO formulae with free variables among $\bar{x}$ and $\bar{X}$.

MSO Semantics: The semantics of an MSO formula are the same as for FO for the common connectives, and they are defined as follows for the additional ones defined above. Let $\mathfrak{A}$ be some finite structure of some signature σ with universe A, and let ρ and ζ be two mappings, where ρ assigns elements of A to first order terms and ζ assigns subsets of A to second order variables.

- $\mathfrak{A} \models_{\rho,\zeta} X(t)$ if and only if $\rho(t) \in \zeta(X)$.
- $\mathfrak{A} \models_{\rho,\zeta} \exists Y\ \varphi(\bar{x}, \bar{X}, Y)$ if and only if there exists $C \subseteq A$ such that $\mathfrak{A} \models_{\rho,\zeta[Y \mapsto C]} \varphi(\bar{x}, \bar{X})$.
- $\mathfrak{A} \models_{\rho,\zeta} \forall Y\ \varphi(\bar{x}, \bar{X}, Y)$ if and only if for all $C \subseteq A$ it holds that $\mathfrak{A} \models_{\rho,\zeta[Y \mapsto C]} \varphi(\bar{x}, \bar{X})$.

Throughout this thesis, given a structure $\mathfrak{A}$ with domain A over a signature σ, and given a formula $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ over the same signature with free first order variables $x_1, \ldots, x_n$ and free second order variables $X_1, \ldots, X_m$, then for any $a_1, \ldots, a_n \in A$ and any $B_1, \ldots, B_m \subseteq A$ we may write $\mathfrak{A} \models \varphi(a_1, \ldots, a_n, B_1, \ldots, B_m)$ instead of $\mathfrak{A} \models_{\rho,\zeta} \varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ where $\rho(x_i) = a_i$ and $\zeta(X_{i'}) = B_{i'}$ for $i \in \{1, \ldots, n\}, i' \in \{1, \ldots, m\}$.

In Section 1.2.1 below, where MS_2 is defined, slightly different syntax and semantics will be presented for the second-order variables and quantifiers over graphs.

The quantifier rank of an MSO formula φ denotes the maximum depth of quantification in a formula and is defined inductively as follows. For φ an MSO formula $\mathrm{qr}(\varphi)$ denotes the quantifier rank of φ.

If φ is a literal	then $\mathrm{qr}(\varphi) = 0$.
If $\varphi = \neg\psi$	then $\mathrm{qr}(\varphi) = \mathrm{qr}(\psi)$.
If $\varphi = \psi_1 \wedge \psi_2$	then $\mathrm{qr}(\varphi) = \max(\mathrm{qr}(\psi_1), \mathrm{qr}(\psi_2))$.
If $\varphi = \exists x\ \psi$	then $\mathrm{qr}(\varphi) = \mathrm{qr}(\psi) + 1$.
If $\varphi = \exists X\ \psi$	then $\mathrm{qr}(\varphi) = \mathrm{qr}(\psi) + 1$.

Let MSO^k denote the set of all MSO formulae of quantifier rank k. Then an MSO rank-k m, l-type, is a consistent set S of formulae in MSO^k with m free second-order variables and l free first-order variables, such that for any formula $\varphi(x_1, \ldots, x_l, X_1, \ldots, X_m) \in \mathrm{MSO}^k$, either $\varphi \in S$ or $\neg\varphi \in S$.

For $\mathfrak{A}$ a structure and $\bar{a}$ a tuple of elements of the universe A of $\mathfrak{A}$ with $|\bar{a}| = l$, and $\bar{V}$ a tuple of sets of elements of A with $|\bar{V}| = m$, the MSO rank-k m, l-type of $(\mathfrak{A}, \bar{a}, \bar{V})$ is denoted by $\text{mso-tp}_k^{m,l}(\mathfrak{A}, \bar{a}, \bar{V})$, and is defined as $\text{mso-tp}_k^{m,l}(\mathfrak{A}, \bar{a}, \bar{V}) = \{\varphi(\bar{x}, \bar{X}) \mid \mathfrak{A} \models \varphi(\bar{a}, \bar{V}), \text{qr}(\varphi) \leq k\}$. Informally, the latter is the set of formulae of quantifier rank less than or equal to k, with l free first-order variables and m free second-order variables, that are satisfied by the structure $\mathfrak{A}$, with the free variables of each formula being interpreted by $\bar{a}$ and $\bar{V}$ in $\mathfrak{A}$. If τ is the type $\text{mso-tp}_k^{m,l}(\mathfrak{A}, \bar{a}, \bar{V})$, we say then that $(\bar{a}, \bar{V})$ realize the type τ in $\mathfrak{A}$.

When two structures $\mathfrak{A}$ and $\mathfrak{B}$ agree on all formulae of quantifier rank k, or equivalently when they are of the same rank-k $0, 0$-type, we write $\mathfrak{A} \equiv_k^{\text{MSO}} \mathfrak{B}$. We may also refer to the MSO rank-k $0, 0$-type of a structure $\mathfrak{A}$, simply as the MSO-k-type of $\mathfrak{A}$. For two structures $\mathfrak{A}$ and $\mathfrak{B}$ such that $\mathfrak{A} \equiv_k^{\text{MSO}} \mathfrak{B}$, we say that $\mathfrak{A}$ and $\mathfrak{B}$ are $\equiv_k^{\text{MSO}}$-equivalent. Similarly, if for two structures $\mathfrak{A}$ and $\mathfrak{B}$ it holds that $\text{mso-tp}_k^{m,l}(\mathfrak{A}, \bar{a}, \bar{V}) = \text{mso-tp}_k^{m,l}(\mathfrak{B}, \bar{b}, \bar{W})$, for $|\bar{a}| = |\bar{b}| = l$ and $|\bar{V}| = |\bar{W}| = m$ being tuples of elements and sets of elements respectively of the corresponding structures $\mathfrak{A}$ and $\mathfrak{B}$, then we write $(\mathfrak{A}, \bar{a}, \bar{V}) \equiv_k^{\text{MSO}} (\mathfrak{B}, \bar{b}, \bar{W})$. The corresponding types for FO are denoted by $\text{fo-tp}_k^l(\mathfrak{A}, \bar{a})$.

Ehrenfeucht-Fraïssé Games: Ehrenfeucht-Fraïssé (EF) Games is the main tool for proving inexpressibility of a property in FO and it has been extended and adapted for additional logics such as MSO or GL, as we will see below. EF Games are based on Ehrenfeucht and Fraïssé's Theorems as presented in [EF99] and [Lib04].

FO EF Games: EF Games are played on two structures $\mathfrak{A}$ and $\mathfrak{B}$ of the same signature σ, by two players, called Spoiler and Duplicator, and consists of k rounds for some fixed $k \in \mathbb{N}$. At each round, Spoiler chooses a structure, say $\mathfrak{A}$, and picks an element, a, of the universe. Duplicator responds by choosing an element b of the universe of the other structure $\mathfrak{B}$. Let $\bar{a} = a_1, \ldots, a_k$ and $\bar{b} = b_1, \ldots, b_k$, be the elements picked in the structures $\mathfrak{A}$ and $\mathfrak{B}$ respectively, during the k rounds of the game. Duplicator wins if the mapping $a_i \mapsto b_i$, for all $i \leq k$, is a partial isomorphism on $\mathfrak{A}$ and $\mathfrak{B}$. Such a mapping is a partial isomorphism if the following hold:

1. For all $i, j \leq k$, $a_i = a_j$ if and only if $b_i = b_j$.
2. For all $i \leq k$ and for each constant in the signature σ, $a_i = c^{\mathfrak{A}}$ if and only if $b_i = c^{\mathfrak{B}}$.
3. For each relation P of arity r in the signature σ, $(a_{i_1}, \ldots, a_{i_r}) \in P^{\mathfrak{A}}$ if and only if $(b_{i_1}, \ldots, b_{i_r}) \in P^{\mathfrak{B}}$, for all $i_1, \ldots, i_r \leq k$.

It has been shown that types and EF Games are closely related according to the following Theorem.

Theorem 1.2.1 (Ehrenfeucht-Fraïssé)**.** *For any relational signature σ, any structures $\mathfrak{A}, \mathfrak{B} \in$ STRUC$[\sigma]$ with sequences of elements $\bar{c}, \bar{d}$ respectively, and any $k \in \mathbb{N}$, Duplicator wins the k-round EF Game on $(\mathfrak{A}, \bar{c})$ and $(\mathfrak{B}, \bar{d})$, if and only if fo-tp$_k^l(\mathfrak{A}, \bar{c}) =$ fo-tp$_k^l(\mathfrak{B}, \bar{d})$, with $l = |\bar{c}| = |\bar{d}|$.*

Therefore, to prove that a property P of structures of some signature σ, is not expressible in FO, it is sufficient to define for each $k \in \mathbb{N}$, two structures $\mathfrak{A}_k$ and $\mathfrak{B}_k$, such that $\mathfrak{A}_k \in P$ and $\mathfrak{B}_k \notin P$, and Duplicator has a winning strategy for the k-round EF Game on these two structures.

MSO EF Games: EF games for MSO are defined as an extension of the ones for FO. In particular, at each round of a k-round game on two structures $\mathfrak{A}$ and $\mathfrak{B}$, Spoiler has the choice of making a first-order move as defined above, or a colouring move. In the first case, the players move as they do in the first-order EF games. In the second case, Spoiler selects a set of elements in the structure of his choice and Duplicator replies in a similar way in the other structure.

Let $\bar{a} = a_1, \ldots, a_{k_1} \in A$ and $\bar{b} = b_1, \ldots, b_{k_1} \in B$ be the elements selected during the first-order rounds of the game, and let $A_1, \ldots, A_{k_2} \subseteq A$ and $B_1, \ldots, B_{k_2} \subseteq B$ be the sets selected during the colouring rounds, for k_1, k_2 such that $k_1 + k_2 = k$. Then Duplicator wins if $(\bar{a}, \bar{b})$ defines a partial isomorphism from $(\mathfrak{A}, A_1, \ldots, A_{k_2})$ to $(\mathfrak{B}, B_1, \ldots, B_{k_2})$. In other words, in addition to every relation of the signature of the structures, the sets A_i, B_i, for $i \leq k_2$, have to be considered in the partial isomorphism.

As for the FO case, it can be shown that Duplicator has a winning strategy for the MSO EF k-round game on $\mathfrak{A}$ and $\mathfrak{B}$ if and only if $\mathfrak{A} \equiv_k^{\text{MSO}} \mathfrak{B}$.

The second-order move described above is called a colouring move, since at each round the players colour the elements of each structure by including them in the set or not. After k rounds each element has one of the 2^k possible colourings. We state below the corresponding Theorem for MSO and MSO EF Games and refer the reader to [EF99] for further details.

Theorem 1.2.2. *For any relational signature σ, any structures $\mathfrak{A}, \mathfrak{B} \in \text{STRUC}[\sigma]$ with sequences of elements $\bar{a}, \bar{b}$ respectively and sequences of sets of elements $\bar{V}, \bar{W}$, and any $k \in \mathbb{N}$, Duplicator wins the k-round* MSO *EF Game on $(\mathfrak{A}, \bar{a}, \bar{V})$ and $(\mathfrak{B}, \bar{b}, \bar{W})$, if and only if mso-$tp_k^{m,l}(\mathfrak{A}, \bar{a}, \bar{V}) = $ mso-$tp_k^{m,l}(\mathfrak{B}, \bar{b}, \bar{W})$, with $m = |\bar{V}| = |\bar{W}|$ and $l = |\bar{a}| = |\bar{b}|$.*

We recall the following special case of the Feferman-Vaught theorem ([FV59]) that will be used later. More on the composition method can be found in [Mak04] and [Tho97].

Lemma 1.2.3. *If $(\mathfrak{A}_1, \bar{a}) \equiv_k^{\text{MSO}} (\mathfrak{B}_1, \bar{b})$ and $(\mathfrak{A}_2, \bar{a}) \equiv_k^{\text{MSO}} (\mathfrak{B}_2, \bar{b})$ then*

$$(\mathfrak{A}_1 \oplus_{\bar{a}} \mathfrak{A}_2, \bar{a}) \equiv_k^{\text{MSO}} (\mathfrak{B}_1 \oplus_{\bar{b}} \mathfrak{B}_2, \bar{b}).$$

Proof. Let $(\mathfrak{A}, \bar{a}) = (\mathfrak{A}_1 \oplus_{\bar{a}} \mathfrak{A}_2, \bar{a})$ and $(\mathfrak{B}, \bar{b}) = (\mathfrak{B}_1 \oplus_{\bar{b}} \mathfrak{B}_2, \bar{b})$. It suffices to show that Duplicator wins the k-round MSO EF Game on the two structures $(\mathfrak{A}_1 \oplus_{\bar{a}} \mathfrak{A}_2, \bar{a})$ and $(\mathfrak{B}_1 \oplus_{\bar{b}} \mathfrak{B}_2, \bar{b})$. By assumption Duplicator wins the k-round game on the two structures $(\mathfrak{A}_1, \bar{a})$ and $(\mathfrak{B}_1, \bar{b})$ and also on the two structures $(\mathfrak{A}_2, \bar{a})$ and $(\mathfrak{B}_2, \bar{b})$. Suppose that at some round $i \leq k$, the position of the game is $\langle(\mathfrak{A}, \bar{a}, \bar{c}, \bar{C}), (\mathfrak{B}, \bar{b}, \bar{d}, \bar{D})\rangle$, and Spoiler selects a set of vertices X_i on one of the two structures, say $\mathfrak{A}$. Then let $X_i = X_i^1 \cup X_i^2$, and let X_i^1 be the subset of X_i of vertices in $\mathfrak{A}_1$ and similarly X_i^2 the subset of X_i of vertices in $\mathfrak{A}_2$. By assumption Duplicator has a

response Y_i^1 for X_i^1 on the game $\langle(\mathfrak{A}_1, \bar{a}, \bar{c}, \bar{C}), (\mathfrak{B}_1, \bar{b}, \bar{d}, \bar{D})\rangle$ and similarly a response Y_i^2 for X_i^2 on the game $\langle(\mathfrak{A}_2, \bar{a}, \bar{c}, \bar{C}), (\mathfrak{B}_2, \bar{b}, \bar{d}, \bar{D})\rangle$. The response of Duplicator to X_i is then $Y_i^1 \cup Y_i^2$ and $(\mathfrak{A}, C_1, \ldots, C_{k_2}, X_i, \bar{a}) \equiv_{k-i}^{\mathrm{MSO}} (\mathfrak{B}, D_1, \ldots, D_{k_2}, Y_i^1 \cup Y_i^2, \bar{b})$. Duplicator's response to first-order moves is similar.

Suppose that at the end of the game $\bar{c} = c_1, \ldots, c_{k_1}$ and $\bar{d} = d_1, \ldots, d_{k_1}$ are the first order moves made on $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$ respectively, and let $C_1, \ldots, C_{k_2}$ and $D_1, \ldots, D_{k_2}$ be the second order moves made on $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$ respectively. Then the mapping $c_i \mapsto d_i$, for all $i \leq k_1$ is a partial isomorphism from $(\mathfrak{A}_1, C_1, \ldots, C_{k_2}, \bar{a})$ to $(\mathfrak{B}_1, D_1, \ldots, D_{k_2}, \bar{b})$ and from $(\mathfrak{A}_2, C_1, \ldots, C_{k_2}, \bar{a})$ to $(\mathfrak{B}_2, D_1, \ldots, D_{k_2}, \bar{b})$, and therefore from $(\mathfrak{A}, C_1, \ldots, C_{k_2}, \bar{a})$ to $(\mathfrak{B}, D_1, \ldots, D_{k_2}, \bar{b})$, hence Duplicator wins the game. In the above, in $(\mathfrak{A}_i, C_1, \ldots, C_{k_2}, \bar{a})$, we use the restrictions of $C_1, \ldots, C_{k_2}$ to the structure $\mathfrak{A}_i$. □

1.2.1 MS_1 and MS_2

On graphs, MSO turns out to have different expressive power depending on the signature of the structures used to represent graphs. In section 1.1.1, the two different signatures $\sigma_{\lfloor G \rfloor}$ and $\tau_{\lceil G \rceil}$ were introduced, where in the latter case incidence graphs are considered. Essentially, MSO in the second case, allows for second-order quantification over edges, with which certain properties can be expressed that are not expressible otherwise. In particular, as was proved by Turán ([Tur84]) and de Rougemont ([dR87]), the class of graphs that have a Hamiltonian cycle or of the ones with a perfect matching, is not definable in the version of MSO without set quantification over edges but is definable in the version with such quantification.

In the works of Courcelle, such as [Cou97] and [Cou03], these two ways of representing a graph are investigated, and in particular, the expressive power of MSO over such structures is discussed. To distinguish between the two situations, Courcelle uses MS_1 for MSO over graphs of the signature $\sigma_{\lfloor G \rfloor}$, and MS_2 for $\tau_{\lceil G \rceil}$. In [GHO02], the latter logic is referred to as *guarded second order logic* (GSO). Over graphs in general, $\mathrm{MS}_1 \subsetneq \mathrm{MS}_2$, but, in [Cou03] and [Cou94], it is shown that over restricted classes of graphs the expressive power of the two logics coincide. Before we state the Theorem in [Cou03], we give a few necessary definitions.

A finite graph $G = (V, E)$ is *k-sparse* if $|E| \leq k \cdot |V|$. A finite graph G is *uniformly k-sparse* if every subgraph of G is k-sparse. The graphs in any class of graphs of bounded degree, as well as any class of planar graphs and any class of graphs of bounded tree-width are uniformly k-sparse for some $k \in \mathbb{N}$.

Theorem 1.2.4 (Courcelle, [Cou03])**.** *For each integer k, one can effectively transform a given monadic second-order formula using edge set quantifications into one that uses only vertex set quantifications and is equivalent to the given one on finite, uniformly k-sparse, simple directed or undirected graphs.*

The above definitions and Theorem are given in [Cou03] more generally for finite or countable graphs, and furthermore for hypergraphs satisfying certain conditions. Throughout most

of this thesis, where classes of graphs are investigated for which $\mathrm{MS}_1 = \mathrm{MS}_2$ holds, we simply write MSO. In cases when it is clear, a graph will simply be denoted by $G = (V, E)$, where V is the set of vertices and E the set of edges of a graph. Unless otherwise specified, graphs will be represented as structures with signature $\sigma_{\lfloor G \rfloor}$. Notice that even when graphs are represented in this way, MS_2 is well defined. In particular, it allows for additional atomic formulae of the form $Z(t_1, t_2)$, for terms t_1, t_2, and quantifiers $\exists Z\ \varphi$, where the latter formula holds in a graph $G = (V, E)$ if there exists a subset $A \subseteq E$ of edges such that φ is true in G when all free occurrences of Z are interpreted by the set A. For classes of graphs where $\mathrm{MS}_1 = \mathrm{MS}_2$, the two different forms of atomic formulae and quantifiers may be used interchangeably, since a translation of formulae can be produced for both directions.

MS_2 EF Games: EF Games for MS_2 on structures of signature $\sigma_{\lfloor G \rfloor}$ are defined similarly to the MSO EF Games described in Section 1.2, but with the colouring move changed so that Spoiler can select at each round a set of edges in one of the two structures, and Duplicator responds accordingly with a set of edges in the other structure. Similarly, in first order moves, Spoiler can select an edge of one of the two graphs, and Duplicator responds with an edge in the other graph. The condition for winning is extended to accommodate the second-order and first-order moves on edges of the graph.

A lemma similar to Lemma 1.2.3 holds for MS_2 as well. A lemma that will be needed later is the following. If $G = (V, E)$ is a graph with vertices V and edges E, and if $A \subseteq E$, then $G{\downarrow_A}$, is the subgraph of G that contains exactly the edges in A.

Lemma 1.2.5. *For any two graphs $G = (V, E)$ and $F = (V', E')$, if $G \equiv_k^{\mathrm{MS}_2} F$, then for any $A \subseteq E$ there exists $A' \subseteq E'$ such that $G{\downarrow_A} \equiv_{k-1}^{\mathrm{MS}_2} F{\downarrow_{A'}}$.*

Proof. Since $G \equiv_k^{\mathrm{MS}_2} F$, for any colouring move A of Spoiler on G, Duplicator can respond with a set of edges A' of F such that $(G, A) \equiv_{k-1}^{\mathrm{MS}_2} (F, A')$. Now suppose for contradiction that $G{\downarrow_A}$ and $F{\downarrow_{A'}}$ are not $\equiv_{k-1}^{\mathrm{MS}_2}$-equivalent. Then there is an MS_2 formula φ of quantifier rank $k-1$ that distinguishes between the two graphs. We show that an MS_2 formula ψ of quantifier rank $k-1$ can be constructed that distinguishes the structures (G, A) and (F, A'), which is a contradiction. Without loss of generality assume that φ is such that $G{\downarrow_A} \models \varphi$ and $F{\downarrow_{A'}} \models \neg\varphi$.

We define a translation by induction on the structure of the MS_2 formula φ. All boolean and atomic cases as well as the case of second-order quantification of edges and vertices remain the same. The translation of formulae whose outermost connective is first-order quantification is as follows. For each MS_2 formula, $[\varphi]^X$ denotes the translation.

$$\begin{aligned} [\exists e\ \varphi_1]^X &= \exists e\ (e \in X \wedge [\varphi_1]^X), \\ [\forall e\ \varphi_1]^X &= \forall e\ (e \in X \rightarrow [\varphi_1]^X). \end{aligned}$$

The formula $\psi = [\varphi]^A$ distinguishes between the two structures (G, A) and (F, A'). □

1.3 Word and Tree Automata

Since words and trees are represented as graphs with labels on the edges instead of the vertices, a modified definition of automata on such structures needs to be stated, but because of the natural correspondence between the two representations, this does not pose a significant problem.

Definition 1.3.1. *A nondeterministic bottom-up tree automaton over ranked trees of maximum rank k, for some fixed $k \in \mathbb{N}$, is a tuple $A = (Q, \Sigma, q_0, (\delta_i)_{i \leq k}, Q_f)$, where Q is a finite set of states, q_0 is the initial state, Q_f is a subset of Q denoting the accepting states, and for each $1 \leq i \leq k$, $\delta_i : (Q \times \Sigma)^i \to 2^Q$ is the transition function for the vertices with i children.*

A deterministic bottom-up tree automaton is as above, but with each transition function δ_i being of type $\delta_i : (Q \times \Sigma)^i \to Q$. The language accepted by an automaton A is denoted as $L(A)$.

If T is a ranked tree over the alphabet Σ, a *run* of the automaton A on T is a function $\rho : V(T) \to Q$ that associates a state with each vertex of T and satisfies the following condition. If $v \in V(T)$ is a leaf vertex, then $\rho(v) = q_0$, and for any vertex $v \in V(T)$ with children $v_1, \ldots, v_j$, for $j \leq k$, and corresponding labels $\sigma_1, \ldots, \sigma_j$ on the edges connecting them to v, then $\rho(v) \in \delta_j(\rho(v_1), \sigma_1, \ldots, \rho(v_j), \sigma_j)$. In the case of deterministic automata, the last condition is changed to $\rho(v) = \delta_j(\rho(v_1), \sigma_1, \ldots, \rho(v_j), \sigma_j)$. A run of A on a tree T with root r is *accepting* if $\rho(r) \in Q_f$. An accepting language of an automaton A is the set of trees for which there exists an accepting run.

When it is clear from the context, for a tree automaton $A = (Q, \Sigma, q_0, (\delta_i)_{i \leq k}, Q_f)$, we denote the collection of transition functions $(\delta_i)_{i \leq k}$, simply as δ.

Notice that word automata under this definition are a special case of tree automata, namely of rank-1 trees, where each vertex has at most one child. Word automata, deterministic or nondeterministic, are usually defined to be top-down and not bottom-up as in this case, but a natural correspondence exists between the two forms.

Definition 1.3.2. *A tree language is regular if and only if it is accepted by some nondeterministic bottom-up tree automaton A.*

One important Theorem by Thatcher and Wright ([TW68]) and Doner ([Don70]) is the following.

Theorem 1.3.3 (Thatcher-Wright)**.** *A set of trees is definable in* MSO *if and only if it is regular.*

Definition 1.3.4. *Let $A = (Q^A, \Sigma, q_0^A, \delta^A, Q_f^A)$ and $B = (Q^B, \Sigma, q_0^B, \delta^B, Q_f^B)$ be two deterministic bottom-up tree automata, over the same alphabet Σ. The product $A \times B$ of the two automata is the deterministic bottom-up tree automaton $A \times B = (Q, \Sigma, q_0, Q_f, \delta)$ where $Q = Q^A \times Q^B$, $q_0 = (q_0^A, q_0^B)$, $Q_f = Q_f^A \times Q_f^B$ and $\delta = (\delta_i)_{i \leq k}$ is the collection of transition functions defined as follows.*

For each $i \leq k$, $\delta_i : (Q \times \Sigma)^i \to Q$, *and for any* $(q_1^A, q_1^B), \ldots, (q_{(i+1)}^A, q_{(i+1)}^B) \in Q$ *and* $\sigma_1, \ldots, \sigma_i \in \Sigma$, *it holds that* $\delta_i((q_1^A, q_1^B), \sigma_1, \ldots, (q_i^A, q_i^B), \sigma_i) = (q_{(i+1)}^A, q_{(i+1)}^B)$ *if and only if* $\delta_i^A(q_1^A, \sigma_1, \ldots, q_i^A, \sigma_i) = q_{(i+1)}^A$ *and* $\delta_i^B(q_1^B, \sigma_1, \ldots, q_i^B, \sigma_i) = q_{(i+1)}^B$.

If L_A is some tree language accepted by some automaton A and L_B some language accepted by some automaton B, then the language accepted by the product of the two automata $A \times B$, can be shown to be $L_A \cap L_B$.

1.4 Fixed-Point Logics

We give a brief introduction to the theory of fixed-points as well as LFP, a logic that extends FO with an additional operator based on fixed-points. We refer the reader to [AN01] for more information on fixed-point theory, and to [Lib04] for an in-depth introduction to LFP.

A complete lattice $(U, \leq)$ is a set U, partially ordered with respect to $\leq$, such that every subset of U has a greatest lower bound and a least upper bound in the ordering of $\leq$. A mapping $F : U \to U$ is defined to be an operator on U and furthermore the operator F is said to be monotone if for all elements x, y of U, it holds that

$$x \leq y \Rightarrow F(x) \leq F(y).$$

An element x of U is a fixed point of F, if $F(x) = x$ and it is a least fixed point of F if it is a fixed point of F and is such that for any other fixed point y of F, $x \leq y$. The least fixed point of F is denoted by $\mathbf{lfp}(F)$, and $x \sqcap y$ denotes the greatest lower bound of x and y.

Theorem 1.4.1 (Tarski-Knaster)**.** *Let* $(U, \leq)$ *be a complete lattice and* $F : U \to U$ *a monotone operator on* U. *Then* F *has a least fixed point and furthermore is such that*

$$\mathbf{lfp}(F) = \bigsqcap\{y \mid F(y) = y\} = \bigsqcap\{y \mid F(y) \leq y\}.$$

For the logics we are considering, the complete lattice is the powerset of a set. In particular, let U be a set. Then the powerset $\wp(U)$ of U is a complete lattice, since it is partially ordered by the set inclusion relation, and any subset of $\wp(U)$ has a least upper bound and a greatest lower bound according to the set inclusion relation. An operator $F : \wp(U) \to \wp(U)$ is monotone in such a lattice if for all subsets X, Y of U, $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$. For such a monotone operator F, Theorem 1.4.1 guarantees the existence of a least fixed point and is defined as

$$\mathbf{lfp}(F) = \bigcap\{Y \mid F(Y) = Y\} = \bigcap\{Y \mid F(Y) \subseteq Y\}.$$

For what follows, we consider complete lattices that are powersets of sets (not necessarily finite), and we use the set inclusion relation to order the elements of such a lattice.

Let $X^0 = \emptyset$, and for each $k \in \mathbb{N}$, let $X^{k+1} = F(X^k)$. Let also $X^\infty = \bigcup_{k \geq 0} X^k$. It can be shown that for every monotone operator F on a complete lattice $(U, \leq)$, the least fixed point of F is equal to X^∞. In other words,

$$\mathbf{lfp}(F) = X^\infty = \bigcup_{k \geq 0} X^k.$$

Logics such as LFP deal only with finite sets, but for logics such as GL_μ and LFLC defined below, infinite complete lattices need to be taken into account.

LFP: Let $\varphi(X, \bar{x}, \bar{y})$ be an FO formula of signature σ, with X being a relation on variables with arity equal to $|\bar{x}| = k$, for some $k \in \mathbb{N}$. For each $\mathfrak{A}$ and each tuple $\bar{b}$ where $|\bar{b}| = |\bar{y}|$, we define the operator $F^{\bar{b}}_{\varphi} : \wp(A^k) \to \wp(A^k)$ as $F^{\bar{b}}_{\varphi}(R) = \{(a_1, \ldots, a_k) \mid \mathfrak{A} \models \varphi(R, a_1, \ldots, a_k, \bar{b})\}$, with $R \subseteq A^k$.

Checking whether the operator associated with an FO formula is monotone, is undecidable, and therefore, in many cases the following sufficient syntactic restriction on the formulae is posed to ensure monotonicity. It can be shown that if X appears under an even number of negations in a formula $\varphi(X, \bar{x})$, then the operator F_φ is monotone, and this restriction will be used for the logic LFP as well as GL_μ defined below. In the case where X appears under an even number of negations in a formula $\varphi(X, \bar{x})$, we say that X appears positive in $\varphi(X, \bar{x})$.

The logic LFP then is defined to extend FO with the following rule for formulae. If $\varphi(X, \bar{x}, \bar{y})$ is an LFP formula of signature σ, with X being of arity $|\bar{x}|$, and X appears positive in $\varphi(X, \bar{x}, \bar{y})$, then $[\mathbf{lfp}_{X,\bar{x}}\varphi(X, \bar{x}, \bar{y})](\bar{t})$ is an LFP formula with free variables among the ones in $\bar{t}$ and the ones in $\bar{y}$. For any structure $\mathfrak{A} \in \mathrm{STRUC}[\sigma]$ and any tuple $\bar{b}$ with $|\bar{b}| = |\bar{y}|$

$$\mathfrak{A} \models [\mathbf{lfp}_{X,\bar{x}}\varphi(X, \bar{x}, \bar{y})](\bar{a}, \bar{b}) \text{ if and only if } \bar{a} \in \mathbf{lfp}(F^{\bar{b}}_{\varphi}).$$

1.5 Graph Logic

Graph Logic (GL) is a spatial query language on graphs introduced in [CGG02]. It is based on a view of graphs as a suitable algebra, with a composition operator. In this thesis it is treated as an extension of First Order Logic, with a second order quantifier over edges of restricted form. Informally, this second order quantifier allows one to express that a graph G can be split into two edge-disjoint subgraphs G_1, G_2, one of which satisfies a formula ϕ and the other satisfies some formula ψ. As is shown in [DGG07], treating Graph Logic as an extension of First-order Logic is equivalent to the view of the logic introduced in [CGG02].

To define GL we represent graphs as follows. Let V be a countable set of vertex names. A graph G consists of a finite set E of edges and an incidence map $I : E \to V \times V$, that associates with each edge a pair of vertices, namely its endpoints. Any such graph can be seen as a finite, directed, unlabelled graph in the usual sense. The syntax of Graph Logic is then as follows.

Definition 1.5.1 (GL Syntax)**.** *Let X be a countable set of vertex variables. A* GL *formula is defined to be one of the following, for $t_i \in V \cup X$, $x \in X$ and* GL *formulae φ_i:*

$$\varphi := \mathbf{0} \mid \top \mid t_1 = t_2 \mid E(t_1, t_2) \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \exists x\ \varphi_1 \mid \varphi_1|\varphi_2.$$

This seems to be simply an extension of FO with the additional operator '|', but in the case of GL, atomic formulae are interpreted slightly differently. In particular, for $t_1, t_2 \in X \cup V$, the GL formula $E(t_1, t_2)$ holds for a graph, exactly when this graph comprises a single edge.

The equivalent of the FO atomic formula $E(v_1, v_2)$, for two vertices v_1, v_2, is given in GL as $E(v_1, v_2) \mid \top$, as will be explained below.

Let G_1 and G_2 be two graphs, with edge sets E_1 and E_2 respectively, and incidence maps I_1 and I_2 respectively. The graph $G = G_1 \mid G_2$ is defined to be the graph whose edge set E and incidence map I are the disjoint unions $E_1 \uplus E_2$ and $I_1 \uplus I_2$ respectively. According to this, the two graphs G_1 and G_2 are allowed to share vertices, or in other words, the set of vertices of G is the (non-disjoint) union of the set of vertices of G_1 and G_2. The formula $\varphi_1 \mid \varphi_2$, for GL formulae φ_i, is defined to hold in a graph G, exactly when there are G_1 and G_2, such that $G = G_1 \mid G_2$, and $G_i \models_{\mathrm{GL}} \varphi_i$, for $i = 1, 2$.

Definition 1.5.2 (GL Semantics)**.** *Let G be a graph with edge set E and incidence map I. Let also $\alpha : X \to V$ be an assignment of vertex names to variables, and $\hat{\alpha}$ to be the extension of α to the domain $V \cup X$, by letting $\hat{\alpha}(x) = x$, for all $x \in V$. Then the following holds.*

$$
\begin{array}{lll}
(G,\alpha) \models_{\mathrm{GL}} \mathbf{0} & \textit{iff} & E \textit{ is empty,} \\
(G,\alpha) \models_{\mathrm{GL}} \top & \textit{iff} & G \textit{ is any graph,} \\
(G,\alpha) \models_{\mathrm{GL}} t_1 = t_2 & \textit{iff} & \hat{\alpha}(t_1) = \hat{\alpha}(t_2), \\
(G,\alpha) \models_{\mathrm{GL}} E(t_1, t_2) & \textit{iff} & E = \{e\} \textit{ and } I(e) = (\hat{\alpha}(t_1), \hat{\alpha}(t_2)), \\
(G,\alpha) \models_{\mathrm{GL}} \neg\varphi_1 & \textit{iff} & \textit{not } (G,\alpha) \models_{\mathrm{GL}} \varphi_1, \\
(G,\alpha) \models_{\mathrm{GL}} \varphi_1 \wedge \varphi_2 & \textit{iff} & (G,\alpha) \models_{\mathrm{GL}} \varphi_1 \textit{ and } (G,\alpha) \models_{\mathrm{GL}} \varphi_2, \\
(G,\alpha) \models_{\mathrm{GL}} \exists x\ \varphi_1 & \textit{iff} & \textit{there is } v \in V \textit{ s.t. } (G, \hat{\alpha}[x \mapsto v]) \models_{\mathrm{GL}} \varphi_1, \\
(G,\alpha) \models_{\mathrm{GL}} \varphi_1 \mid \varphi_2 & \textit{iff} & \textit{there are } G_1, G_2 \textit{ s.t. } G = G_1 \mid G_2 \textit{ and } (G_i,\alpha) \models_{\mathrm{GL}} \varphi_i.
\end{array}
$$

In spite of the different interpretation of atomic formulae in GL, any FO sentence can be translated into a GL sentence such that they both agree on all graphs. In particular, the atomic FO formula $E(v_1, v_2)$ is equivalent to the GL one $E(v_1, v_2) \mid \top$. On the other hand, it is shown in [DGG07], that any GL formula can be translated into an MS_2 one.

The version of GL introduced in [CGG02], works over graphs whose edges can have labels. Labels can be introduced into the language of GL in two different ways. If the set of labels Σ is finite, we can regard each label $\sigma \in \Sigma$ to define a set of edges E_σ with the incidence map $I : E_\sigma \to V \times V$. For each $\sigma \in \Sigma$ then, GL allows atomic formulae $E_\sigma(t_1, t_2)$. On the other hand, if the set of labels is unbounded, we include in the language the set Σ and a countable set of variable names L. In this case, a graph is a set of edges E with an incidence map $I : V \times \Sigma \times V$. For each $l, l' \in \Sigma \cup L$, $l'' \in L$ and GL formula φ, GL allows the atomic formulae $E_l(t_1, t_2)$ and $l = l'$, as well as quantification over edge labels with the formula of the form $\exists l''\ \varphi$. Marcinkowski termed this version of GL, GL+ in [Mar06], and showed that it is strictly less expressive than MSO+, the corresponding extension of MS_2.

For the case where the label alphabet Σ is bounded, many interesting applications of MSO are included. It was shown in [DGG07] that GL and MSO are equi-expressive over the class of words. Similarly, labelled edges on trees can be identified correspondingly with tree languages over some alphabet. It remains an open question whether GL defines all the regular tree languages.

We define the quantifier rank of a GL formula by counting the first order quantifiers and the splitting operators equally. In other words, the quantifier rank of a GL formula φ is denoted by $\mathrm{qr}(\varphi)$, and defined as usual for the boolean connectives, and as follows for the rest:

$$\begin{array}{lll} \text{If } \varphi = & \mathbf{0}, \top, t_1 = t_2, E(t_1, t_2) & \text{then } \mathrm{qr}(\varphi) = 0. \\ \text{If } \varphi = & \varphi_1 \mid \varphi_2 & \text{then } \mathrm{qr}(\varphi) = \max(\mathrm{qr}(\varphi_1), \mathrm{qr}(\varphi_2)) + 1. \\ \text{If } \varphi = & \exists x\ \psi(x) & \text{then } \mathrm{qr}(\varphi) = \mathrm{qr}(\psi) + 1. \end{array}$$

As stated above, any GL formula φ can be translated into an MS_2 formula ψ, such that over any graph G, $G \models_{\mathrm{GL}} \varphi$ if and only if $G \models_{\mathrm{MSO}} \psi$. We present a modified version of the translation defined in [DGG07]. We use a modified version so that $\mathrm{qr}(\psi) \leq \mathrm{qr}(\varphi) + 2$. The translation given in [DGG07] guarantees instead that $\mathrm{qr}(\psi) \leq 2 \cdot \mathrm{qr}(\varphi)$. For each GL formula φ, $[\![\varphi]\!]^S$, with S a set of edges, denotes its translation into MS_2 and is defined as follows.

Let **Set** be a set of set-expressions that is the smallest set, such that if S is a set, then $S \in \mathbf{Set}$, $\top \in \mathbf{Set}$, and for any $S_1, S_2 \in \mathbf{Set}$, $(S_1 \cap S_2) \in \mathbf{Set}$ and $(S_1 - S_2) \in \mathbf{Set}$. For $S \in \mathbf{Set}$, we define inductively the formula $e \in S$ to be shorthand notation for the following MS_2 formulae, depending on what the structure of S is.

$$\begin{array}{ll} \text{If } S \text{ is a set} & \text{then } e \in S \text{ is shorthand for } e \in S. \\ \text{If } S = \top & \text{then } e \in S \text{ is shorthand for } e = e. \\ \text{If } S = S_1 \cap S_2 \text{ for } S_1, S_2 \in \mathbf{Set} & \text{then } e \in S \text{ is shorthand for } e \in S_1 \wedge e \in S_2. \\ \text{If } S = S_1 - S_2 \text{ for } S_1, S_2 \in \mathbf{Set} & \text{then } e \in S \text{ is shorthand for } e \in S_1 \wedge e \notin S_2. \end{array}$$

Given that, we inductively define the actual translation to MS_2 below, with $S, S' \in \mathbf{Set}$.

$$\begin{array}{rcl} [\![\mathbf{0}]\!]^S & = & \neg\exists e\ e \in S, \\ [\![t_1 = t_2]\!]^S & = & t_1 = t_2, \\ [\![E_a(t_1, t_2)]\!]^S & = & \exists e\ (e \in S \wedge \mathrm{inc}_1^a(e, t_1) \wedge \mathrm{inc}_2^a(e, t_2) \wedge \forall e'\ (e' \in S \rightarrow e = e')), \\ [\![\neg\varphi]\!]^S & = & \neg[\![\varphi]\!]^S, \\ [\![\varphi_1 \wedge \varphi_2]\!]^S & = & [\![\varphi_1]\!]^S \wedge [\![\varphi_2]\!]^S, \\ [\![\exists x\ \varphi]\!]^S & = & \exists x\ [\![\varphi]\!]^S, \\ [\![\varphi_1 \mid \varphi_2]\!]^S & = & \exists S_1\ ([\![\varphi_1]\!]^{S_1 \cap S} \wedge [\![\varphi_2]\!]^{S - S_1}). \end{array}$$

The set S given as a parameter to the translation can be seen as the set of edges, to which the translation of the formulae are relativized. We remind the reader that $\mathrm{qr}(\varphi_1 \mid \varphi_2) = \max(\mathrm{qr}(\varphi_1), \mathrm{qr}(\varphi_2)) + 1$ and therefore all the connectives preserve the quantifier rank of the GL formula, and only the atomic formulae $E_a(t_1, t_2)$ and $\mathbf{0}$ increase the quantifier rank. Therefore for a GL formula φ, $\mathrm{qr}([\![\varphi]\!]^\top) \leq \mathrm{qr}(\varphi) + 2$.

The translation given in [DGG07] is similar, with the only inductive case that is different being the one for the separation connective. In [DGG07], the translation is given by

$$[\![\varphi_1 \mid \varphi_2]\!]^S = \exists S_1, S_2\ (S = S_1 \mid S_2) \wedge ([\![\varphi_1]\!]^{S_1} \wedge [\![\varphi_2]\!]^{S_2})$$

where $S = S_1 \mid S_2$ is shorthand notation for the MS_2 formula $\forall e\ (e \in S \rightarrow (e \in S_1 \vee e \in S_2)) \wedge (e \in S_1 \leftrightarrow e \notin S_2)$. Note that in this translation, the set of set-expressions **Set** is not

required. The proof of the correctness of the translation given above is similar to the one in [DGG07] and we skip the details. The following Theorem follows from that translation.

Theorem 1.5.3. *For any $k \in \mathbb{N}$, and any two graphs $\mathfrak{A}, \mathfrak{B}$, if $\mathfrak{A} \equiv^{\mathrm{MS}_2}_{k+2} \mathfrak{B}$ then $\mathfrak{A} \equiv^{\mathrm{GL}}_{k} \mathfrak{B}$.*

Proof. Fix k and suppose for contradiction that there are two graphs $\mathfrak{A}$ and $\mathfrak{B}$ such that $\mathfrak{A} \equiv^{\mathrm{MS}_2}_{k+2} \mathfrak{B}$ and it is not the case that $\mathfrak{A} \equiv^{\mathrm{GL}}_{k} \mathfrak{B}$. Then, there is a GL formula φ of quantifier rank k such that $\mathfrak{A} \models_{\mathrm{GL}} \varphi$ and $\mathfrak{B} \models_{\mathrm{GL}} \neg\varphi$. But then $\mathfrak{A} \models_{\mathrm{MSO}} [\![\varphi]\!]^{\top}$ and $\mathfrak{B} \models_{\mathrm{MSO}} \neg[\![\varphi]\!]^{\top}$, which is a contradiction. □

It should be noted that the following derived operator is defined in [DGG07] and is also used in this thesis. We write $\varphi \Rrightarrow \psi$ for $\neg(\varphi \mid \neg\psi)$, meaning that for any graph G, $G \models_{\mathrm{GL}} (\varphi \Rrightarrow \psi)$ if and only if for any G_1, G_2 such that $G = G_1 \mid G_2$, if $G_1 \models_{\mathrm{GL}} \varphi$ then $G_2 \models_{\mathrm{GL}} \psi$.

1.5.1 GL Games

The main tool for proving non-expressibility of a property in FO or MSO is Ehrenfeucht-Fraïssé games (EF games), as described above and in [Lib04] and [EF99]. EF games have been adapted for spatial logics such as in [DGG04], and here we present how they are adapted for GL.

A GL game is played by two players, which are called Spoiler and Duplicator. The game is played on two graphs F and G and consists of k rounds for some $k \in \mathbb{N}$. The position at each round i, is defined to be a pair of structures with distinguished vertices, $\langle (F^i, \bar{a}), (G^i, \bar{b}) \rangle$, where $\bar{a} = a_1, \ldots, a_p$ and $\bar{b} = b_1, \ldots, b_p$, for some $p \leq i$, and the graphs F^i, G^i are subgraphs of F and G respectively.

At each round, Spoiler picks one of the two structures and makes either a *first order* move or a *colouring* move. Assume that the position at round i is $\langle (F^i, \bar{a}), (G^i, \bar{b}) \rangle$, and assume, without loss of generality, that Spoiler picks the graph F^i. In a first order move, he picks a vertex $a_{p'}$ in F^i, and Duplicator replies by picking a vertex $b_{p'}$ in G^i. The position at the next round is $\langle (F^i, \bar{a}, a_{p'}), (G^i, \bar{b}, b_{p'}) \rangle$.

If Spoiler chooses to make a colouring move, he chooses two graphs F^i_1, F^i_2 such that $F^i = F^i_1 \mid F^i_2$, and Duplicator responds with two graphs G^i_1 and G^i_2 such that $G^i = G^i_1 \mid G^i_2$. Spoiler finally chooses which is the position at the next round, either $\langle (F^i_1, \bar{a}), (G^i_1, \bar{b}) \rangle$ or $\langle (F^i_2, \bar{a}), (G^i_2, \bar{b}) \rangle$.

The game ends after k rounds, or if one of the graphs in some position is empty or consists of a single edge. Let $h : X \rightharpoonup X$ be the partial map defined by $a_j \mapsto b_j$. Spoiler wins at the end of the game if one of the following conditions hold.

1. Exactly one of the graphs is empty.
2. One of the graphs is a single edge, with both its endpoints in the domain of h, and h is not an isomorphism between the graphs restricted to the non-isolated vertices.
3. The mapping h is not one-to-one.

The following Lemma is given in [DGG07] and follows the usual methods of Ehrenfeucht-Fraïssé games.

Lemma 1.5.4 ([DGG07])**.** *Duplicator has a winning strategy for the k round game on two graphs F and G, if and only if for any* GL *formula with $qr(\varphi) \leq k$, it holds that*

$$F \models \varphi \Leftrightarrow G \models \varphi.$$

To show that a property P is not expressible in GL we use the above Lemma in the following way.

Corollary 1.5.5. *A property P is not expressible in* GL *if and only if for every $k \in \mathbb{N}$, there exist graphs F_k and G_k such that $F_k \in P$ and $G_k \notin P$, and Duplicator has a winning strategy for the k round game on $\langle F_k, G_k \rangle$.*

We write $F \equiv_k^{\mathrm{GL}} G$ for two graphs F and G that cannot be distinguished by any GL formula φ with $\mathrm{qr}(\varphi) \leq k$. The following lemma is analogous to Lemma 1.2.3, for GL.

Lemma 1.5.6. *Let G_1, G_2, F_1 and F_2 be graphs where $\bar{a}$ is a tuple of elements of the graphs G_1 and G_2, and $\bar{b}$ is a tuple of elements of the graphs F_1 and F_2. If $(G_1, \bar{a}) \equiv_k^{\mathrm{GL}} (F_1, \bar{b})$ and $(G_2, \bar{a}) \equiv_k^{\mathrm{GL}} (F_2, \bar{b})$ then*

$$(G_1 \oplus_{\bar{a}} G_2, \bar{a}) \equiv_k^{\mathrm{GL}} (F_1 \oplus_{\bar{b}} F_2, \bar{b}).$$

Proof. We show by induction on the number of rounds i, that for any G_1, G_2, F_1 and F_2, and any tuple of vertices $\bar{v} \in G_1 \oplus_{\bar{a}} G_2$ and $\bar{w} \in F_1 \oplus_{\bar{b}} F_2$, if $(G_1, \bar{a}, \bar{v}) \equiv_i^{\mathrm{GL}} (G_2, \bar{a}, \bar{w})$ and $(F_1, \bar{b}, \bar{v}) \equiv_i^{\mathrm{GL}} (F_2, \bar{b}, \bar{w})$ then

$$(G_1 \oplus_{\bar{a}} G_2, \bar{a}, \bar{v}) \equiv_i^{\mathrm{GL}} (F_1 \oplus_{\bar{b}} F_2, \bar{b}, \bar{w}).$$

Let $G = G_1 \oplus_{\bar{a}} G_2$ and $F = F_1 \oplus_{\bar{b}} F_2$. For the base case let $i = 0$ and suppose for contradiction that Spoiler wins the 0-round GL Game on $\langle (G, \bar{a}, \bar{v}), (F, \bar{b}, \bar{w}) \rangle$. Let h be the function that maps $a_j \mapsto b_j$ and $v_{j'} \mapsto w_{j'}$, and let h_i, for $i = 1, 2$, be the restriction of h that maps the vertices from G_i to F_i. Then by definition, either exactly one of the graphs is empty, or one of the graphs is a single edge with endpoints in the domain of h and h is not an isomorphism, or finally h is not one-to-one. Suppose that exactly one of the graphs is empty, and let this be G. Then both G_1 and G_2 are empty and at least one of F_1 or F_2 is not empty, which is a contradiction. Suppose then that one of the graphs is a single edge with endpoints in the domain of h and h is not an isomorphism. Without loss of generality assume that this is the case for G. Then one of the graphs G_1 or G_2 is a single edge, and the other is empty. Let this be G_1 and G_2 respectively, and let the endpoints of the edge in G_1 be x_1, x_2, each an element of $\bar{a}$ or $\bar{v}$. Since h is not an isomorphism then either F is not a single edge or if it is a single edge, then x_1 and x_2 are not mapped to the endpoints of the edge. Both these conditions lead to a contradiction, since by assumption, h_1 and h_2 are isomorphisms from $(G_1, \bar{a}, \bar{v})$ to $(F_1, \bar{b}, \bar{w})$ and from $(G_2, \bar{a}, \bar{v})$ to $(F_2, \bar{b}, \bar{w})$ respectively. Finally, if h is not one-to-one then either h_1 or h_2 is not one-to-one, which is a contradiction.

Suppose then that the statement holds for all $i \leq K$, for some $K \in \mathbb{N}$, and let $i = K + 1$. We need to show that the position at the next round, is a winning position for Duplicator. Suppose

that Spoiler decides to make a first-order move, and does so by choosing a vertex x in one of the two graphs, say G. Then this vertex is in G_1 or G_2. If the vertex is one of the vertices in $\bar{a}$ then Duplicator responds with the appropriate vertex in $\bar{b}$, and $(G, \bar{a}, \bar{v}) \equiv_{i-1}^{\mathrm{GL}} (F, \bar{b}, \bar{w})$. If the vertex that Spoiler chose in G is not one of the vertices in $\bar{a}$, then the vertex is in G_1 or in G_2, but not both. Without loss of generality assume that it is in G_1. Duplicator has a winning strategy for the i-round GL Game on $\langle (G_1, \bar{a}, \bar{v}), (F_1, \bar{b}, \bar{w})\rangle$ and uses this strategy to respond with a vertex y in F_1. The position at the next round is $\langle (G, \bar{a}, \bar{v}, x), (F, \bar{b}, \bar{w}, y)\rangle$ and by the inductive hypothesis this is a winning position for Duplicator.

Suppose then that Spoiler decides to make a colouring move, and splits G into G^1 and G^2 such that $G = G^1 \mid G^2$. Then there are graphs G_1^1, G_2^1, G_1^2 and G_2^2, where $G^1 = G_1^1 \oplus_{\bar{a}} G_2^1$ and $G^2 = G_1^2 \oplus_{\bar{a}} G_2^2$, and are such that $G_1 = G_1^1 \mid G_1^2$ and $G_2 = G_2^1 \mid G_2^2$. By the assumption, Duplicator can find subgraphs $F_1^1, F_2^1, F_1^2, F_2^2$ of F, such that $F_1 = F_1^1 \mid F_1^2$ and $F_2 = F_2^1 \mid F_2^2$, and are such that Duplicator wins the $(i-1)$-round game on the corresponding subgraphs. Therefore, he replies with F^1 and F^2, where $F^1 = F_1^1 \oplus_{\bar{b}} F_2^1$ and $F^2 = F_1^2 \oplus_{\bar{b}} F_2^2$. By the inductive hypothesis, both $\langle (G^1, \bar{a}, \bar{v}), (F^1, \bar{b}, \bar{w})\rangle$ and $\langle (G^2, \bar{a}, \bar{v}), (F^2, \bar{b}, \bar{w})\rangle$ are winning positions for Duplicator. □

1.6 Graph Logic with Recursion

Graph Logic with recursion, which is denoted as GL_μ, is equipped with an additional operator which helps reasoning about fixed-point operators, and it was also introduced in [CGG02]. The syntax of GL is extended with an additional atomic formula R, for each recursion variable R, and the operator $\mu R.\varphi$, where φ is a GL_μ formula and, in order to ensure monotonicity, R appears positive in φ, or in other words under an even number of negations. For the following let $\mathcal{G}$ be the class of all graphs, let V be a countable set of vertices, and X be a set of vertex variables, as in Section 1.5. Similarly, for any graph G, let E_G be its set of edges.

Definition 1.6.1 (GL_μ Semantics)**.** *Let $\alpha : X \to V$ be an assignment of vertex names to variables and let $\hat{\alpha}$ be its extension to the domain $X \cup V$ using the identity function for mapping elements of V. Let ρ map recursion variables to subsets of $\mathcal{G}$. The semantics are defined inductively as follows:*

$$
\begin{aligned}
[\![\mathbf{0}]\!]_{\alpha;\rho} &= \{G \mid \ G \text{ is empty}\},\\
[\![\top]\!]_{\alpha;\rho} &= \mathcal{G},\\
[\![E_l(t_1,t_2)]\!]_{\alpha;\rho} &= \{G \mid \ E_G = \{e\},\ e \in E_l \text{ and } I(e) = (\hat{\alpha}(t_1), \hat{\alpha}(t_2))\},\\
[\![t_1 = t_2]\!]_{\alpha;\rho} &= \mathcal{G} \text{ if } \hat{\alpha}(t_1) = \hat{\alpha}(t_2),\ \emptyset \text{ otherwise},\\
[\![R]\!]_{\alpha;\rho} &= \rho(R),\\
[\![\neg\varphi]\!]_{\alpha;\rho} &= \mathcal{G} \setminus [\![\varphi]\!]_{\alpha;\rho},\\
[\![\varphi_1 \wedge \varphi_2]\!]_{\alpha;\rho} &= [\![\varphi_1]\!]_{\alpha;\rho} \cap [\![\varphi_2]\!]_{\alpha;\rho},\\
[\![\varphi_1 | \varphi_2]\!]_{\alpha;\rho} &= \{G \mid \ G = G_1|G_2 \text{ and } G_1 \in [\![\varphi_1]\!]_{\alpha;\rho}, G_2 \in [\![\varphi_2]\!]_{\alpha;\rho}\},\\
[\![\exists x\ \varphi]\!]_{\alpha;\rho} &= \bigcup_{v \in V} [\![\varphi]\!]_{\alpha[x\mapsto v];\rho},\\
[\![\mu R.\varphi]\!]_{\alpha;\rho} &= \bigcap\{S \subseteq \mathcal{G} \mid \ [\![\varphi]\!]_{\alpha;\rho[R\mapsto S]} \subseteq S\}.
\end{aligned}
$$

The semantics for the recursion operator are defined as usual, taking the complete lattice to be $\wp(\mathcal{G})$.

1.7 Contributions of this Thesis

In this thesis we investigate the expressive power of GL and GL_μ over various classes of graphs. For GL, a comparison to MSO is made and the expressibility in GL of some MSO-definable properties over trees is studied. Although we do not present inexpressibility results for GL over trees, we do give such a result for GL over forests. The expressive power of GL_μ over words is investigated and compared to a few graph grammars in the literature. Similarly, over trees the expressive power of GL_μ is compared to other logics, and it is shown to be more expressive than some.

In Chapter 2, the expressive power of GL is studied over graphs in general as well as over words and trees specifically. The chapter focuses on properties that are definable in GL over the classes of graphs mentioned. Over words, GL and MSO are known to be equi-expressive by results in [DGG07], and it is shown here that this extends to graphs that comprise a disjoint union of words.

For trees, it is shown that the class of trees containing binary trees with an even number of leaf vertices, and more generally any set of trees accepted by some deterministic bottom-up automaton with 2 states, is definable in GL. This establishes that this natural candidate for separating GL from MSO over the class of trees cannot be used for this purpose.

Comparison of GL is made to specific automata and logics that cannot define all regular tree languages, for the purpose of separating GL from MSO, by showing that the set of GL-definable languages is subsumed by one of the languages definable in the logics and automata we consider. In particular, for some of these logics it is established that GL is incomparable to them, as there exist properties definable in GL that are not in the logics considered.

In Chapter 3, the main result is presented, separating GL from MSO over forests, and a few consequences of this are investigated. In particular, it is shown that GL is strictly less expressive than MSO over graphs, and furthermore the case is so even when restricted over the class of forests. Marcinkowski showed in [Mar06] that GL+ and MSO+, two extensions of GL and MSO equipped with quantification over edge labels, do not have the same expressive power over graphs in general, and thus GL+ is strictly contained in MSO+. The main result we present in Chapter 3, strengthens this result.

Additionally, it is established that GL is not closed under FO-definable interpretations of width 1 and MSO transductions or more specifically, not even transductions specified by FO formulae.

In [BDL08], Brochenin et al. show that a syntactic fragment of Separation Logic called $\mathrm{SL}(-\!\!*)$, that includes the magic wand connective, but not the separation connective '$*$', is equi-expressive to Second-order Logic over memory heaps. Furthermore, they consider an additional syntactic fragment, $\mathrm{SL}(*)$, which includes the separation connective '$*$', but omits the magic

	GL	GL_μ
On Words	GL = MSO (†)	MSO $\subsetneq$ConjCF$\subseteq$ GL_μ (†)
On Sets of Words	GL = MSO	MSO $\subsetneq$ GL_μ
On Trees		MSO $\subsetneq$ GL_μ (†)
On Forests	GL $\subsetneq$ MSO	
On Graphs	GL $\subsetneq$ MS_2	

Table 1.1: Summary of known inclusions. The ones marked with '(†)' were first shown in [DGG07].

wand connective, and conjecture that this fragment is strictly less expressive than MSO over memory heaps. We show that this is indeed the case using our main result in Chapter 3. In particular, it is shown that the MSO property that is not expressible in GL, can be translated in a straightforward way to a property not definable in the fragment of Separation Logic SL($*$).

In Chapter 4, the expressive power of GL_μ is examined, over graphs in general, as well as words and trees in particular. It is shown that even when restricted to words, GL_μ can express properties that are PSPACE-complete. Its expressive power is compared with graph grammars, more specifically word grammars, such as Conjunctive Grammars, that are shown to be subsumed by GL_μ. Over trees, GL_μ is shown to be strictly more expressive than MSO and on graphs in general it is conjectured that GL_μ and MSO are incomparable, as it is established that there are GL_μ-definable properties that are not expressible in MSO, but also that there are properties such as 3-colourability of graphs, which are MSO-definable, and are conjectured to be inexpressible in GL_μ. Table 1.1 gives a summary of known inclusions regarding GL and GL_μ.

Chapter 2

Graph Logic

In this chapter we look into the expressive power of Graph Logic over graphs in general. Dawar et al. in [DGG07] have already proved some intriguing results about GL. One of them is that MSO and GL over words have the same expressive power. This is also the motivation behind big part of the research we conduct in this thesis, namely to see whether this result extends to other classes of graphs as well.

Useful GL Formulae: Throughout the next chapters, some properties of structures will be inspected, and it is worth collecting some of them and describe how one could express them in GL. Some simple examples are the formulae used to describe the degree of some vertex in a graph.

$$\begin{aligned}
\mathrm{In}_0(x) &= \forall y\ \neg(E(y,x) \mid \top),\\
\mathrm{Out}_0(x) &= \forall y\ \neg(E(x,y) \mid \top),\\
\mathrm{In}_{\geq k}(x) &= \exists x_1,\ldots,x_k\ \textstyle\bigwedge_{i\leq k}(E(x_i,x) \mid \top) \wedge \bigwedge_{i\neq j\leq k} x_i \neq x_j,\\
\mathrm{In}_{=k}(x) &= \mathrm{In}_{\geq k}(x) \wedge \neg \mathrm{In}_{\geq (k+1)}(x),\\
\deg_{=0}(x) &= \mathrm{In}_0(x) \wedge \mathrm{Out}_0(x),\\
\deg_{\geq k}(x) &= \textstyle\bigvee_{(i+j=k)} \big(\mathrm{In}_{\geq i}(x) \wedge \mathrm{Out}_{\geq j}(x)\big),\\
\deg_{=k}(x) &= \deg_{\geq k}(x) \wedge \neg \deg_{\geq (k+1)}(x).
\end{aligned}$$

Because in general the graphs that are dealt with have directed edges, the formulae $\mathrm{In}_{\geq k}(x)$ and $\mathrm{Out}_{\geq k}(x)$ express the number of incoming and outgoing edges of x respectively. The formula $\deg_{\geq k}(x)$ above, uses the former formulae to depict the number of overall edges adjacent to the vertex x. It is important to note that a formula of the above such as $\deg_{=k}(x)$, will usually be denoted as $\deg_k(x)$.

When we write $\forall x\ \varphi$, the latter is an abbreviation for the formula $\forall x\ (\mathrm{Here}(x) \rightarrow \varphi)$, where

$$\mathrm{Here}(x) = \mathrm{In}_{\geq 1}(x) \vee \mathrm{Out}_{\geq 1}(x).$$

As was stated in Section 1.5, the semantics for GL is defined in such a way where a graph is a set of edges with an incidence map, associating with each edge a pair of vertex names, namely

its endpoints. The set of vertex names is a countable set, and not all of them are adjacent to some edge. The formula above is used to ensure that the quantifier selects only vertices that are present in the graph, namely adjacent to some edge.

Another property that will recurrently appear throughout the next sections, is the one expressing the existence of a path between two nodes. The formula $\mathrm{Path}(x, y)$ will be used to express Graph Connectivity as well.

$$\begin{aligned} \mathrm{Path}_+(x,y) &= \big(\deg_1(x) \wedge \deg_1(y)\big) \wedge \forall z \ \big(x \neq z \wedge y \neq z \rightarrow \deg_2(z)\big), \\ \text{Connected} &= \forall x, y \ \big(\top \mid \mathrm{Path}_+(x,y)\big), \\ \mathrm{Path}(x,y) &= \mathrm{Path}_+(x,y) \wedge \text{Connected}. \end{aligned}$$

The formula $\mathrm{Path}_+(x, y)$ is satisfied by a graph comprising an undirected path from x to y together with a set of disjoint cycles. If such a path exists between any two vertices of the graph, then the graph is connected. The formula $\mathrm{Path}(x, y)$ makes the condition stricter so that a graph satisfies the latter if and only if it is exactly an undirected path from x to y. A formula expressing the existence of a *directed* path between two vertices x and y is defined in a similar manner to $\mathrm{Path}_+(x, y)$, and we skip the details. We proceed by showing how a path of even length can be expressed in GL, using the two formulae below.

$$\begin{aligned} \text{disjointEdges} &= \forall x \ \deg_1(x), \\ \mathrm{EvenPath}(x,y) &= \mathrm{Path}(x,y) \wedge \big(\deg_1(x) \wedge \text{disjointEdges} \mid \deg_1(y) \wedge \text{disjointEdges}\big). \end{aligned}$$

The above formula $\mathrm{EvenPath}(x, y)$ defines the class of even paths. Notice that, although the formula disjointEdges specifies that all vertices have degree 1, this is so for all the vertices present in the subgraph satisfying it. Therefore, it is stated explicitly that, say x in the first subgraph, has degree 1, to ensure that x in that subgraph is present, or in other words an edge adjacent to it is present. A formula for the odd paths can be defined similarly, with

$$\mathrm{OddPath}(x,y) = \mathrm{Path}(x,y) \wedge \big(\deg_1(x) \wedge \deg_1(y) \wedge \text{disjointEdges} \mid \text{disjointEdges}\big),$$

by specifying that both endpoints are present in the same subgraph of disjoint edges. All trees and words are represented as acyclic graphs with directed edges where each node has at most one incoming edge. The root has no incoming edges. The following formulae define basic properties of vertices in a tree or word.

$$\begin{aligned} \mathrm{root}(x) &= \forall y \ \neg(E(y,x) \mid \top) \\ \mathrm{leaf}(x) &= \forall y \ \neg(E(x,y) \mid \top) \\ \text{one-child}(x) &= \exists! y \ (E(x,y) \mid \top) \\ \mathrm{fork}(x) &= \exists y, z \ (y \neq z \wedge (E(x,y) \mid E(x,z) \mid \top)) \end{aligned}$$

When dealing with graphs that are sets of disjoint words, or simply words, we will use the formulae $\mathrm{first}(x)$ and $\mathrm{last}(x)$ instead of $\mathrm{root}(x)$ and $\mathrm{leaf}(x)$, where the former are defined identically to the latter respectively.

2.1 On Graphs

In [DGG07] the authors have proved that GL can express complete problems at every level of the Polynomial Hierarchy. This is done by encoding instances of the Quantified Boolean Formula problem (QBF) into graphs and then defining a GL formula that is true if and only if the Quantified Boolean Formula at hand is satisfiable.

Many useful properties can be expressed in GL. As explained in [DGG07], a graph G satisfies the formula φ_{cyc} below exactly when the graph G is a disjoint union of cycles.

$$\begin{aligned} \varphi_{cyc} &= \forall x\ (\deg_{=2}(x)), \\ \varphi_{even} &= \varphi_{cyc} \wedge (\text{disjointEdges} \mid \text{disjointEdges}). \end{aligned}$$

Similarly, a graph G satisfies the formula φ_{even}, exactly when G is a set of disjoint cycles all of which are of even length, as it is only then that a cycle can be split into two sets of disjoint edges. Hence using the formula $\varphi_{two-col}$ one can express that a graph is 2-colourable.

$$\begin{aligned} \varphi_{odd} &= \varphi_{cyc} \wedge \neg(\text{disjointEdges} \mid \text{disjointEdges}), \\ \varphi_{two-col} &= \neg(\varphi_{odd} \mid \top). \end{aligned}$$

This is so since $\varphi_{two-col}$ states that we cannot find a subgraph of G that satisfies φ_{odd}. The latter is satisfied by a graph comprising a set of disjoint cycles, at least one of which being of odd length.

The class of graphs that are 2-colourable is in P and a natural question is whether any NP-complete colouring problem is definable in GL. We show with the help of edge-disjoint subgraphs and 2-colourability, that one can express 4-colourability of graphs in GL. Unfortunately, the method used to show this cannot be easily extended to express 3-colourability of graphs, and furthermore it is conjectured here, as well as in [DGG07], that the class of 3-colourable graphs is not definable in GL. The next lemma helps us define the formula to express 4-colourability or more generally 2^k-colourability for any $k \in \mathbb{N}$.

Lemma 2.1.1. *A graph G is $(m \cdot n)$-colourable if and only if it can be split into two edge-disjoint subgraphs, one of which is m-colourable and the other is n-colourable.*

Proof. Let G be a $(m \cdot n)$-colourable graph. Then there is a colouring function $c : V(G) \to C$ where $|C| = m \cdot n$ and without loss of generality we can assume that $C = \{(c_1, c_2) |\ c_1 \in C_1,\ c_2 \in C_2\}$, for sets C_1 and C_2 of cardinality m and n respectively. In other words the function c maps vertices from $V(G)$ to the set $C_1 \times C_2$. Consider the two projections c_1 and c_2 of this colouring function, where $c_i : V(G) \to C_i$, for $i \in \{1, 2\}$. Let G_1 be the graph that contains exactly those edges (x, y) such that $c_1(x) \neq c_1(y)$. Let also G_2 be the graph that contains the edges between vertices x', y' such that $c_1(x') = c_1(y')$ but $c_2(x') \neq c_2(y')$. The graph G_1 is m-colourable and the graph G_2 is n-colourable. Furthermore, no edge of G belongs to both G_1 and G_2, and every edge of G belongs to one of the graphs since for any two vertices x and y connected with an edge in G either $c_1(x) \neq c_1(y)$ or $c_2(x) \neq c_2(y)$.

For the other direction assume that the graph G can be split into two edge-disjoint graphs G_1 and G_2 such that G_1 is m-colourable and G_2 is n-colourable. Then there are colouring functions c_1 and c_2 such that $c_i : V(G_i) \to C_i$, for $i \in \{1, 2\}$, where C_1 and C_2 are of cardinality m and n respectively. Then we can define a colouring c for G such that $c : V(G) \to C_1 \times C_2$ and where $c(x) = (c_1(x), c_2(x))$ for any vertex $x \in V(G)$. This colouring function c is a witness to the graph G being $(m \cdot n)$-colourable. □

Corollary 2.1.2. *For any $m, n \in \mathbb{N}$ such that m-colourability and n-colourability are definable in* GL*, $(m \cdot n)$-colourability is also definable in* GL*.*

Proof. If the GL formula ψ_m defines m-colourability and ψ_n defines n-colourability, then the GL formula $\psi = (\psi_m \mid \psi_n)$ defines $(m \cdot n)$-colourability, by Lemma 2.1.1. □

Since in the example above we have shown that 2-colourability is definable in GL we get the following:

Corollary 2.1.3. *For any $k \in \mathbb{N}$ there exists a* GL *formula φ_k such that for any graph G it is the case that $G \models \varphi_k$ if and only if G is 2^k-colourable.*

According to this Corollary there exists a GL formula φ such that for any graph G, $G \models \varphi$ if and only if G is 4-colourable. This constitutes an example of a colouring problem definable in GL, that lies in NP, and in fact is NP-complete, unlike 2-colourability that is in PTIME. The lowest k for which k-colourability is NP-complete, is $k = 3$, for which it is thought to not be definable in GL. Furthermore, for any $k > 1$, 2^k-colourability is an NP-complete problem as well.

2.2 On Words

As stated above, Dawar et al. showed in [DGG07] that GL and MSO are equi-expressive when the two logics are restricted to the class of finite words. This result is the main motivation behind looking into what is the case when restricted to other, more general classes of graphs. A few more general classes of graphs will be investigated below, but earlier a few things about GL on words and related structures should be examined further.

Theorem 2.2.1 (Dawar et. al.)**.** *A word language is* GL *definable if and only if it is regular.*

The result of GL and MSO being equi-expressive on words relies on a direct translation from regular expressions to GL formulae. An important subclass of regular word languages is the class of star-free languages. This class is exactly the class of languages definable in First Order Logic in the presence of order. A language is star-free if the regular expression that describes the language does not use the Kleene star, but can use the boolean operations, including complementation. Similarly, on a word structure of the usual signature together with a predicate $<$ for linear order, one can define in First Order Logic over this signature exactly the star-free languages, as explained in [RS71] and [Tho96].

The signature of the structures in GL though, does not include an order relation, and therefore restricting GL to the first order fragment, namely the fragment of GL that lacks the separation connective, is not enough for defining all star-free regular languages. For example, the word language $L = \{w \in \Sigma^* \mid \text{there is an } a \text{ preceding a } b\}$ with $\Sigma = \{a, b, c\}$, is star-free but is not definable in first order logic without order. The star-free expression for L is the following:

$$L = \overline{\emptyset}a\overline{\emptyset}b\overline{\emptyset}.$$

It can be shown that this language is not definable in FO without order. In particular let $L_1 = \{w \mid w = c^*ac^*bc^*\}$ and $L_2 = \{w \mid w = c^*bc^*ac^*\}$. Then it holds that $L_1 \subseteq L$ and $L_2 \cap L = \emptyset$. It can be proved that in a first-order Ehrenfeucht-Fraïssé game, for any k there exist words $w_1 \in L_1, w_2 \in L_2$ such that Duplicator wins the k-round game on these two words.

Since it is known that GL is closed under all boolean operations and given the direct translation from regular expressions to GL formulae as it appears in [DGG07], one can ignore the case of the Kleene star operator in the inductively defined translation in [DGG07], include a case for complementation, and get a translation from star-free regular expressions to GL formulae. It should be noted that if only the rest of the cases of the translation in [DGG07] are used to translate a star-free regular expression into a GL formula, it holds that in the resulting formula, at each application of the splitting operator '|' of GL, a word is split in at most one place, so that the two subgraphs after the split are each a connected component. This operation is hence similar to the Chop operator defined in [Lan02], which essentially works by splitting a word into two subwords. Therefore, star-free regular languages can be defined in the syntactic fragment of GL in which each time a word is split into two connected subwords and not more.

One more class of structures most closely related to the class of finite words, is the one of structures that are disjoint unions of words. As we will see, MSO without the presence of order can only count the number of different words up to some threshold, and from this it follows that GL has the same expressive power as MSO over sets of words. Informally, any MSO sentence is equivalent to one that expresses which MSO-k-types of words appear in the set and in what number each, up to some threshold value K. The same can be done in GL using First Order quantification and the fact that GL and MSO are equi-expressive on words.

The reason why such structures are of interest is the following. In the translation given in [DGG07] from MSO to GL over words, the case of the Kleene star operator is dealt with by splitting up a word in many subwords, and it is interesting to know that GL continues to be equi-expressive to MSO over the structures comprising subwords, which are disjoint sets of words.

Furthermore, in [Mar06], where Marcinkowski shows that GL+ is strictly less expressive than MSO+, the separating example is an appropriately constructed structure comprising a set of disjoint words, with labels on the edges of these words. We remind the reader that GL+ and MSO+ allow for first-order quantification over edge labels, and the structures are defined over countably many labels. Marcinkowski in the paper explains that the same method would not

work over sets of disjoint words of a finite alphabet for GL and MSO, and we show here that over structures composed of disjoint words, GL and MSO are equi-expressive.

We proceed by giving a lemma required for establishing the above and remind the reader of the following. The disjoint union of two structures in MSO results in a structure containing the disjoint union of the universes of the two original structures, and hence in the case of graphs in particular, no two edges from two graphs G_1, G_2 are adjacent in the graph resulting from the disjoint union of G_1 and G_2. It should be noted that Lemma 2.2.2 below also follows from results in [Com89].

Lemma 2.2.2. *For any k there exists $s \in \mathbb{N}$ such that:*

1. *For any structures $\mathfrak{B}_1, \ldots, \mathfrak{B}_s, \mathfrak{C}_1, \ldots, \mathfrak{C}_{s+1}$ of some signature σ, if $\mathfrak{B}_i \equiv_k^{\mathrm{MSO}} \mathfrak{C}_j$ for all $1 \leq i \leq s$ and $1 \leq j \leq s+1$, then $\mathfrak{B}_1 \uplus \ldots \uplus \mathfrak{B}_s \equiv_k^{\mathrm{MSO}} \mathfrak{C}_1 \uplus \ldots \uplus \mathfrak{C}_{s+1}$.*

2. *For any structures $\mathfrak{A}, \mathfrak{B}$ and $\mathfrak{C}$, if $(s \times \mathfrak{A}) \uplus \mathfrak{B} \equiv_k^{\mathrm{MSO}} \mathfrak{C}$ then for any m, $(m \times \mathfrak{A}) \uplus \mathfrak{C} \equiv_k^{\mathrm{MSO}} \mathfrak{C}$.*

Proof. For (1), let $T = \{\tau_1, \ldots, \tau_n\}$ be an enumeration of all MSO-k-types realized in structures of some signature σ. For each $1 \leq t \leq n$, let $\mathfrak{A}_t$ be the smallest structure of type τ_t, and let $m = \max\{|\mathfrak{A}_t| \mid 1 \leq t \leq n\}$. Furthermore, let $s = 2^{k \cdot m}$. For what follows let i range over the set $\{0, \ldots, s\}$ and j, j' over the set $\{0, \ldots, s+1\}$. First notice that since for all i and j it holds that $\mathfrak{B}_i \equiv_k^{\mathrm{MSO}} \mathfrak{C}_j$, it is also true for any j, j' that $\mathfrak{C}_j \equiv_k^{\mathrm{MSO}} \mathfrak{C}_{j'}$. Similarly for the structures $\mathfrak{B}_i$. Note that there exists $\mathfrak{A}_t$, being the structure of smallest size of type τ_t as defined above, such that for all structures $\mathfrak{B}_i$ and $\mathfrak{C}_j$, it is the case that $\mathfrak{B}_i \equiv_k^{\mathrm{MSO}} \mathfrak{A}_t \equiv_k^{\mathrm{MSO}} \mathfrak{C}_j$. Without loss of generality let this structure be $\mathfrak{A}_1$.

We want to prove that Duplicator has a winning strategy for the k-round game on the pair of structures $(\mathfrak{B}_1 \uplus \ldots \uplus \mathfrak{B}_s, \mathfrak{C}_1 \uplus \ldots \uplus \mathfrak{C}_{s+1})$. Note that, by the Feferman-Vaught Theorem (Theorem 1.2.3), since $\mathfrak{B}_i \equiv_k^{\mathrm{MSO}} \mathfrak{A}_1$ and $\mathfrak{C}_j \equiv_k^{\mathrm{MSO}} \mathfrak{A}_1$ for all i and all j, it is enough to establish that Duplicator has a winning strategy for the k-round game on $(s \times \mathfrak{A}_1, (s+1) \times \mathfrak{A}_1)$.

We proceed to show that Duplicator has a winning strategy for the k-round MSO EF game on $(s \times \mathfrak{A}_1, (s+1) \times \mathfrak{A}_1)$. We prove by induction on the number of remaining rounds ℓ that there is a colouring c that colours at least $2^{\ell \cdot m}$ copies of $\mathfrak{A}_1$ in both structures, with no distinguished elements from the previous First Order rounds, and for all other colourings c', an equal number of copies in each structure is coloured with c'. In particular, if the position at some round ℓ is $\langle (s \times \mathfrak{A}_1, \bar{a}, \bar{X}), ((s+1) \times \mathfrak{A}_1, \bar{b}, \bar{Y}) \rangle$ then there are at least $2^{\ell \cdot m}$ copies of $\mathfrak{A}_1$ in $s \times \mathfrak{A}_1$ and $(s+1) \times \mathfrak{A}_1$, such that for each such copy $\mathfrak{A}$ in $s \times \mathfrak{A}_1$ and $\mathfrak{A}'$ in $(s+1) \times \mathfrak{A}_1$, none of the elements of $\bar{a}$ and $\bar{b}$ are in $\mathfrak{A}$ and $\mathfrak{A}'$ respectively, and $(\mathfrak{A}, \bar{X}) \cong (\mathfrak{A}', \bar{Y})$ and the following condition is satisfied. For the substructures $\mathfrak{D}$ of $s \times \mathfrak{A}_1$ and $\mathfrak{E}$ of $(s+1) \times \mathfrak{A}_1$, obtained when disregarding the identically coloured copies defined above, it holds that $(\mathfrak{D}, \bar{a}, \bar{X}) \cong (\mathfrak{E}, \bar{b}, \bar{Y})$.

For $\ell = k$ there are at least $2^{k \cdot m}$ copies of $\mathfrak{A}_1$ in each structure by definition, with no distinguished elements. Assume now that the claim is true for $\ell' > 0$ rounds remaining. We want to prove that it is also the case for $\ell' - 1$ considering the two possible moves Spoiler can make.

First Order move: Suppose Spoiler selects an element x that lies in a copy of $\mathfrak{A}_1$ in one of the two structures, say $s \times \mathfrak{A}_1$. By the induction hypothesis there are $2^{\ell' \cdot m}$ identically coloured copies of $\mathfrak{A}_1$ with no distinguished elements. If the element x lies inside one of these copies, then Duplicator replies with an element x' inside a copy of the corresponding $2^{\ell' \cdot m}$ copies that have been coloured identically in the other structure. Otherwise Duplicator can find a copy of $\mathfrak{A}_1$ in $(s+1) \times \mathfrak{A}_1$ and an element x' in that copy since the rest of the colourings appear in equal numbers. In the next round the claim holds for $2^{(\ell'-1) \cdot m}$. The case is similar when Spoiler chooses an element of $(s+1) \times \mathfrak{A}_1$.

Colouring move: Spoiler chooses one of the two structures, say the structure $s \times \mathfrak{A}_1$, and chooses also a set X colouring the elements of the copies. Each copy of $\mathfrak{A}_1$ has at most m elements and by induction hypothesis at least $2^{\ell' \cdot m}$ copies have been coloured identically as a result of the previous rounds. There are at most 2^m possible colourings of each copy of $\mathfrak{A}_1$ using X and so at least $2^{\ell' \cdot m}/2^m = 2^{(\ell'-1) \cdot m}$ copies of $\mathfrak{A}_1$ will be coloured identically after applying X. Duplicator then colours s of the copies in $(s+1) \times \mathfrak{A}_1$ as they are coloured in $s \times \mathfrak{A}_1$ and colours the last copy using the colouring that appears more than $2^{(\ell'-1) \cdot m}$ times in $s \times \mathfrak{A}_1$. Duplicator acts similarly in the case where Spoiler colours $(s+1) \times \mathfrak{A}_1$.

Consider now the statement (2). By (1) we know that for any m, $(m \times \mathfrak{A}) \uplus (s \times \mathfrak{A}) \equiv_k^{\mathrm{MSO}} s \times \mathfrak{A}$, and therefore by a composition argument $(m \times \mathfrak{A}) \uplus (s \times \mathfrak{A}) \uplus \mathfrak{B} \equiv_k^{\mathrm{MSO}} (s \times \mathfrak{A}) \uplus \mathfrak{B}$. Also, $(m \times \mathfrak{A}) \uplus (s \times \mathfrak{A}) \uplus \mathfrak{B} \equiv_k^{\mathrm{MSO}} (m \times \mathfrak{A}) \uplus \mathfrak{C}$. This completes the proof since by assumption $(s \times \mathfrak{A}) \uplus \mathfrak{B} \equiv_k^{\mathrm{MSO}} \mathfrak{C}$. □

A similar lemma to Lemma 2.2.2 can be shown for graphs and MS_2.

Lemma 2.2.3. *For any* MSO *sentence* ϕ *there exists a* MSO *formula* $\chi(x, y)$ *such that for any word* W *and its leftmost vertex* a *and rightmost vertex* b*, it is the case that* $W \models \phi$ *if and only if* $W \models \chi(a, b)$*. Furthermore, if* $\mathfrak{A}$ *is any structure then* $\mathfrak{A} \uplus W \models \chi(a, b)$ *if and only if* $W \models \phi$*.*

Proof. Let ϕ be some MSO formula. We define $\chi(x, y)$ to be the MSO formula with free variables x and y, presented below. We use the formulae $\mathrm{In}_k(x)$ and $\mathrm{Out}_k(x)$ defined above, since they are essentially first-order formulae.

$$\begin{aligned}
\chi(x,y) &= \mathrm{In}_0(x) \wedge \mathrm{Out}_0(y) \wedge \mathrm{ExistsPath}(x,y) \wedge \\
&\qquad \exists X \left(\begin{array}{ll} & \forall z \ (X(z) \leftrightarrow \mathrm{ExistsPath}(x,z)) \\ \wedge & X(x) \wedge X(y) \\ \wedge & \phi^X \end{array} \right), \\
\mathrm{ExistsPath}(x,z) &= \forall Y \ \Big(Y(x) \wedge \big(\forall w, u \ (Y(u) \wedge E(u,w) \rightarrow Y(w)) \big) \rightarrow Y(z) \Big).
\end{aligned}$$

The notation ϕ^X denotes ϕ relativized to the vertices of X. According to the above definitions, if $W \models \phi$ then there exist vertices x, y at the endpoints of W, and a set of vertices X containing exactly the vertices between x, y, such that relativizing ϕ to the set of vertices X holds. Therefore if a and b are the endpoints of W, it is the case that W satisfies $\chi(a, b)$.

Likewise, if $W \models \chi(a,b)$ for W a word with endpoints a and b, then ϕ is satisfied when restricted to the vertices of W and therefore $W \models \phi$. The argument for the second case, where structures of the form $\mathfrak{A} \uplus W$ are considered, follows similar lines. □

Since the MSO formula ϕ can be translated to an equivalent over words formula ϕ' in GL, one can also translate $\chi(x,y)$ into an equivalent GL formula $\chi'(x,y)$. The formula $\chi'(x,y) = \mathrm{In}_0(x) \wedge \mathrm{Out}_0(y) \wedge (\top \mid \mathrm{Path}(x,y) \wedge \phi')$, where $\mathrm{Path}(x,y)$ is satisfied only by the subgraph comprising the word from x to y, is a translation of χ into GL.

Theorem 2.2.4. *For every* MSO *formula* φ *there exists a* GL *formula* ψ *such that for any structure* $\mathfrak{A}$ *that is a set of strings, it is the case that* $\mathfrak{A} \models_{\mathrm{MSO}} \varphi$ *if and only if* $\mathfrak{A} \models_{\mathrm{GL}} \psi$.

Proof. Fix an MSO formula φ of quantifier rank k and let s be given by Lemma 2.2.2 for k. The formula φ is equivalent to a disjunction of MSO-k-types. Let T be the finite set of MSO-k-types of structures that are sets of words, and let $T_\varphi \subseteq T$ denote the set of types in the latter disjunction that is equivalent to φ.

Similarly, define T_k to be the finite set of MSO-k-types of words and let $F^k = \{f \mid f : T_k \to \{0,\dots,s\}\}$ be the finite set of functions from T_k to $\{0,\dots,s\}$. With each structure $\mathfrak{A}$ that is a set of words, we associate a function $f^{\mathfrak{A}} \in F^k$ which is such that for each MSO-k-type $\tau \in T_k$, $f^{\mathfrak{A}}(\tau) = z$ if there are exactly z words in $\mathfrak{A}$ of MSO-k-type τ and $z \leq s$, and $f^{\mathfrak{A}}(\tau) = s$ if there are more than s words in $\mathfrak{A}$ of MSO-k-type τ. Notice that for each such structure $\mathfrak{A}$, $f^{\mathfrak{A}}$ is well-defined and that any two structures $\mathfrak{A}_1, \mathfrak{A}_2$ with which the same $f \in F^k$ is associated, can only disagree on the number of types that appear at least s times. But then, according to Lemma 2.2.2, two such structures are $\equiv_k^{\mathrm{MSO}}$-equivalent.

Then, F^k can be partitioned into sets $F_{\tau_1}, \dots, F_{\tau_M}$, for $M = |T|$, such that for each $\tau_j \in T$ and any structure $\mathfrak{A}$ of type τ_j there is $f \in F_{\tau_j}$ associated with $\mathfrak{A}$. Furthermore, any structure with which some function $f \in F_{\tau_j}$ is associated, for some $0 \leq j \leq M$, is of type τ_j. Then let $F = \bigcup_{\tau_j \in T_\varphi} F_{\tau_j}$.

With each MSO rank-k word type τ_i, we associate the formula $\Psi_i(x,y)$, given by Lemma 2.2.3, such that for any word w, w is of type τ_i if and only if a, b are the endpoints of w and $w \models_{\mathrm{MSO}} \Psi_i(a,b)$. For $n \in \mathbb{N}$, we define the formula $\Phi_i^{\geq n}$ to be the formula

$$\Phi_i^{\geq n} = \exists x_1, y_1, \dots, x_n, y_n \bigwedge_{1 \leq \ell \neq \ell' \leq n} \Big(x_\ell \neq x_{\ell'} \wedge y_\ell \neq y_{\ell'} \wedge x_\ell \neq y_{\ell'} \Big) \wedge \bigwedge_{1 \leq \ell \leq n} \Psi_i(x_\ell, y_\ell),$$

which expresses that there are at least n distinct pairs of vertices that are endpoints of words of type τ_i. Similarly the formula

$$\Phi_i^{=n} = \Phi_i^{\geq n} \wedge \neg \Phi_i^{\geq n+1},$$

expresses that there are exactly n distinct pairs of vertices that are endpoints of words of type

τ_i. According to this, the formula φ is equivalent to the following formula Φ.

$$\Phi = \bigvee_{f \in F} \left(\begin{array}{lll} & \bigwedge_{i:f(\tau_i)=s_i, 0<s_i<s} & \Phi_i^{=s_i} \\ \wedge & \bigwedge_{j:f(\tau_j)=s} & \Phi_j^{\geq s} \\ \wedge & \bigwedge_{h:f(\tau_h)=0} & \forall x, y \ \neg \Psi_h(x,y) \end{array} \right).$$

Therefore, since the formulae $\Psi_j(x,y)$ are expressible in GL, so is the formula Φ. □

2.3 On Trees

GL is equi-expressive to MSO on words and a natural question is whether this equivalence generalizes to other classes of graphs. One of the simplest extensions to the class of words is the class of trees and inspecting the expressive power of GL on trees is most reasonable.

A prominent weakness of GL is for example the fact that if a tree T satisfies a formula ψ of the form $\psi = (\psi_1 \mid \psi_2)$, then neither of the ψ_i for $i \in \{1,2\}$ can express that a vertex is a leaf vertex of the original tree. This is because some of the information may have been lost when splitting the tree, into two disjoint forests.

A natural property, therefore, that one should inspect whether it is GL definable is the property of evenness of the number of leaves of a tree, and in general checking whether the number of leaves of ranked trees is equal to 0 (mod k), for some $k \in \mathbb{N}$. Such properties are MSO definable. Although it still remains an open question whether, for example, one can define in GL the class of binary trees with 0 (mod 4) number of leaves, a positive answer is given for evenness on binary trees.

The latter is a corollary of a more general result that states that any language of binary trees accepted by some product of bottom-up deterministic 2-state tree automata, is GL definable. It should be noted that such a property shows that GL is in some respect quite expressive, and that this property distinguishes GL from many of the logics in the literature. It will also be shown that this is not enough for defining any regular binary tree language, since, for any fixed alphabet, it is a finite number of languages that are accepted by products of deterministic bottom-up 2-state tree automata.

Theorem 2.3.1. *Any language of binary trees accepted by a product of bottom-up deterministic 2-state tree automata, is* GL *definable.*

Proof. We only need to show that any language of binary trees accepted by a deterministic automaton with 2 states is definable in GL, since conjunction of two GL formulae can be used to define the product of two such automata. Furthermore, although we allow binary trees with unary branches, we assume here that the root of any tree considered has two children. If this is not the case for some tree, we can split the tree into the unary branch from the root to the first fork x, and the tree rooted at that fork. The tree rooted at x will be dealt with using the GL

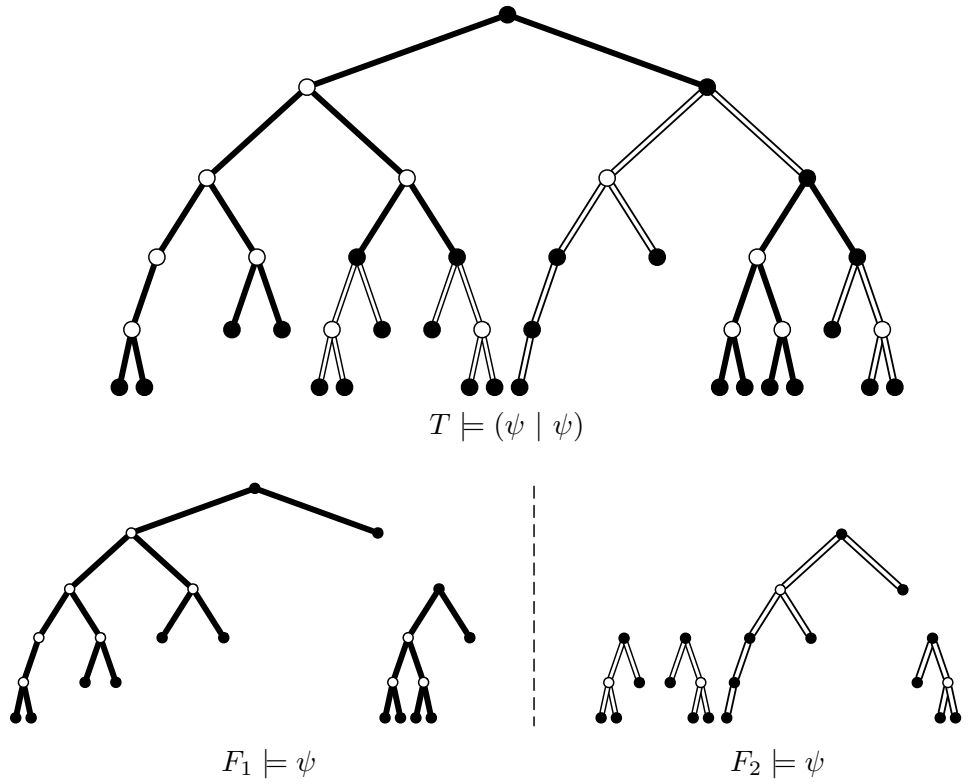


Figure 2.1: The black-filled nodes (•) are assigned the state q_1 and the white-filled nodes (∘) are assigned the state q_2 by the automaton.

formulae presented below, whereas the unary branch will be dealt with as a special case, using the fact that GL and MSO are equi-expressive over words.

Suppose that $A = (Q, \Sigma, q_0, \delta_1, \delta_2, Q_f)$ is a deterministic bottom-up tree automaton as defined in Section 1.3, where $Q = \{q_1, q_2\}$, the initial state q_0 is defined to be equal to q_1, and δ_1,δ_2 are the two transition functions of type $\delta_1 : Q \times \Sigma \to Q$ and $\delta_2 : Q \times \Sigma \times Q \times \Sigma \to Q$, respectively. Let $\mathcal{T}$ be the class of binary trees T' each with a root that is a fork, where A assigns the state q_1 to all the leaves and the root of T', and assigns the state q_2 for all forks other than the root of T'. Suppose there exists a GL formula ψ such that for any forest F, $F \models \psi$ if and only if for every tree T' in F, $T' \in \mathcal{T}$. Before we present such a GL formula ψ explicitly, we show how it can be used to define the class of trees accepted by A.

In particular, we show that for any binary tree T, $T \models (\psi \mid \psi)$ if and only if A assigns the state q_1 to the root of T. For the *only if* direction fix a tree T and suppose that $T \models (\psi \mid \psi)$. Then, the tree T can be split into two forests F_1 and F_2, where each one satisfies ψ. Therefore, for any tree T'' in F_1 or F_2, A assigns the state q_1 to the leaves and the root of T''. Notice, that since every tree in the two forests has a root with two children, if a tree T_1 in F_1 is connected to a tree T_2 in F_2 within the original tree T, then the root of one is the same vertex as a leaf of the other. In other words the tree T is split into F_1 and F_2 in such a way that every vertex

that is shared between both forests, is a leaf of a tree in one forest and root of another tree in the other forest. Therefore, a run of A can be constructed that assigns the state q_1 to the leaves and the root of T, and in addition assigns the state q_1 to all the vertices that appear in both forests.

For the *if* direction, suppose that A assigns the state q_1 to the root of T. For each fork x in T such that A assigns the state q_1 to x, we define Y_x to be the set of descendants closest to x that are either leaves or forks to which A assigns the state q_1. The subtree T_x is defined to be the subtree rooted at x, with leaves the vertices in Y_x. We define F_1 and F_2 to be two forests comprising of trees T_x for some fork x satisfying the assumption given above, and furthermore we define the two forests to be such that no two trees T_x and $T_{x'}$ that are connected in T, are both in the same forest. By definition, every fork other than the root in a tree T_x is assigned the state q_2 by A, and hence $F_i \models \psi$, for $i = 1, 2$. A simple example of a tree T with an odd number of leaves, and the run of an automaton accepting such trees, is shown in Figure 2.1. We skip the definition of such an automaton.

We proceed by defining explicitly the GL formula ψ. A *unary branch* from some vertex x to a vertex y, is a path in the tree from x to one of its descendants y, where all vertices in that path apart from y are unary nodes. On a unary branch, a tree automaton works as a word automaton and in particular uses only the transition function δ_1 defined above. On words, MSO and GL are equi-expressive, so let the GL formula φ_{q_i,q_j}, for $q_i, q_j \in \{q_1, q_2\}$, be the formula defining the set of words on which the automaton, assigning the state q_i to the first vertex, reaches the state q_j at the last one.

Two vertices a and b in a tree T, are endpoints of a unary branch if and only if $T \models \text{unaryBranch}(a, b)$, where:

$$\begin{aligned}\text{unaryBranch}(x, y) \;=\; & \text{one-child}(x) \wedge (\text{leaf}(y) \vee \text{fork}(y)) \wedge (\text{Path}(x, y) \mid \top) \wedge \\ & \wedge \forall z\ (z \neq y \wedge (\text{Path}(x, z) \mid \text{Path}(z, y) \mid \top) \rightarrow \text{one-child}(z)).\end{aligned}$$

Notice that in the definition above, the formula $(\text{Path}(x, y) \mid \top)$ is used, since, whenever $T \models \text{unaryBranch}(a, b)$ for some tree T with vertices a, b, we don't require T to be a unary branch, but instead we require the vertices a and b in T to be the endpoints of a unary branch in T. The following formulae assist with the definition of the formula ψ and are defined for $i, j \in \{1, 2\}$.

$$\begin{aligned}\text{unary-}q_1q_j(x) &= \exists y\ (\text{leaf}(y) \wedge \text{unaryBranch}(x, y) \wedge (\text{Path}(x, y) \wedge \varphi_{q_1,q_j} \mid \top)), \\ \text{unary-}q_2q_j(x) &= \exists y\ (\text{fork}(y) \wedge \text{unaryBranch}(x, y) \wedge (\text{Path}(x, y) \wedge \varphi_{q_2,q_j} \mid \top)), \\ \text{unary-}q_i(x) &= \text{one-child}(x) \wedge (\text{unary-}q_2q_i(x) \vee \text{unary-}q_1q_i(x)), \\ \text{state-}q_1(x) &= \text{leaf}(x) \vee \text{root}(x) \vee \text{unary-}q_1(x), \\ \text{state-}q_2(x) &= (\text{fork}(x) \wedge \neg\text{root}(x)) \vee \text{unary-}q_2(x).\end{aligned}$$

The formula unary-$q_1q_j(x)$ expresses that x is the top vertex of a unary branch from a vertex y, a descendant of x, where y is a leaf vertex and the unary branch from y to x belongs to the set of words on which the automaton A starting with state q_1 finishes with the state q_j. The

case is similar for the formula unary-$q_2q_j(x)$, which expresses that there exists a fork y which is a descendant of x and the path from y to x is a unary branch that belongs to the set of words on which A, starting with state q_2, ends with the state q_j. The formula unary-$q_j(x)$ is simply the disjunction of the two formulae above. The formula state-$q_1(x)$ holds at any vertex x that is either a leaf, a root or the top vertex of a unary branch where A reaches q_1 according to the assumptions above. Similarly, the formula state-$q_2(x)$ holds at any vertex x that is a non-root fork or the top vertex of a unary branch, at which A reaches q_2 under the assumptions above.

Remember that $\mathcal{T}$ is the class of trees where A assigns the state q_1 to all leaves and the root, and assigns the state q_2 to all forks that are not roots. We show below, that for any vertex a of any tree $T \in \mathcal{T}$, A assigns the state q_i to a if and only if $T \models$ state-$q_i(a)$. The definition of the formula ψ follows below.

$$\begin{aligned}
\varphi_{q_1}(x,y,z) &= \bigvee_{\delta_2(q_i,\sigma,q_j,\sigma')=q_1} (\text{state-}q_i(y) \wedge (E_\sigma(x,y) \mid \top) \wedge \text{state-}q_j(z) \wedge (E_{\sigma'}(x,z) \mid \top)),\\
\text{fork-roots} &= \forall x\ (\text{root}(x) \rightarrow \text{fork}(x)),\\
\psi &= \mathbf{0} \vee \Big(\text{fork-roots} \wedge \forall x\ \big((\exists y,z\ (y \neq z \wedge \varphi_{q_1}(x,y,z))) \leftrightarrow \text{root}(x)\big)\Big).
\end{aligned}$$

Notice that for any forest F, $F \models \psi$ if and only if for every tree T in that forest, $T \models \psi$. This is because ψ expresses that a combination of states and symbols that leads to the state q_1, occurs at a fork if and only if that fork is the root of the tree it belongs to. Furthermore, whether a vertex satisfies any of the formulae given above, depends only on the tree it belongs to. It remains to show that for any tree T, $T \models \psi$ if and only if $T \in \mathcal{T}$.

Claim 1. *For any tree T, $T \models \psi$ if and only if $T \in \mathcal{T}$.*

Proof. For the *only if* direction, suppose that for some tree T, $T \models \psi$. We say that a node x in a tree T is of height h if the height of the subtree of T rooted at x is of height h. We prove that for any vertex a, $T \models$ state-$q_i(a)$ if and only if A assigns the state q_i to a in T. Suppose this is not the case and let a be the vertex of minimal height such that $T \models$ state-$q_i(a)$ but A does not assign the state q_i at a. We consider the three possible cases for a, namely that a is either a leaf, a unary vertex or a fork. Notice that for the latter two cases, the statement holds for all vertices below a by assumption. We show that each case leads to a contradiction. If a is a leaf, then A assigns the state q_1 to a by definition of the automaton A. But it also holds that $T \models$ state-$q_1(a)$ and $T \models \neg$state-$q_2(a)$, by definition of the two formulae.

Suppose then that a is a unary node, and let b be the closest descendant of a that is either a fork or a leaf. By the minimality of the height of a, $T \models$ state-$q_j(b)$ if and only if A assigns the state q_j to b in T. Let it be the case that A assigns the state q_j to b. Then, $T \models$ state-$q_i(a)$ if and only if the unary branch from b to a satisfies ϕ_{q_j,q_i}. This is the case if and only if the unary branch from b to a belongs to the set of words on which if A starts at the state q_j, finishes at the state q_i, which is a contradiction.

Finally, suppose a is a fork of minimal height on which A and the formula state-$q_i(x)$ disagree. Let b and c be the two children of a and notice that by the minimality of the height of a, the statement holds for b and c. Assume that a is a fork other than the root. By definition of the

formulae given above, if this is the case then $T \models$ state-$q_2(a)$ and $T \models \neg$state-$q_1(a)$. Since by assumption the statement does not hold for a, A assigns the state q_1 to a. Therefore, there must be $q_j, q_{j'}, \sigma, \sigma'$ such that $\delta_2(q_j, \sigma, q_{j'}, \sigma') = q_1$ and A assigns the state q_j to b, the state $q_{j'}$ to c and the edges from b to a and from c to a have labels σ and σ' respectively. But $T \models \psi$ and according to the formula $\varphi_{q_1}(a, b, c)$ this can only be the case if a is the root, which is a contradiction. The argument for when a is the root and not an inner fork of the tree T is similar, and therefore, for any a, $T \models$ state-$q_i(a)$ if and only if A assigns the state q_i to a in T, for $q_i \in \{q_1, q_2\}$.

For the *if* direction, fix a tree T and suppose that $T \in \mathcal{T}$. Therefore, A assigns the state q_1 to the leaves and the root of the tree T, and the state q_2 to all forks other than the root. To show that $T \models \psi$ we only need to show that for any x, $(\exists y, z\ y \neq z \wedge \varphi_{q_1}(x, y, z)) \leftrightarrow \text{root}(x)$, since the root of T by assumption is a fork. For any vertex x that is not a fork, the formula holds. Suppose for contradiction that there is a vertex a other than the root that is a fork, with children b and c, such that $T \models \varphi_{q_1}(a, b, c)$. First note that for any vertex x' in T, the formulae state-$q_i(x')$ and the automaton A agree. This is because of the following. If x' is a leaf, then by definition of the formulae state-$q_i(x')$ and the assumption that $T \in \mathcal{T}$, it holds that $T \models$ state-$q_1(x')$, $T \models \neg$state-$q_2(x')$ and A assigns the state q_1 by definition. Similarly, if x' is a fork other than the root, it holds that $T \models$ state-$q_2(x')$, $T \models \neg$state-$q_1(x')$ and A assigns the state q_2 to x', and if x' is the root, the statement holds for similar reasons. Finally, if x' is a unary vertex, then there exists y' that is either a leaf or a fork and the path from y' to x' is a unary branch. Furthermore, according to the above, A and the formulae state-$q_i(x)$ agree on y', and therefore A agrees with the formulae on the vertex x' as well, by the definition of φ_{q_i,q_j}.

Therefore, for the vertices b and c, which are the two children of a, it is the case that $T \models$ state-$q_i(b)$ (respectively, $T \models$ state-$q_j(c)$) if and only if A assigns the state q_i to b (respectively, A assigns the state q_j to z). If a, b and c satisfy the formula $\varphi_{q_1}(x, y, z)$, then there exist $q_j, q_{j'}, \sigma, \sigma'$ such that $\delta_2(q_j, \sigma, q_{j'}, \sigma') = q_1$, and furthermore $T \models$ state-$q_j(b)$, $T \models$ state-$q_{j'}(c)$ and σ, σ' are the labels of the edges from b to a and form c to a, respectively. But since, the automaton A and the formulae state-$q_i(x)$ agree on b and c, it means that A assigns the state q_1 to a, which is a contradiction by the assumption that $T \in \mathcal{T}$. The argument is similar for the case where a is the root of the tree T. □

This completes the proof of Theorem 2.3.1. □

The following Corollary is a direct consequence of Theorem 2.3.1.

Corollary 2.3.2. *The class of binary trees with an even number of leaves is* GL *definable.*

A natural question that arises is whether the languages accepted by a product of 2-state deterministic bottom-up tree automata, capture all the regular languages. This question can be refuted though by a counting argument. The set of regular languages is not finite, but the set of languages accepted by a product of 2-state automata is.

Lemma 2.3.3. *For any two deterministic bottom-up binary tree automata A and B, the language accepted by their product $A \times B$ is equal to the intersection of the languages of A and B.*

Proof. For each tree T, by Definition 1.3.4, $T \in L(A \times B)$ if and only if there is an accepting run of A on T and also an accepting run of B on T. Therefore $T \in L(A \times B)$ if and only if $T \in L(A) \cap L(B)$. □

Using the lemma above we can prove that there are regular languages that are not accepted by any product of 2-state automata.

Theorem 2.3.4. *For any fixed alphabet Σ, the set of languages on the alphabet Σ accepted by a product of 2-state deterministic bottom-up tree automata, is finite.*

Proof. If the size of the alphabet Σ is m then even without considering possible symmetries there are at most $k = 2^{m^2 \cdot 2 \cdot 2}$ different transition functions on 2 states, and hence depending on which states are in the set of accepting states we have at most $4 \cdot k$ different automata on Σ with 2 states. Therefore, since the operation of taking intersection of two sets is associative, commutative and idempotent, the number of different languages accepted by products of 2-state deterministic bottom-up tree automata, is finite. □

2.3.1 Chain and Antichain Logic

It is shown in [Tho84] and [PT93], that for binary ordered trees that do not allow unary branching, Antichain Logic, a variant of MSO which is defined below, is enough to express all properties definable in MSO. In the same paper it is also shown that on binary trees with unary branching Antichain Logic is not enough to express all MSO definable properties. Another version of MSO, termed Chain Logic, is shown similarly to express some MSO definable properties on trees with unary branching, and in particular is equivalent to MSO over words, over which Antichain Logic is equivalent to FO. In [Tho84], Thomas shows that there is a regular binary tree language that is expressible in neither of the two logics. We show that this language is expressible in GL and therefore, GL is not included in the union of Chain and Antichain Logic, and therefore also not included in either of them.

Consider the partial order on vertices of a tree obtained by taking the transitive closure of the child relation. In other words, let $x \prec y$ hold for two vertices x and y when x is an ancestor of y. It is known that $\prec$ is definable in MSO even when it is not present as a relation in the signature. Despite that, we use MSO[$\prec$] to denote the logic MSO, when such a relation is given explicitly. We remind the reader of the following.

Definition 2.3.5. *Let S be a word or a tree, and let $\prec$ denote the transitive closure of the child relation of S. Then a subset X of the vertices of S is an antichain if for all x and y in X, neither $x \prec y$ nor $y \prec x$ holds.*

Definition 2.3.6 ([Tho84])**.** *Antichain Logic defined over the class of trees, is a version of* MSO[$\prec$] *having the same syntax as* MSO[$\prec$] *and is such that for any tree T, $T \models \exists X\ \varphi$, with X*

being a set variable, if and only if there exists a subset A of the vertices of T that is an antichain and T satisfies φ when all the free occurrences of the set X are interpreted by A.

Therefore the syntax of Antichain Logic is like the one for MSO, but the semantics allow the second order variables to range only over antichains. Chain Logic is defined in a similar manner with the difference that second order variables range over chains. A set Y is a chain if for any $w, z \in Y$, either $w \prec z$ or $z \prec w$. According to the definition of antichains, any second order variable in Antichain Logic over words, can only contain a single element of the universe, and therefore an Antichain Logic formula over words is equivalent to a First Order Logic one. Similarly, Chain Logic over words is equi-expressive to MSO, since any subset is a chain.

In a tree without unary branching, Antichain Logic is as expressive as MSO and informally this can be achieved as follows. We give an illustration of the argument presented in [PT93], for binary ordered trees in particular. For any tree T let V_T^L be the set of leaf vertices and V_T^I the set of all non-leaf vertices of T. According to this, for any tree T, let $f_T : V_T^I \to V_T^L$ be an injection from the set V_T^I to the set V_T^L, defined as follows. For any $x \in V_T^I$, let x_1 be its right child. Then the leaf $f_T(x)$ is the unique leaf node that can be reached from x_1 using only the edges connecting the vertices to their left child. In the presence of the relation $\prec$, this function is definable in FO, as shown in [PT93]. Suppose that $\psi_f(x, y)$ is such an FO formula that holds exactly for any two vertices x and y when $f_T(x) = y$.

By Theorem 1.3.3 given by Thatcher and Wright, and in particular as a result of the formulae constructed in the proof of the theorem, any MSO formula over trees is equivalent to an existential one, namely one of the form $\exists X_1, \ldots, X_n\ \chi(X_1, \ldots, X_n)$, with $\chi(X_1, \ldots, X_n)$ being an FO formula. In this form, an MSO formula over binary ordered trees is translated inductively by replacing any occurrence of $\exists X\ \varphi$ in the formula, with $\exists X_1, X_2\ \varphi'$, where φ' is the translation of φ, and similarly every atomic formula $X(x)$ is replaced with

$$(\text{leaf}(x) \wedge X_1(x)) \vee (\neg\text{leaf}(x) \wedge \forall y\ (\psi_f(x, y) \to X_2(y))).$$

Essentially, each set variable X is translated into two sets, one containing the leaves of X and the other containing leaves that encode through the function f_T, the non-leaves of X. For a formal introduction to this idea we refer the reader to [PT93].

Since words are trees, Antichain Logic is not expressive enough to define all regular tree languages. Furthermore, in [Tho84], it is proved that there exists a binary tree regular language that is neither Antichain nor Chain definable. We shall see that this tree language is GL definable. We present the theorem given in [Tho84] slightly changed below.

In what follows, for any $n \in \mathbb{N}$, and some fixed alphabet $\Sigma = \{a, b, c\}$, let S_n denote a string with n edges whose leaf, and only its leaf, is labelled with a. For $a, b \in \Sigma$, let $\mathcal{T}_{a,b}$ denote the class of proper binary trees whose leaf vertices, and only those, are labelled with either a or b. Finally, if $\mathcal{T}_1$ and $\mathcal{T}_2$ are two classes of trees, with $\mathcal{T}_1$ having leaves (not necessarily all of them) labelled with some label $d \in \Sigma$, we denote with $\mathcal{T}_1 \cdot_d \mathcal{T}_2$, the class of all trees T that result from concatenating trees of the class $\mathcal{T}_2$ to all the leaves of a tree in $\mathcal{T}_1$, which are labelled with d.

Theorem 2.3.7 (Thomas)**.** *From the tree languages below, the tree languages $\mathcal{T}_1$ and $\mathcal{T}_2$ are Chain definable but not Antichain definable, the tree language $\mathcal{T}_3$ is Antichain definable but not Chain definable, and the tree language $\mathcal{T}_4$ is a regular language but is neither Chain definable nor Antichain definable.*

$$\begin{aligned}
\mathcal{T}_1 &= \{S_n \mid n \text{ even}\},\\
\mathcal{T}_2 &= \{S_n \mid n \text{ odd}\},\\
\mathcal{T}_3 &= \{t \in \mathcal{T}_{a,b} \mid t \text{ contains an odd number of } a\text{'s}\},\\
\mathcal{T}_4 &= (\mathcal{T}_3 \cdot_a \mathcal{T}_1) \cdot_b \mathcal{T}_2.
\end{aligned}$$

The alphabet of the binary tree classes $\mathcal{T}_i$ defined above, was only included to define the tree classes and is not used in what follows. Therefore, the class of binary trees $\mathcal{T}_4$ above, contains trees that comprise a proper binary tree T (every vertex has either no children or two children), with strings of even length concatenated to an odd number of leaves of T, and strings of odd length concatenated to the rest of the leaves of T.

Antichain Logic is strictly less expressive than MSO over words and we already know that GL captures MSO on words. Therefore there is a tree language definable in GL and not definable in Antichain Logic. In particular the languages $\mathcal{T}_1, \mathcal{T}_2$ are GL definable. In order to prove that there is a property definable in GL but not in Chain Logic, we must consider trees that are not simply words. To this end, it is enough to show that $\mathcal{T}_4$ is GL definable.

Note that, ignoring the alphabet used, $\mathcal{T}_4$ can be accepted by the product of two 2-state automata A and B, where B checks that the tree has no unary node that is an ancestor of a fork, and A essentially checks the parity condition given above. In particular, let A be the 2-state automaton of the form $(Q, \Sigma, q_0, Q_f, \delta)$, where $Q = \{q_1, q_2\}$, $\Sigma = \emptyset$, $q_0 = q_2$, $Q_f = \{q_2\}$, and the transition function δ is defined as:

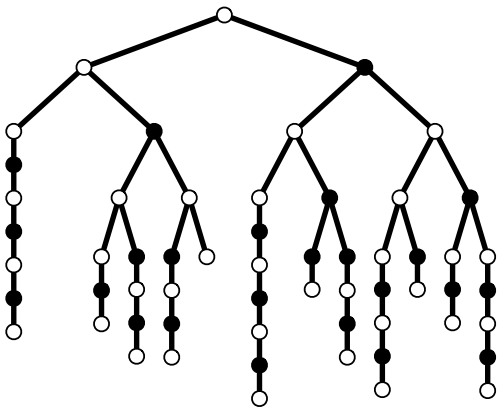
$$\begin{aligned}
\delta_1(q_1) &= q_2, & \qquad \delta_2(q_1, q_2) &= q_2,\\
\delta_1(q_2) &= q_1, & \qquad \delta_2(q_2, q_1) &= q_2,\\
\delta_2(q_1, q_1) &= q_1, & \qquad \delta_2(q_2, q_2) &= q_1.
\end{aligned}$$

An example of a run of the automaton A on a binary tree is shown in Figure 2.2.

By Theorem 2.3.1 this class of binary trees is expressible in GL. We proceed to give a more direct proof of definability of the language $\mathcal{T}_4$ in GL that also illustrates the ideas presented earlier in the form of an example. Let us first define the following GL formulae.

Let T be any tree, and x a vertex where its parent has two children and the subtree of T rooted at x is a single string of even length. Then we call this node x an *even-path-node*. Similarly, a vertex y whose parent has two children and the subtree rooted at y is a single string of odd length, is called an *odd-path-node*.

$$\begin{aligned}
\text{odd}(x) = \; & \exists y \; (\text{leaf}(y) \wedge (\text{OddPath}(x, y) \mid \top))\\
& \wedge \exists w \; ((E(w, x) \mid \top) \wedge \text{fork}(w))\\
& \wedge \forall z \; ((\text{Path}(x, z) \mid \top) \rightarrow \neg \text{fork}(z)),
\end{aligned}$$



$T \in \mathcal{T}_4$

Figure 2.2: A run of the automaton A on a tree $T \in \mathcal{T}_4$. The black-filled nodes ($\bullet$) are assigned the state q_1 and the white-filled nodes ($\circ$) are assigned the state q_2 by the automaton A.

$$\begin{aligned}\text{even}(x) \;=\; & \exists y\ (\text{leaf}(y) \wedge (\text{EvenPath}(x,y) \mid \top)) \\ & \wedge \exists w\ ((E(w,x) \mid \top) \wedge \text{fork}(w)) \\ & \wedge \forall z\ ((\text{Path}(x,z) \mid \top) \rightarrow \neg \text{fork}(z)).\end{aligned}$$

The two formulae above define the odd-path-nodes and even-path-nodes respectively. A node that is an even-path-node will be called an *Even-node* and a node that is either a node with two children or an odd-path-node will be called an *Odd-node*. Notice that a leaf whose parent is a fork is also an even-path-node, and therefore also an Even-node. A vertex b in a tree T is an Even-node (respectively Odd-node) if and only if $T \models \text{EvenType}(b)$ (respectively $T \models \text{OddType}(b)$) defined below.

$$\begin{aligned}\varphi_1 \;&=\; \forall x,y,z\ \left(\left(\begin{array}{ll} & z \neq y \\ \wedge & (E(x,y) \mid \top) \\ \wedge & (E(x,z) \mid \top)\end{array}\right) \rightarrow \Big(\text{DiffType}(y,z) \leftrightarrow \text{root}(x)\Big)\right), \\ \varphi_2 \;&=\; \forall x\ (\text{root}(x) \rightarrow\ \text{fork}(x)), \\ \text{DiffType}(y,z) \;&=\; (\text{EvenType}(y) \wedge \text{OddType(z)}) \vee (\text{OddType}(y) \wedge \text{EvenType}(z)), \\ \text{EvenType}(y) \;&=\; \text{even}(y), \\ \text{OddType}(z) \;&=\; \text{fork}(z) \vee \text{odd}(z).\end{aligned}$$

Let also the formula χ defined below, be the GL formula that defines the set of trees where no unary node is an ancestor of a fork.

$$\chi = \forall x,y\ \big(\text{one-child}(x) \wedge \text{fork}(y) \rightarrow \neg(\text{Path}(x,y) \mid \top)\big).$$

Using the above formulae, one can define the language $\mathcal{T}_4$ in GL using the formula $\psi_T =$

$\chi \wedge (\varphi_1 \wedge \varphi_2 \mid \varphi_1 \wedge \varphi_2)$. This formula essentially splits the tree into two subgraphs, each one of them comprising a set of components, the roots of which have two children.

Lemma 2.3.8. *If a connected tree T' satisfies φ_1 then the set X containing the Even-nodes of T' is of odd cardinality. Furthermore, for every node y in the tree with two children other than the root, the set X_y of Even-nodes below it, is of even cardinality.*

Proof. Suppose a tree T' satisfies φ_1. We first prove the second statement of the Lemma by induction on the height of the subtree rooted at each node. For nodes whose two children are leaves or even-path-nodes, the statement is true. Assume the statement is true for nodes with subtrees rooted at them of height less than h for some $h \in \mathbb{N}$. Let y be a fork where the subtree of T' rooted at y is of height h. By inspecting the formula φ_1, if y is not a root, then either both its children are Even-nodes, or they are both Odd-nodes.

In the first case, where both children of y are leaves or even-path-nodes, the case is similar to the base case above and X_y is of even cardinality. In the second case, suppose y_1, y_2 are the children of y. If one of them, say y_1, is a fork, then by the inductive hypothesis X_{y_1} is of even cardinality. If it is an odd-path-node then again X_{y_1} is of even cardinality. Therefore for each y_i, for $i = 1, 2$, X_{y_i} is of even cardinality and hence so is X_y.

For the first statement of the lemma note that if r is the root of the tree T' and r_1, r_2 are its children, one of them (say r_1) is an Even-node and the other one (r_2) is an Odd-node. The set $X_{r_1} = \{r_1\}$ is hence of odd cardinality and as it follows from the second statement of the Lemma, that we proved above, the set X_{r_2} is of even cardinality. Therefore $X_r = X$ is of odd cardinality as required. □

Theorem 2.3.9. *A tree T satisfies the* GL *formula $\psi_T = \chi \wedge (\varphi_1 \wedge \varphi_2 \mid \varphi_1 \wedge \varphi_2)$ if and only if $T \in \mathcal{T}_4$.*

Proof. For the *only if* direction assume T satisfies ψ_T. Then T can be split up into subgraphs G_1, G_2, each satisfying $\varphi_1 \wedge \varphi_2$. Because of φ_2, in each of the two subgraphs, all components have a root with two children. This respects the degree of the nodes with exactly one child in a tree, in the sense that if a node z has a single child in one of the subgraphs that satisfy $\varphi_1 \wedge \varphi_2$ then it also has exactly one child in T as well, and vice versa. Consequently, for any connected component of G_1 that in the original tree T is connected to a component of G_2, or the other way round, it is the case that the root of one of them will be the leaf of the other in the original tree T.

Now since each component in each of the two subgraphs satisfies φ_1, by Lemma 2.3.8 we have that for each such component the set containing the Even-nodes of this component has odd cardinality. Connecting any two such components, preserves the first property of Lemma 2.3.8, since T satisfies χ and therefore the vertex that is common to these two components is a leaf in one of the two, with a parent that is a fork, and consequently is in the set of Even-nodes of that component. Hence the tree T belongs to the class of trees $\mathcal{T}_4$. An example is shown in Figure 2.3.

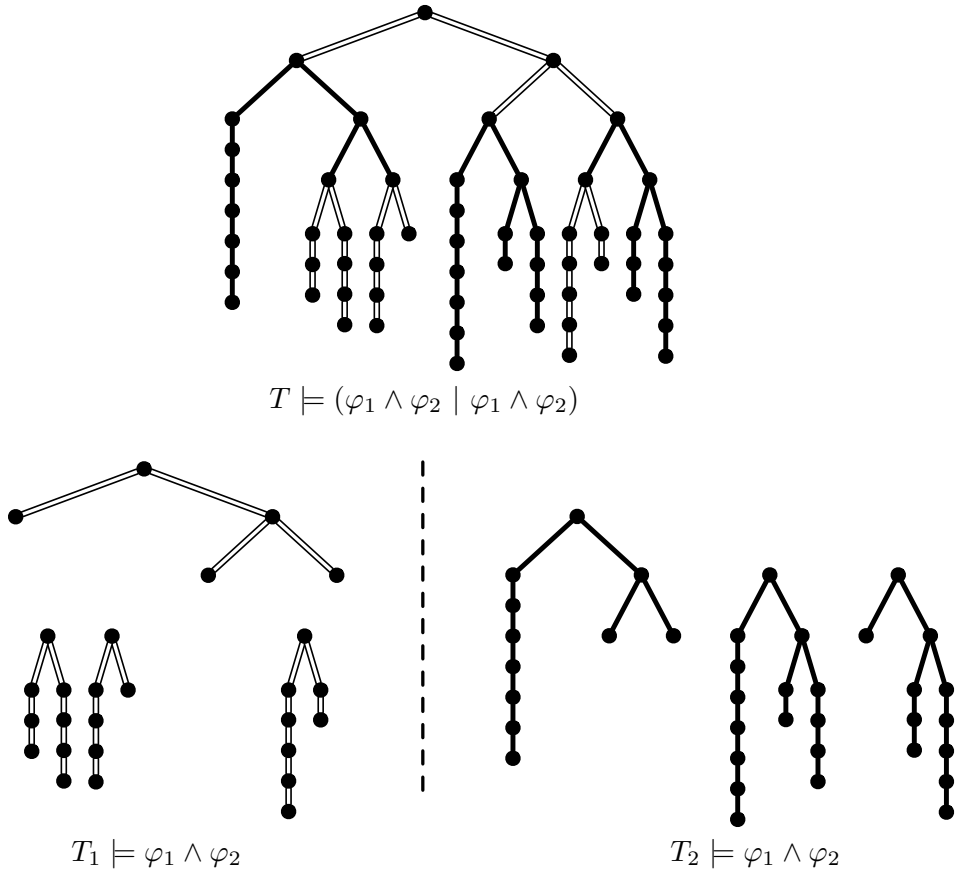


Figure 2.3: A tree T that satisfies $(\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \wedge \varphi_2)$.

For the *if* direction we proceed by induction on the number of nodes n of the tree T that have two children. The base cases of $n = 1$ and $n = 2$ satisfy ψ_T. There are three possible cases for $n = 2$, but only in one of them do we need to split the tree in two components, each of which satisfies $\varphi_1 \wedge \varphi_2$.

Assume then that for any tree T' where n is less than h, for some $h \in \mathbb{N}$, if $T' \in \mathcal{T}_4$ then $T' \models \psi_T$. We have already proved that if $T' \models \psi_T$ then $T' \in \mathcal{T}_4$. Let us consider a tree T in the class of trees $\mathcal{T}_4$ where $n = h$. For every path P from the root to a leaf, there is a non empty set X_P of nodes in P other than the root of T, that are either leaf nodes or even-path-nodes or nodes where the subtree rooted at them is in $\mathcal{T}_4$. Let X be the largest set of nodes in $\bigcup_P X_P$ that have no ancestor in $\bigcup_P X_P$. Finally let Y be the subset of the nodes in X that are nodes where the tree rooted at them is in $\mathcal{T}_4$.

For each $y \in Y$ the subtree rooted at y, which will be denoted by T_y, is by definition in $\mathcal{T}_4$ and has less than h nodes with two children. Hence, by the inductive hypothesis, each T_y can be split up into two subgraphs, each containing connected trees satisfying $(\varphi_1 \wedge \varphi_2)$. It remains to show that the tree obtained from T by removing the trees T_y for each $y \in Y$, satisfies $(\varphi_1 \wedge \varphi_2)$.

Let us call T_{-Y} the tree obtained after this transformation. The root of T and both its children remain in T_{-Y} and hence $T_{-Y} \models \varphi_2$.

Any node with two children in T_{-Y} other than the root has, in the original tree T, a subtree rooted at it that is not a member of $\mathcal{T}_4$, since otherwise it would be in the set Y defined above. Now consider such a node x in T_{-Y} with two children, other than the root. It cannot be the case that one of its children is an Even-node and the other one an Odd-node since that would entail that the subtree T_x rooted at x is a member of $\mathcal{T}_4$, by the inductive hypothesis. Similarly if r is the root of T, and since by assumption T is in the class of trees $\mathcal{T}_4$, one of its children must have a subtree rooted at it that belongs to $\mathcal{T}_4$ and the other one must not. This is what φ_1 says about the root, and therefore $T \models \varphi_1 \wedge \varphi_2$. □

Corollary 2.3.10. *There exists a regular tree language that is neither Chain definable nor Antichain definable, but is* GL *definable.*

In the paper [Tho84] by Thomas, a connection is given between the Chain definable tree languages and the class of tree languages recognized by deterministic top-down automata. Since, the latter does not form a Boolean algebra, its closure under boolean operations is considered and named Boolean Deterministic Tree Languages. It is stated in that paper, that the languages in this class can be defined in Chain Logic. We prove here a similar result for the relationship between Boolean Deterministic Tree Languages and GL, namely that the former are GL definable.

We will mainly consider ranked trees and in particular binary ones. In any binary ordered tree recognized by a deterministic top-down tree automaton, the state assumed at some node depends only on the state of its parent node. Ordered binary trees are represented in GL by indexing the labels used for the edges using the indices 0 and 1 for left and right child respectively. If the alphabet of the trees is Σ, we define the labels for the edges in the tree representation for GL to be the set $\{(\sigma, i) \mid \sigma \in \Sigma, i \in \{0,1\}\}$. An edge labelled by $(a, 1)$ for $a \in \Sigma$, is thus a right edge with letter a.

In order to show that the Boolean Deterministic Tree Languages are GL definable, we only need to establish that any binary tree language recognized by a deterministic top-down tree automaton is GL definable, since GL is closed under boolean operations.

Let $A = (Q, \Sigma, q_i, \delta, Q_f)$ be a deterministic top-down binary tree automaton, with transition function $\delta : Q \times \Sigma \to Q \times Q$. We define the *path automaton* of A to be the automaton $B = (Q, \Sigma, q_i, \delta', Q_f)$ on words, where the transition function is given by $\delta' : Q \times \Sigma \times \{0,1\} \to Q$. The function δ' is the projection of δ to the first or second output states according to the input $\{0,1\}$. Thus for any states q_1, q_2, q_3 and alphabet symbol σ, it is the case that $\delta(q_1, \sigma) = (q_2, q_3)$ if and only if $\delta'(q_1, \sigma, 0) = q_2$ and $\delta'(q_1, \sigma, 1) = q_3$. Note that the path automaton is unique up to the renaming of states.

Lemma 2.3.11. *Let $A = (Q, \Sigma, q_i, \delta, Q_f)$ be a deterministic top-down binary tree automaton and let $B = (Q, \Sigma, q_i, \delta', Q_f)$ be the path automaton associated with A. Then a binary tree T is accepted by A if and only if for every leaf, the path from the root to the leaf is accepted by B.*

Proof. For the *only if* direction assume that T is accepted by A. Then the automaton A reaches an accepting state at the end of each path of the tree T and so the path automaton B using the transition function δ' will also reach an accepting state at the end of each path.

For the *if* direction assume that every path from the root to a leaf in T is accepted by B. Take any two paths P_1 and P_2 from the root to leaf vertices y_1 and y_2 respectively. These two paths will have some initial segment in common. Let x be the vertex at which the two paths split, and let P denote the initial segment that is common to both paths and starts at the root and finishes at the vertex x. Since the path automaton is deterministic, in every run that goes through x the same state is always assumed there. Let that state be q_x, and let the symbol at x be σ_x. Denote with ρ_1 the run of the path automaton according to the transition function δ', on the path P_1, and similarly denote with ρ_2 the run on P_2. The children of x, will be assigned a state by the runs ρ_1 and ρ_2, say q_0 at the left child x_2, and q_1 at the right child x_1. Hence $\delta'(q_y, \sigma_y, 0) = q_0$ and $\delta'(q_y, \sigma_y, 1) = q_1$. By the definition of a path automaton, this can be the case only if $\delta(q_y, \sigma_y) = (q_0, q_1)$ and since A is deterministic only one output is possible. This means there is a successful run of the automaton A on T as well, since the above hold for any node such as x. The latter follows from the fact that the two paths P_1 and P_2 were chosen arbitrarily. □

Theorem 2.3.12. *Let L be any tree language of binary ordered trees, recognized by some deterministic top-down tree automaton A. There exists a* GL *formula ψ_L such that for any binary ordered tree T, $T \models \psi_L$ if and only if $T \in L$.*

Proof. Fix a deterministic top-down tree automaton A and let B be the path automaton associated with A. We define a word automaton $C = (Q, \Sigma_C, q_i, \gamma, Q_f)$ according to the following. The state space is the same as the one used for A and B and $\Sigma_C = \{(\sigma, i) \mid \sigma \in \Sigma, i \in \{0, 1\}\}$. Furthermore we define the transition function γ of C, to be such that for any $q, (\sigma, i)$ and q', it is the case that $\gamma(q, (\sigma, i)) = q'$ if and only if $\delta'(q, \sigma, i) = q'$.

Remember that the representation of binary ordered trees in GL is using indexed labels on edges as symbols for the alphabet, and the index indicates whether the edge leads to the left or right child of a node. The two automata B and C are essentially the same and according to Theorem 2.2.1, any language accepted by some regular word automaton is GL definable. Let I be the path language or equivalently word language, accepted by the automaton C, and let r and ϕ_r be the regular expression and the GL formula corresponding to it, respectively. We define ψ_L to be the following formula:

$$\psi_L = \forall x, y \Big(\text{root}(x) \wedge \text{leaf}(y) \rightarrow \neg\big(\top \mid (\text{Path}(x, y) \wedge \neg\phi_r)\big)\Big).$$

In the above formula $\text{Path}(x, y)$ is defined as usual. In addition, ψ_L states that we cannot find a subgraph that is exactly a path from the root to a leaf, that does not satisfy ϕ_r. In other words, ψ_L expresses that any path from the root to a leaf satisfies ϕ_r and therefore it is accepted by the automaton C. The theorem follows by the definition of C and Lemma 2.3.11. □

Hence the class of tree languages definable in GL contains any boolean combination of binary ordered tree languages accepted by some deterministic top-down automaton. This result can be extended to ranked trees in general for any rank k.

2.3.2 Tree Walking Automata

Aho and Ullman introduced the tree-walking automata in [AU71], and although it is clear that the set of languages they can define is a subset of regular tree languages, the other inclusion remained open until Bojańczyk et al. gave a negative answer to the problem in [BC08]. This means that the set of tree languages definable by tree-walking automata is a proper subset of regular tree languages, and showing that GL over trees is subsumed by tree-walking automata would be sufficient for separating GL from regular tree languages. On the contrary, however, we show that the class of GL-definable tree languages is not included in the class of tree languages accepted by tree-walking automata. In particular, we show that the separating language used by Bojańczyk et al., which is not accepted by any tree-walking automaton, is GL definable.

A tree walking automaton is one that informally starts at the root of a binary tree and traverses the tree visiting each node three times. Once when traveling down the vertices of the tree, once when going up and having visited all the nodes of the left subtree, continuing to the nodes of the right subtree, and finally once more when having finished visiting the nodes of the latter as well.

A tree-walking automaton is defined as a tuple $A = \langle Q, \Sigma, q_I, \Delta \rangle$, where Q is the set of states, Σ is the alphabet used to label the nodes of the tree, $q_I \in Q$ is the initial state and Δ is the transition relation. When the automaton is over some node x of the tree, it can check whether x is a right child, a left child, a root or a leaf and it can also check the label of the node. According to the results of such possible tests and the state the automaton has assumed over the particular node x, the transition relation describes possible combinations of what command to execute and what state to assume next. The commands range over moving to the parent of x, or moving to the left or right child of x.

Every tree-walking automaton can be translated into a bottom-up tree automaton, and therefore any language accepted by some tree-walking automaton is a regular tree language. Bojańczyk et al. showed in [BC08] that this inclusion is actually proper.

Definition 2.3.13. *Let T be some binary tree of alphabet $\Sigma = \{a, b\}$, where the label a is allowed only at the leaves of T. The branching structure of T is defined as the tree S obtained from T, by keeping all the leaves labelled with a, and the closest common ancestors (of minimum height) between any two such leaves labelled with a, and finally contracting the paths between these nodes that exist in T, into single edges.*

Let K be the language that contains the binary trees for which it is the case that for every leaf labelled with a there is an even number of proper ancestors in the branching structure of the tree.

Theorem 2.3.14 (Bojańczyk et al., [BC08])**.** *Tree-walking automata, even nondeterministic ones, do not capture all regular tree languages. In particular, there is no tree-walking automaton accepting the language K.*

We proceed by showing that binary tree languages definable in GL are not contained in the set of languages accepted by tree-walking automata. In particular, we prove that K is definable in GL. Consider the following GL formulae that define the binary tree language K.

$$
\begin{aligned}
\text{TwoBranches} &= \forall z \, \big(\text{fork}(z) \to \exists! w \, (\text{fork}(w) \wedge (\text{Path}(z,w) \vee \text{Path}(w,z) \mid \top))\big),\\
\text{Split} &= \text{TwoBranches} \mid \text{TwoBranches},\\
\text{Trunk}(x,y) &= \forall z \, (\text{fork}(z) \to (\text{Path}(x,z) \mid \text{Path}(z,y) \mid \top)) \wedge\\
&\quad \wedge \text{Split} \wedge \text{root}(x) \wedge \text{leaf}(y) \wedge \text{Connected},\\
\text{fork-roots} &= \forall x \, (\text{root}(x) \to \text{fork}(x)),\\
\text{Even}(x,y) &= (\text{fork-roots} \mid \text{Trunk}(x,y)),\\
\text{EvenBranch} &= \forall x, y \, ((\text{root}(x) \wedge \text{leaf}(y)) \to \text{Even}(x,y)),\\
\text{Only-a} &= \forall x, y \, \Big(\big(\text{leaf}(x) \wedge (E(y,x) \mid \top)\big) \to (E_a(y,x) \mid \top)\Big),\\
\text{Only-b} &= \forall x, y \, \Big(\big(\text{leaf}(x) \wedge (E(y,x) \mid \top)\big) \to (E_b(y,x) \mid \top)\Big),\\
\varphi_K &= (\text{Only-b} \mid (\text{Only-a} \wedge \text{Connected} \wedge \text{fork-roots} \wedge \text{EvenBranch})).
\end{aligned}
$$

The formula TwoBranches above, is satisfied by graphs comprising disjoint subtrees where each subtree has exactly 2 forks. We remind the reader that a fork is a vertex with 2 children. The formula $\text{Trunk}(x,y)$ is satisfied by any tree where x is the root, y is a leaf, every fork in the tree is in the path from x to y, and finally the tree can be split into two subgraphs that satisfy TwoBranches. Therefore every tree of this form that satisfies $\text{Trunk}(x,y)$ has an even number of forks.

The formula EvenBranch with the help of $\text{Even}(x,y)$, expresses that if x and y is the root and a leaf of a tree respectively, then this tree can be split into a forest and a connected tree, where each tree in the forest has a root that is a fork, and the connected tree satisfies $\text{Trunk}(x,y)$. Thus, the formula φ_K expresses that in the subgraph S of a tree T resulting from removing all subgraphs with leaves labelled b and leaving all leaves of T labelled a, for every leaf y of the remaining tree S there is a subtree of S that satisfies $\text{Trunk}(x,y)$, with x the root of S. Essentially, the number of forks in S between the root and any leaf is of even number.

Theorem 2.3.15. *A binary tree T satisfies the* GL *formula φ_K if and only if T is in the tree language K.*

Proof. Let S be the branching structure of a tree T and r_S the root of S. We call a substructure of a tree T the *extended branching structure* of T if it can be obtained from the branching structure S of T by replacing each edge e of S, with endpoints v_1, v_2, with a path of the same length as the path in T between the vertices corresponding to v_1 and v_2, and furthermore the vertex corresponding to r_S is the root of the extended branching structure of T.

Note first that a tree T satisfies (Only-b | (Only-a $\wedge$ Connected $\wedge$ fork-roots $\wedge \psi$)) if and only if its extended branching structure satisfies ψ. This is because if $T = T_1 \mid T_2$ where $T_1 \models$ Only-b and $T_2 \models$ Only-a $\wedge$ Connected $\wedge$ fork-roots $\wedge \psi$ then T_1 contains only leaf edges that are labelled with b and T_2 contains only leaf edges that are labelled with a. Therefore, since a appears only at the leaves of T_2, and since T_2 is connected, it must be the case that T_2 is a subtree of T and the formula fork-roots ensures that it is the extended branching structure of T.

It remains to show that the extended branching structure of a tree T satisfies EvenBranch if and only if the number of proper ancestors of any leaf in the branching structure S of T, is even. The latter condition is exactly what is needed for a tree T to be in the language K.

For the *if* direction, suppose that the branching structure of T satisfies the required condition. Then the path between any leaf and the root of the branching structure has an even number of nodes. These nodes in the branching structure always have two children unless they are leaves. In the extended branching structure, these nodes correspond to the nodes with two children, and therefore the extended branching structure must have an even number of forks on the path from the root to any leaf. We need to prove that this is what the formula Even(x, y) asserts about the extended branching structure, with x being the root of it and y any leaf.

Let B be the extended branching structure of T with root x and fix a leaf y. Let X be the nodes on the path from x to y that have two children. Note that the nodes in X correspond exactly to the nodes on the path from y to r_S in the branching structure S. Remove from B all subtrees rooted at nodes with two children that are descendants of the nodes in X in B, and are such that contain no node $x \in X$. This is a tree B' where for any two nodes with two children, one is an ancestor of the other. Finally split B' into two substructures each containing only connected components, each with exactly two nodes from X. This is possible from the assumption. The transformation above is exactly what the formula Even(x, y) asserts and hence B satisfies EvenBranch.

For the *only if* direction assume that the extended branching structure B of T satisfies EvenBranch. Then for the root x' and any leaf y' of B the following holds. One can remove subtrees of B each with a root being a fork so that the remaining graph is a connected one such that for any two forks, one is an ancestor of the other, and x' and y' are in the remaining subgraph. Let this tree be B'. The formula Split asserts that B' can be split into two substructures each with connected components that have exactly two forks. Since these forks are the ancestors of y' in the branching structure S of T, the above can only hold if the number of ancestors in S is even. □

As previously stated, this result establishes that GL-definable binary tree languages are not included in the ones definable by tree-walking automata. For our purposes this shows that tree-walking automata cannot be used to prove that the class of GL-definable tree languages is a strict subset of the class of regular tree languages. It does however show that there are GL definable classes of trees that are not definable by tree walking automata.

Chapter 3

GL$\subsetneq$MSO on Forests

In this chapter we present the main result of this thesis, together with a few of its consequences. In particular we show that GL is strictly less expressive than MSO over finite graphs in general, and the case is so even when restricted to the class of forests. Whether GL is strictly less expressive than MSO on trees remains open. Because of the properties expressible in MSO being nonlocal, it is expected that any hard enough property on trees would require at least one use of the separation connective of GL, and in particular applied on the whole tree, resulting in many smaller subtrees in each of the two subgraphs. In other words, one would expect a splitting of the tree, not in just one particular vertex, but in many. The two resulting subgraphs in such a case though, would be forests, and as stated above, we show that over forests GL is strictly less expressive. Therefore we conjecture that GL does not define all regular tree languages.

GL was introduced to work over graphs where edges can have a labelling, but the main result below is on forests of only 1 label, namely a graph in the traditional sense, and thus holds for both the class of graphs with no labelling and the class of graphs with labelling.

In addition, we establish that GL is not closed under most natural mappings defined on graphs, such as FO definable interpretations and MSO transductions. Finally, in the last section we show, how our main result can be used to show that Separation Logic without the magic wand operator is strictly less expressive than MSO on memory heaps, with the latter being structures representing pointer variable mappings and are used to reason about programs that involve pointers for memory management.

3.1 Main Theorem

We show that GL is strictly less expressive than MSO over forests, structures comprising disjoint trees. The property we show that is not expressible in GL, but is definable in MSO, is that of a forest containing only trees that have 1 (mod 3) number of leaves. This is accomplished by giving a winning strategy of Duplicator for a GL game on appropriately constructed forests, as defined below. Essentially, the forests are composed of a binary tree and a set of disjoint strings. The existence of these strings is crucial to the proof, and this is shown explicitly in the next

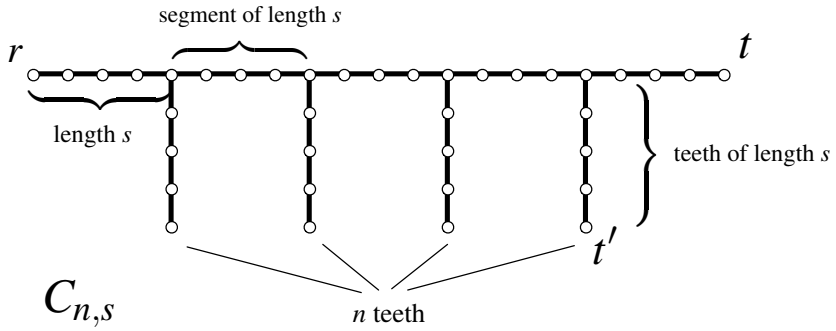


Figure 3.1: The comb $C_{n,s}$.

section (Section 3.2) where these structures are interpreted to single trees on which the property is GL definable. Therefore, the question of whether GL can define all regular tree languages remains open. We proceed by giving a few definitions required for the construction of the forests on which the GL game is played.

Definition 3.1.1. *A* comb *is a binary tree T such that for any two forks v_1 and v_2 in T, either v_1 is an ancestor of v_2 or v_2 is an ancestor of v_1.*

Let C be a comb with root r. As long as there is at least one fork, there are two leaves t and t', such that all forks of C lie in the path from r to t and in the path from r to t'. For each such comb C, we fix t to be one of these two leaves, and define the path from r to t to be the *spine* of C. Each fork a in a comb C has two children, one of which is in the spine of C, and the other is the root of a subtree of C that consists of a single path to a leaf b. For each such fork a, we call the path from a to b, a *tooth* of C. Note that the number of leaves of C is one more than the number of teeth of C.

Suppose that a and a' are two forks such that no fork lies in the path from a to a'. We call such two forks *successive* and the path between them is called a *segment* as shown in Figure 3.1. Furthermore, the segment from a to a' together with the tooth attached to a' is called a *block*. For any $n, s \in \mathbb{N}$, we denote by $C_{n,s}$ the comb with spine r-t that has n teeth (or equivalently has n forks), and the length of each segment and of each tooth is s. Furthermore, if a' is the fork of $C_{n,s}$ such that the path from a' to t contains no forks, the length of this path is also s. The distance from the root to the first fork is also s. Similarly, for $1 \leq i \leq n$, we denote by $C_{n,s}^{-i}$ the comb that is isomorphic to the comb $C_{n,s}$, but with the i-th tooth removed. In other words, the comb $C_{n,s}^{-i}$ has $n-1$ teeth and the segment from the $(i-1)$-th tooth to its successive one is of length $2s$. Finally, we denote by S_s a string of length s, where a string is a graph consisting of a single path.

Lemma 3.1.2. *For each $k \in \mathbb{N}$, there exist $s, n, l \in \mathbb{N}$, such that:*

1. *for any $w \geq s$ and any m, $S_w \equiv_k^{\mathrm{MSO}} S_{w+ms}$,*

2. for any $t > n$, and any m, $C_{t-2l,ms} \equiv_k^{\mathrm{MSO}} C_{t+2,ms}$.

Proof. For the statement (1), let Φ_k be the set of all MSO formulae of rank k up to equivalence, over the class of strings, and for each $\varphi \in \Phi_k$, let L_φ be the string language that φ defines. Notice that the set Φ_k is finite. By the Pumping Lemma, for each such language there exist $p, q \in \mathbb{N}$ such that for any string S_w of length w, where $w > p$, and for all m, $S_w \in L_\varphi$ if and only if $S_{w+mq} \in L_\varphi$, which is of length $w + m \cdot q$. Therefore for any string S of length $w > pq$ and any m, $S_w \in L_\varphi$ if and only if $S_{w+mpq} \in L_\varphi$. For each language L that is equal to L_φ, for some $\varphi \in \Phi_k$, let s_L be equal to the appropriate $p \cdot q$, and let s of the Lemma be the product of all those s_L. Then for each L_φ and each $w \geq s$,

$$S_w \in L_\varphi \Leftrightarrow S_{w+ms} \in L_\varphi$$

and therefore $S_w \equiv_k^{\mathrm{MSO}} S_{w+ms}$.

For (2), fix k, let s be as above, and let Φ_k be the set of MSO formulae of rank k up to equivalence. Since Φ_k is finite, there exists minimum m and r such that the combs $C_{m,s} \equiv_k^{\mathrm{MSO}} C_{m+r,s}$. Notice that by a composition argument, as by Lemma 1.2.3, for any m'', $C_{m''+m,s} \equiv_k^{\mathrm{MSO}} C_{m''+m+r,s}$, and therefore for all $m' \geq m$ it is the case that $C_{m',s} \equiv_k^{\mathrm{MSO}} C_{m'+r,s} \equiv_k^{\mathrm{MSO}} C_{m'+2r,s}$. Let $2l = 2r - 2$ and $n = m + 2l$. □

One consequence of Lemma 3.1.2 is that $C_{n,s} \equiv_k^{\mathrm{MSO}} C_{n+1,s}^{-i}$, for any n and the s given by the Lemma. This is because there are subgraphs C_1, C_2 and S such that S is a string of length s, and $C_{n,s} = C_1 \oplus_{c_1} S \oplus_{c_2} C_2$ for some vertices c_1, c_2, and similarly there are subgraphs C_1', C_2' and S', with S' a string of length $2s$, such that $C_{n+1,s}^{-i} = C_1' \oplus_{c_1'} S' \oplus_{c_2'} C_2'$ for vertices c_1', c_2'. In addition to that, $C_1 \cong C_1'$ and $C_2 \cong C_2'$, and by (1) of Lemma 3.1.2, $S_s \equiv_k^{\mathrm{MSO}} S_{2s}$. Therefore, by the composition argument given by Lemma 1.2.3, $C_{n,s} \equiv_k^{\mathrm{MSO}} C_{n+1,s}^{-i}$.

Lemma 3.1.3. *For any $k \in \mathbb{N}$, there exists λ, such that if $\mathbb{A}$ is the disjoint union of λ pairwise $\equiv_k^{\mathrm{MSO}}$-equivalent structures, and $\mathbb{B}$ is the disjoint union of $\lambda + 1$ such structures, each also equivalent with respect to $\equiv_k^{\mathrm{MSO}}$ to the ones in $\mathbb{A}$, then $\mathbb{A} \equiv_k^{\mathrm{MSO}} \mathbb{B}$.*

Proof. Follows from Lemma 2.2.2 □

Theorem 3.1.4. *The class of forests that contain only trees with* 1 (*mod* 3) *number of leaves, is not definable in* GL.

Proof. Let $\mathcal{F}$ be the class of forests that contain only trees which have 1 (mod 3) number of leaves. We show that for any k, there exist forests F and G, such that $F \in \mathcal{F}$, $G \notin \mathcal{F}$ and Duplicator wins the k-round game on F and G. By Corollary 1.5.5, this is sufficient for showing that the class $\mathcal{F}$ is not GL definable.

Fix $k \in \mathbb{N}$ such that $k \geq 2$, and let s, n, l be as given by the Lemma 3.1.2, for k. Let also λ be as given by Lemma 3.1.3 for k. Notice that by Lemma 3.1.2, for every $w \in \mathbb{N}$, there is $w' \in \{1, \ldots, 2s\}$, such that $S_w \equiv_k^{\mathrm{MSO}} S_{w'}$. We define $N = \prod_{1 \leq i,j \leq 2s}(i + j)$. Then N has the property that for any $i, j \leq 2s$, $(i + j) \mid N$.

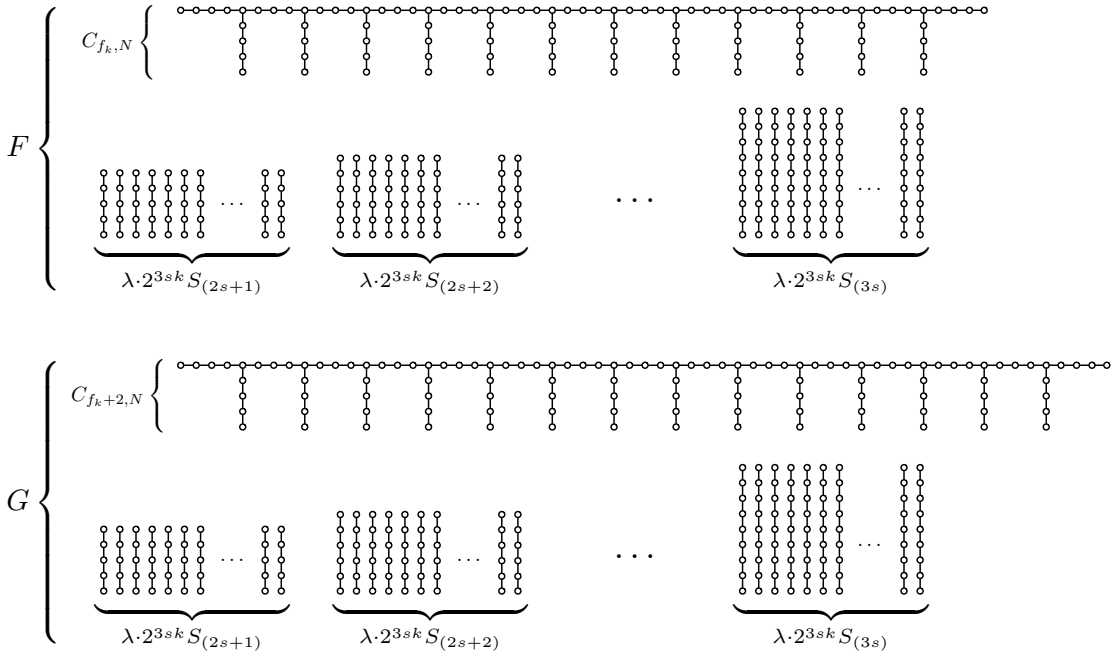


Figure 3.2: The forests F and G.

For each k let $f_k = (2^5 \cdot \lambda s^2 N)^k \cdot 6n \cdot \lambda \cdot 2^{6N}$. For $k \in \mathbb{N}$, we define the forests F and G to be as follows.

$$\begin{aligned} F &= C_{f_k,N} \oplus \bigoplus_{i=2s+1}^{3s} (\lambda \cdot 2^{3sk} S_i), \\ G &= C_{f_k+2,N} \oplus \bigoplus_{i=2s+1}^{3s} (\lambda \cdot 2^{3sk} S_i). \end{aligned}$$

In both forests F and G, the collection of strings isomorphic to $\bigoplus_{i=2s+1}^{3s} (\lambda \cdot 2^{3sk} S_i)$, is called the *noise* of the forests F and G respectively. Each individual string S_i for some $i \in \{2s+1, \ldots, 3s\}$, is called a *noise-string*, and when such a noise-string is of length h, for $h \in \mathbb{N}$, it is called an h-noise-string. The forests F and G are depicted in Figure 3.2.

By Theorem 1.5.3, for each $\ell \geq 2$, and for any two graphs H_1 and H_2, $H_1 \equiv_{\ell}^{\mathrm{MSO}} H_2$ implies that $H_1 \equiv_{\ell-2}^{\mathrm{GL}} H_2$. In the following, to simplify notation we show that for any $k \geq 2$, Duplicator can win the $(k-2)$-round GL game on F and G. To show that $F \equiv_{k-2}^{\mathrm{GL}} G$, we establish that Duplicator can maintain the following condition in the $(k-2)$-round game on F and G.

> If the game position after i rounds is $(F_i, \bar{a})$ and $(G_i, \bar{b})$ then one of the following conditions hold for $k' = k - i$.
>
> 1. $(F_i, \bar{a}) \equiv_{k'}^{\mathrm{MSO}} (G_i, \bar{b})$ or,
> 2. $F_i = F' \oplus_{c_1,c_2} \overline{F'}$ and $G_i = G' \oplus_{d_1,d_2} \overline{G'}$, where:
>   (a) $(F', c_1, c_2) \cong (C_{f_{k'},N}, r, t) \oplus \bigoplus_{i=2s+1}^{3s} (\lambda \cdot 2^{3sk'} S_i)$,
>   (b) $(G', d_1, d_2) \cong (C_{f_{k'}+2,N}, r, t) \oplus \bigoplus_{i=2s+1}^{3s} (\lambda \cdot 2^{3sk'} S_i)$,
>   (c) no element of $\bar{a}$ is in F' and no element of $\bar{b}$ is in G', and

(d) $(\overline{F'}, \bar{a}, c_1, c_2) \equiv^{\mathrm{MSO}}_{k'} (\overline{G'}, \bar{b}, d_1, d_2)$.

Notice that in the above, if j is the number of remaining rounds in the game, then $k' = j+2$. The condition above essentially states, that at each round of the game with $k'-2$ rounds remaining, either both graphs are $\equiv^{\mathrm{MSO}}_{k'}$-equivalent, and thus by Theorem 1.5.3, Duplicator wins the game, or the following holds. In both graphs F_i and G_i after i rounds, there exists a subgraph F' and G' respectively, composed of a large enough comb and enough noise-strings. The set of noise-strings are identical in both subgraphs, but the combs are of different sizes, namely in one of them the number of teeth is 0 (mod 3), whereas in the other it is 2 (mod 3). Furthermore, the complements of these subgraphs F' and G' inside F_i and G_i are $\equiv^{\mathrm{MSO}}_{k'}$-equivalent, and if they are connected to the rest of the structure, they are connected through the edges of the comb. We proceed by showing how Duplicator can guarantee this condition.

Clearly, if $(F_i, \bar{a}) \equiv^{\mathrm{MSO}}_{k'} (G_i, \bar{b})$, then for any move that Spoiler can make, Duplicator has a response that ensures this condition still holds at the next stage. So, we need to prove that if (2) holds, Duplicator can respond to any move by Spoiler and guarantee that the condition still holds. Notice that by definition, condition (2) holds in the beginning of the game, and that even if the game ends with condition (2) being true, Duplicator still wins the game.

In the following, if C is a comb with spine c_1-c_2, and x, y are two vertices in the spine of C, we denote by $[C]^x_y$ the part of the comb between x and y. In other words, $[C]^x_y$ is such that there are C_1, C_2 such that $(C, c_1, c_2) \cong (C_1, c_1, x) \oplus_x ([C]^x_y, x, y) \oplus_y (C_2, y, c_2)$. For short, we may write the latter as $C \cong C_1 \cdot [C]^x_y \cdot C_2$.

We denote by (D, d_1, d_2) the comb isomorphic to $(C_{f_{k'+2,N}}, r, t)$ inside the graph G'. We denote similarly by (C, c_1, c_2) the subgraph of F' that is isomorphic to the comb $(C_{f_{k',N}}, r, t)$. By F_S and G_S we denote the remaining of the graphs F' and G' respectively, that are both isomorphic to $\bigoplus_{i=2s+1}^{3s}(\lambda \cdot 2^{3sk'} S_i)$. We therefore have that $F' = C \oplus F_S$. Similarly for $G' = D \oplus G_S$.

Duplicator's strategy for the second case of the condition is as follows. Let us first consider the case where Spoiler makes a first order move and let us assume that Spoiler chooses a vertex on the graph F_i. The argument is similar for when he chooses a vertex on the graph G_i. By (2), $F_i = F' \oplus_{c_1,c_2} \overline{F'}$. If he chooses to pick a vertex in $\overline{F'}$, then, since $(\overline{F'}, \bar{a}, c_1, c_2) \equiv^{\mathrm{MSO}}_{k'} (\overline{G'}, \bar{b}, d_1, d_2)$, Duplicator can use the strategy for the game on $\overline{F'}$ and $\overline{G'}$ and respond to the vertex Spoiler chose. Duplicator then, can define subgraphs F'^1 and G'^1 of F' and G' respectively such that

$$\begin{aligned} F'^1 &\cong (C_{f_{k'-1,N}}, r, t) \oplus \textstyle\bigoplus_{i=2s+1}^{3s}(\lambda \cdot 2^{3s(k'-1)} S_i), \text{and} \\ G'^1 &\cong (C_{f_{k'-1}+2,N}, r, t) \oplus \textstyle\bigoplus_{i=2s+1}^{3s}(\lambda \cdot 2^{3s(k'-1)} S_i). \end{aligned}$$

which are the appropriate subgraphs for $k'-1$ and can be such that $F' - F'^1 \cong G' - G'^1$ in order to ensure that the condition holds at the next round. Similarly, if Spoiler chooses to pick a vertex in F_S, the subgraph of F' consisting of the noise, a response can be given by Duplicator, since $F_S \cong G_S$ and subgraphs isomorphic to F'^1 and G'^1, with $F' - F'^1 \cong G' - G'^1$, can be found. If Spoiler chooses a vertex c in C, then there are subgraphs C_1 and C_2 of C, such that

$C = C_1 \oplus_c C_2$, and at least one of C_1 and C_2 has a subgraph isomorphic to $C_{f_{k'-1},N}$. Suppose C_2 has such a subgraph. Therefore, Duplicator can respond with a vertex d in D, such that $D = D_1 \oplus_d D_2$, for some subgraphs D_1, D_2 of D, where $C_1 \cong D_1$. Then subgraphs isomorphic to F'^1 and G'^1 can be defined, similarly to the cases above, such that the condition holds in the next round.

We now proceed to the more interesting case where Spoiler decides to make a splitting move. A colouring move by Spoiler is considered in cases depending on how he colours the subgraph F' or G'. The argument is similar for whichever graph Spoiler chooses to colour, and therefore we assume that he colours G_i, as in this case the argument is slightly more interesting. Each edge of the graphs is coloured either black or white, and therefore, for each noise-string of length h, in either F_i or G_i, there are at most 2^h ways of how its edges are coloured. Since at each round there are more than $\lambda \cdot 2^{3sk'}$ noise-strings of length h, there must be a colouring that is repeated on many noise-strings of length h. We call the most frequently occurring colouring c (or any one of them if there is a tie), the *primary colouring* of the noise-strings of length h. Notice that at any round and any h, a primary colouring colours more than λ noise-strings of length h identically. In what follows, after F_i (respectively G_i) has been split into two subgraphs by Spoiler or Duplicator in the colouring move, a *component* is a maximally connected component of F_i (respectively G_i) in one of the two subgraphs. Similarly a *string-component* is a component that in addition is a string. A subgraph of F_i or G_i is said to be coloured entirely in black (respectively white) in a colouring move, if all the edges of the subgraph are coloured in black (respectively white).

We give an outline of the main procedure that Duplicator applies as a reply to Spoiler's moves in many of the cases below. Since (2) holds at the current stage we know that $(G', c_1, c_2) \cong (C_{f_{k'+2},N}, r, t) \oplus \bigoplus_{i=2s+1}^{3s} (\lambda \cdot 2^{3sk'} S_i)$.

Note that Duplicator's reply is composed of a colouring of C, F_S and $\overline{F'}$. For F_S and $\overline{F'}$, such a colouring is given by the strategy in the GL game on F_S and G_S, and on $\overline{F'}$ and $\overline{G'}$ respectively, since $(\overline{F'}, \bar{a}, c_1, c_2) \equiv_{k'}^{\mathrm{MSO}} (\overline{G'}, \bar{b}, d_1, d_2)$ and $F_S \cong G_S$. By Lemma 1.2.5, it is sufficient for Duplicator to provide a colouring of C as a response to the colouring of D. Duplicator defines vertices c_3, c_4 in C and d_3, d_4 in D such that, for some C_1, C_2, C_3 and D_1, D_2, D_3:

$$\begin{aligned} (C, c_1, c_2) &= (C_1, c_1, c_3) \oplus_{c_3} (C_2, c_3, c_4) \oplus_{c_4} (C_3, c_4, c_2), \\ (D, d_1, d_2) &= (D_1, d_1, d_3) \oplus_{d_3} (D_2, d_3, d_4) \oplus_{d_4} (D_3, d_4, d_2), \end{aligned}$$

and such that $(C_1, c_1, c_3) \equiv_k^{\mathrm{MSO}} (D_1, d_1, d_3)$ and $(C_3, c_4, c_2) \equiv_k^{\mathrm{MSO}} (D_3, d_4, d_2)$. Furthermore, Duplicator ensures the following for C_2 and D_2. Given a choice of vertices d_3 and d_4 in D, then either the black and white components of D_1 and D_3 are disconnected from the ones in D_2, or not. In the first case, Duplicator ensures the same for the vertices c_3 and c_4, and also makes sure that all the black and white components of D_2 appear in equal numbers up to $\equiv_k^{\mathrm{MSO}}$-equivalence in C_2, and that C_2 has additional black and white components, all of which appear more than λ times up to $\equiv_k^{\mathrm{MSO}}$-equivalence in D_1, D_3 and G_S, and thus they also appear enough times in C_1, C_3 and F_S, as described in Lemma 2.2.2, by definition. The resulting white (respectively

black) subgraphs of F' and G' are in this case $\equiv^{\mathrm{MSO}}_{(k-1)}$-equivalent.

In the second case, Duplicator ensures that the spine of C_2 is $\equiv^{\mathrm{MSO}}_{k}$-equivalent to the whole of D_2, and the teeth of C_2 are split into white and black components that appear enough times up to $\equiv^{\mathrm{MSO}}_{k}$-equivalence in C_1, C_3 and F_S, as described by Lemma 2.2.2, and the splitting is such that the resulting components from the teeth of C_2 are disconnected from the spine of C_2. Again, the resulting white (respectively black) subgraphs of F' and G' are $\equiv^{\mathrm{MSO}}_{(k-1)}$-equivalent.

We call the vertices c_3, c_4 of C and d_3, d_4 of D, the *bordering* vertices of C and D respectively. The structures C_2 and D_2 which are of more importance in the strategy are called *middle* structures of C and D, and finally, the structures C_1, C_3 and D_1, D_3 are called the *surrounding* structures of C and D. We refer to this procedure applied by Duplicator, as the *main procedure*. Finally, using concatenation the above equations may be written as:

$$\begin{aligned} C &= C_1 \cdot C_2 \cdot C_3, \\ D &= D_1 \cdot D_2 \cdot D_3, \end{aligned}$$

or as

$$\begin{aligned} C &= [C]^{c_1}_{c_3} \cdot [C]^{c_3}_{c_4} \cdot [C]^{c_4}_{c_2}, \\ D &= [D]^{d_1}_{d_3} \cdot [D]^{d_3}_{d_4} \cdot [D]^{d_4}_{d_2}. \end{aligned}$$

We split the argument according to the strategy Spoiler follows in colouring the graph G_i, and in particular its subgraph D.

Case 1: Spoiler chooses to colour in black some substring of a segment in D, such that this substring is longer than s edges.
Case 1.1: One of the following holds:

(a) For all $h \in \{2s+1, \ldots, 3s\}$, the h-noise-strings are primarily coloured entirely white by Spoiler in G'.

(b) There exist $h, h' \in \{2s+1, \ldots, 3s\}$, such that the h-noise-strings are primarily coloured entirely white by Spoiler in G', and the h'-noise-strings are primarily coloured black.

(c) There exists $h \in \{2s+1, \ldots, 3s\}$, such that the primary colouring of the h-noise-strings uses both colours.

Suppose there exist vertices x_1, x_2 inside the same segment of the j-th block of D, for some $j \in \{1, \ldots, f_{k'} + 2\}$, such that the string w from x_1 to x_2 is coloured in black, and such that $|w| > s$. The strategy of Duplicator is as follows. Duplicator applies the main procedure discussed above and she defines x_1 and x_2 to be the bordering vertices of D. Then $D = [D]^{d_1}_{x_1} \cdot [D]^{x_1}_{x_2} \cdot [D]^{x_2}_{d_2}$.

By the definition of $f_{k'}$, $[D]^{d_1}_{x_1}$ or $[D]^{x_2}_{d_2}$ has a subgraph isomorphic to $C_{f_{(k'-1)}+2,N}$. Without loss of generality, assume this is so for $[D]^{x_2}_{d_2}$. Duplicator then defines the bordering vertices of C as follows. She picks a vertex y_1 inside the j-th block, at the same distance from the tooth preceding it as x_1 is from the tooth preceding it inside the j-th block of D. Duplicator picks

the second bordering vertex y_2 to be the vertex at the $(j+2l)$-th block of C, at a distance from the preceding tooth, as x_2 is from its preceding tooth in the j-th block of D. Then it holds that $[C]^{c_1}_{y_1} \cong [D]^{d_1}_{x_1}$ and $[C]^{y_2}_{c_2} \equiv^{\text{MSO}}_k [D]^{x_2}_{d_2}$. The latter is because $[C]^{y_2}_{c_2}$ has simply $2l+2$ less teeth than $[D]^{x_2}_{d_2}$, and therefore follows from Lemma 3.1.2.

Duplicator has to colour $[C]^{y_1}_{y_2}$ in such a way that the connected components used, also appear in the graph G'. Note that $[D]^{x_1}_{x_2}$ is isomorphic to a string of length $|w|$. The spine of $[C]^{y_1}_{y_2}$ is isomorphic to a string of length $2l \cdot N + |w|$, which according to Lemma 3.1.2, it is $\equiv^{\text{MSO}}_k$-equivalent to a string of length $|w|$, such as $[D]^{x_1}_{x_2}$. Then Duplicator colours in black the spine of $[C]^{y_1}_{y_2}$. According to how the noise-strings are coloured by Spoiler in G', Duplicator hides the existence of the teeth of $[C]^{y_1}_{y_2}$, in the following way.

If (a) holds, more than λ of the $3s$-noise-strings are coloured entirely white. Therefore, colouring all the teeth in white as well, makes the two resulting subgraphs $\equiv^{\text{MSO}}_{(k'-1)}$-equivalent.

If (b) holds, the h-noise-strings are primarily coloured entirely white and the h'-noise-strings are primarily coloured black. Therefore, there exist h_w and h_b less than or equal to $2s$, such that $S_{h_w} \equiv^{\text{MSO}}_k S_h$ and $S_{h_b} \equiv^{\text{MSO}}_k S_{h'}$. Since $(h_w + h_b) \mid N$, Duplicator can split the teeth of $[C]^{y_1}_{y_2}$ into strings whose lengths alternate between h and h' up to $\equiv^{\text{MSO}}_k$-equivalence, starting with h. The resulting black and white subgraphs are $\equiv^{\text{MSO}}_{(k'-1)}$-equivalent.

Finally if (c) holds, the h-noise-strings are primarily coloured using both colours. Then in each h-noise-string, there are substring-components of length h_1 and h_2 coloured in white and black respectively. There exist h_w and h_b less than or equal to $2s$, such that $S_{h_w} \equiv^{\text{MSO}}_k S_{h_1}$ and $S_{h_b} \equiv^{\text{MSO}}_k S_{h_2}$. Since $(h_w + h_b) \mid N$, the teeth of $[C]^{y_1}_{y_2}$ can be split up into strings whose lengths alternate between h_1 and h_2 up to $\equiv^{\text{MSO}}_k$-equivalence. Therefore, Duplicator can colour the teeth using strings that are $\equiv^{\text{MSO}}_k$-equivalent to the h_1-noise-strings and h_2-noise-strings respectively, starting at the top with the white coloured h_1. The resulting subgraphs are $\equiv^{\text{MSO}}_{(k'-1)}$-equivalent.

Case 1.2: For all $h \in \{2s+1, \ldots, 3s\}$, the h-noise-strings are primarily coloured black.
Case 1.2.1: Spoiler chooses to colour in white some part of a segment in D, that is larger than s edges.

This case is analogous to Case 1.1 above, replacing white with black and vice versa.

Case 1.2.2: No substring of a segment, longer than s edges is coloured white in D by Spoiler.

This case is split further into the following subcases.

Case 1.2.2.1: Spoiler colours D in such a way that there are less than or equal to $8 \cdot \lambda \cdot s^2 \cdot N$ blocks in D that have both edges coloured in white and edges coloured in black.

Let $\nu = 8 \cdot \lambda \cdot s^2 \cdot N$. Let $B_1, \ldots, B_{\nu'}$, for $\nu' \leq \nu$, be the blocks in D that are coloured using both colours. Let also $D_0, \ldots, D_{\nu'}$ be such that for each $i \in \{1, \ldots, \nu'\}$, D_i is the subgraph between B_i and B_{i+1}, and D_0 is before B_1. In other words, let the following hold:

$$D = D_0 \cdot B_1 \cdot D_1 \cdot B_2 \cdot \ldots \cdot B_{\nu'} \cdot D_{\nu'}.$$

Then, by definition of $f_{k'}$, at least one of the D_i has a subgraph isomorphic to the comb $C_{f_{(k'-1)}+2,N}$. This is because, even if the B_i are spread out as much as possible from each other, and even if $\nu' = \nu$, there is i such that D_i comprises at least $(f_{k'} + 2 - \nu)/(\nu + 1)$ many blocks. But $(f_{k'} + 2 - \nu)/(\nu + 1) > f_{(k'-1)} + 2$. Fix $i' \in \{0, \ldots, \nu'\}$, such that $D_{i'}$ contains a subgraph isomorphic to $C_{f_{(k'-1)}+2,N}$.

Each D_j for $j \in \{0, \ldots, \nu'\}$, is coloured entirely in black, as the blocks $B_1, \ldots, B_{\nu'}$ are the only ones coloured using both colours, and because D_j cannot be coloured entirely in white, since there is no substring of a segment larger than s edges coloured in white, by assumption. Duplicator's strategy then is as follows. Let D_C be a subgraph of $D_{i'}$ that is isomorphic to the comb $C_{f_{(k'-1)}+2,N}$, and let x_1, x_2, y_1 and y_2 be the vertices such that

$$D = [D]^{d_1}_{x_1} \cdot [D]^{x_1}_{x_2} \cdot [D]^{x_2}_{d_2},$$

$$C = [C]^{c_1}_{y_1} \cdot [C]^{y_1}_{y_2} \cdot [C]^{y_2}_{c_2},$$

where $[C]^{c_1}_{y_1} \cong [D]^{d_1}_{x_1}$, $[C]^{y_2}_{c_2} \cong [D]^{x_2}_{d_2}$ and $D_C = [D]^{x_1}_{x_2}$. We define C_C to be the subgraph $[C]^{y_1}_{y_2}$. By definition of the above, (C_C, y_1, y_2) is isomorphic to $(C_{f_{(k'-1)},N}, r, t)$. By assumption, there are at least $\lambda \cdot 2^{3s(k'-1)}$ copies of S_h for each $h \in \{2s + 1, \ldots, 3s\}$, that have been coloured in black.

If Duplicator colours entirely in black the subgraph C_C, then according to the $\equiv^{\mathrm{MSO}}_{k'}$-equivalence for the rest of F_i, the white subgraph of F_i is $\equiv^{\mathrm{MSO}}_{(k'-1)}$-equivalent to the white subgraph of G_i, and the black subgraphs of both maintain the condition (2).

Case 1.2.2.2: Spoiler colours more than $8 \cdot \lambda \cdot s^2 \cdot N$ blocks using both colours, and there exist more than λ white string-components of some size β up to $\equiv^{\mathrm{MSO}}_{k}$-equivalence in the colouring of D.

By assumption there is a substring w in a segment of D, with endpoints x_1, x_2, such that w is coloured in black, and $|w| > s$. Let x_1 and x_2 be in the segment of the j-th block. Duplicator follows the main procedure, and defines the middle structure $[D]^{x_1}_{x_2}$ of D, using the bordering vertices x_1 and x_2. The argument is similar for whether $[D]^{d_1}_{x_1}$ or $[D]^{x_2}_{d_2}$ has a subgraph isomorphic to $C_{f_{(k'-1)}+2,N}$, so let this be $[D]^{x_2}_{d_2}$. Then Duplicator defines y_1 to be the vertex on the j-th block of C, with equal distance from the tooth before it, as x_1 is from the tooth before it in D. Similarly, she defines y_2 to be the vertex on the $(j+2l)$-th block, with the same distance from the tooth before it as x_2 is from its preceding tooth. Therefore $[C]^{c_1}_{y_1} \cong [D]^{d_1}_{x_1}$ and $[C]^{y_2}_{c_2} \equiv^{\mathrm{MSO}}_{k} [D]^{x_2}_{d_2}$ according to Lemma 3.1.2.

By definition of the vertices x_1, x_2, y_1 and y_2, it holds that the spine of $[C]^{y_1}_{y_2}$ is isomorphic to $S_{|w|+2lN}$ and it is hence $\equiv^{\mathrm{MSO}}_{k}$-equivalent to the substring w of D, by Lemma 3.1.2. Duplicator colours in black the spine of $[C]^{y_1}_{y_2}$. By assumption, there is a white string-component that appears more than λ times in the colouring of D, up to $\equiv^{\mathrm{MSO}}_{k}$-equivalence. Notice, that since $[D]^{x_1}_{x_2}$ is a string coloured in black, it follows that this white string-component appears more than λ times in the colouring of $[D]^{d_1}_{x_1}$ and $[D]^{x_2}_{d_2}$. Thus, such a white string-component appears enough

times, up to $\equiv_k^{\mathrm{MSO}}$-equivalence, as described by Lemma 2.2.2, in the colouring of $[C]_{y_1}^{c_1}$ and $[C]_{c_2}^{y_2}$. To colour the teeth attached to the spine of $[C]_{y_2}^{y_1}$, Duplicator uses the white string-component above, together with the black h_1-noise-strings, for some $h_1 \in \{2s+1, \ldots, 3s\}$, ensuring that the resulting black components of the teeth are not connected to the spine of $[C]_{y_2}^{y_1}$. In this way, the resulting white and black subgraphs of F_i and G_i are $\equiv_{(k'-1)}^{\mathrm{MSO}}$-equivalent.

Case 1.2.2.3: Spoiler colours more than $8 \cdot \lambda \cdot s^2 \cdot N$ blocks using both colours, but there exists no white string-component that appears more than λ times up to $\equiv_k^{\mathrm{MSO}}$-equivalence, in the colouring of D.

Consider a vertex d that is a fork in D, such that the component d is on, is coloured white in D. This component has in general the shape of the letter 'T'. In other words, it can comprise parts of the two segments to which d is attached together with part of the tooth to which it is attached. Components of this form will be referred to as T-shaped components. Notice that the isomorphism type of the T-shaped components is determined by the length of the portion of the segments and the tooth included, and since, by assumption, there cannot be a white substring that is part of a segment and larger than s edges, and since the tooth has N edges, there are at most s^2N isomorphism types for the possible T-shaped components coloured entirely white in the colouring of D.

There are at most $2s$ strings up to $\equiv_k^{\mathrm{MSO}}$-equivalence, and therefore, by assumption that no string-component appears more than λ times in the colouring of D up to $\equiv_k^{\mathrm{MSO}}$-equivalence, there can be at most $2 \cdot \lambda \cdot s$ white string-components in the colouring of D, including the string-components that may appear in the colouring of the teeth of D. Therefore there are at most $4 \cdot \lambda \cdot s$ blocks that contain part of those white string-components. Hence, at least $8 \cdot \lambda \cdot s^2 \cdot N - 4 \cdot \lambda \cdot s > 4 \cdot \lambda \cdot s^2 \cdot N$ blocks will contain part of a T-shaped component coloured white.

Each such T-shaped white component lies in at most 2 blocks, and therefore there are at least $2 \cdot \lambda \cdot s^2 \cdot N$ white T-shaped components, and since there are at most $s^2 \cdot N$ isomorphism types for the possible white T-shaped components, there must be an isomorphism type that appears at least λ times in the colouring of D. Let this type be τ and let j be such that a white T-shaped component of type τ is at the junction between the j-th and $(j+1)$-th blocks. Let x_1 be the endpoint of this component on the segment of the j-th block and similarly let x_2 be the one on the $(j+1)$-th block. Let x_3 be finally the endpoint of this component that is in the tooth of D. It holds that $D = [D]_{x_1}^{d_1} \cdot [D]_{x_2}^{x_1} \cdot [D]_{d_2}^{x_2}$, where $[D]_{x_2}^{x_1} \cong T_\tau \oplus_{x_3} S_z$ for some $z \leq N$ and with T_τ being the T-shaped component of isomorphism type τ. Then, $[D]_{x_1}^{d_1}$ or $[D]_{d_2}^{x_2}$ has a subgraph isomorphic to $C_{f_{(k'-1)}+2,N}$. Without loss of generality, let this be $[D]_{d_2}^{x_2}$.

Duplicator then replies as follows. She defines y_1 to be the vertex on the j-th block of C, that is of equal distance from the tooth after it, as x_1 is from the tooth after it in D. Similarly she defines y_2 to be the vertex in the $(j+1+2l)$-th block, at the same distance from the tooth before it, as x_2 is from its preceding tooth in the $(j+1)$-th block of D. It holds that $C = [C]_{y_1}^{c_1} \cdot [C]_{y_2}^{y_1} \cdot [C]_{c_2}^{y_2}$, where $[C]_{y_1}^{c_1} \cong [D]_{x_1}^{d_1}$ and $[C]_{c_2}^{y_2} \equiv_k^{\mathrm{MSO}} [D]_{d_2}^{x_2}$ by Lemma 3.1.2. Duplicator

therefore has a reply to the colouring of $[D]^{d_1}_{x_1}$ and $[D]^{x_2}_{d_2}$. For the colouring of $[C]^{y_1}_{y_2}$, Duplicator colours in white all the junctions between the blocks that are in $[C]^{y_1}_{y_2}$, creating white T-shaped components of type τ. Furthermore, she colours in black all the string-components between them as well as all the string-components that remain at the teeth. By assumption, the white T-shaped components of isomorphism type τ appear more than λ times in the colouring of D, and therefore more than or equal to λ times in the colouring of $[D]^{d_1}_{x_1}$ and $[D]^{x_2}_{d_2}$. Thus, such white components appear enough times also in the colouring of $[C]^{c_1}_{y_1}$ and $[C]^{y_2}_{c_2}$, as described by Lemma 2.2.2. The black string components used in the colouring of the remaining subgraph of $[C]^{y_1}_{y_2}$, are each $\equiv^{\text{MSO}}_k$-equivalent to the h_1-noise-strings for some $h_1 \in \{2s+1, \ldots, 3s\}$, which are all primarily coloured in black by assumption. This is because, even if the string remaining below the T-shaped component of type τ, is of length smaller than or equal to s, Dulicator can colour in white $2l$ of the $2l+1$ junctions in a way such that the resulting white T-shaped components are $\equiv^{\text{MSO}}_k$-equivalent to the one of isomorphism type τ, but having a shorter string as part of the tooth. In this way, the strings remaining below these components are longer than s edges and therefore $\equiv^{\text{MSO}}_k$-equivalent to the h_2-noise-strings for some $h_2 \in \{2s+1, \ldots, 3s\}$. The resulting white and black subgraphs of F_i and G_i are $\equiv^{\text{MSO}}_{(k'-1)}$-equivalent.

These subcases exhaust all possible ones for Case 1.

Case 2: Spoiler chooses to colour in white some substring of a segment in D, such that this substring is larger than s edges.

This case is similar to Case 1, after replacing white with black and vice versa.

Case 3: Spoiler chooses to colour D in such a way that there exists no substring of a segment longer than s edges coloured completely in black or completely in white.

The colouring of any segment of D, switches from black to white or vice versa at least $N/s-1$ times. Let a *3-block* be 3 consecutive blocks of a comb and consider the structure D as a series of connected 3-blocks as shown below and in Figure 3.3.

$$(D, d_1, d_2) \cong (V_1, d_1, t_1) \oplus_{t_1} (V_2, t_1, t_2) \oplus_{t_2} \ldots \oplus_{t_v} (V_v, t_{v-1}, t_v) \oplus_{t_v} (R, t_v, d_2),$$

where R is the structure remaining after considering the blocks before it, in triples. The vertices $t_1, \ldots, t_v$ are the forks in D on which the different 3-blocks meet. In the above, $v = f_{k'}/3$. Each 3-block V_i, for $i \in \{1, \ldots, v\}$, has $6 \cdot N$ edges, and therefore there are at most $2^{6 \cdot N}$ possible different colourings of 3-blocks. For all $k' \geq 1$, $f_{k'} \geq 6\lambda n \cdot 2^{6 \cdot N}$, and therefore there are at least $\mu = 2\lambda n$ 3-blocks from the set $\{V_1, \ldots, V_v\}$ that have been coloured identically. Let $\{V_{c_1}, \ldots, V_{c_\mu}\}$ be a subset of $\{V_1, \ldots, V_v\}$ of size μ containing 3-blocks that have been coloured using the same colouring c. Let W be the set of all white string-components up to isomorphism that appear in the 3 segments of the 3-blocks V_{c_i}. Similarly let B be the set of black string-components up to isomorphism that appear in these segments. Notice that W and B contain only white or black string-components appearing in the segments of the blocks, and therefore for any $w_1 \in W$ and

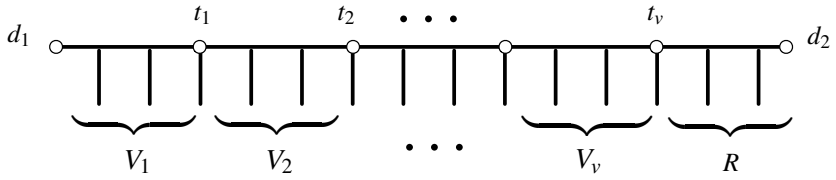


Figure 3.3: D as a series of connected 3-blocks.

$b_1 \in B$, $|w_1| \leq s$ and $|b_1| \leq s$. Notice also for each $w_1 \in W$ and $b_1 \in B$ there are at least $2\lambda n$ white and black string-components in the colouring of D, that are $\equiv_k^{\mathrm{MSO}}$-equivalent to w_1 and b_1 respectively. We define $w \in W$ to be the longest string component in W. Similarly for $b \in B$.

Fix $V_I \in \{V_{c_1}, \ldots, V_{c_\mu}\}$ comprising the j-th, $(j+1)$-th and $(j+2)$-th blocks of D for some $j \in \{1, \ldots, f_{k'} - 2\}$, and let V^1, V^2 and V^3 be these blocks respectively. Fix V_I to also be more than n blocks away from both 3-blocks V_1 and V_v. Notice that such a 3-block always exists since $\mu = 2 \cdot \lambda \cdot n$. Let $t_{I_1}, t_{I_2} \in \{t_1, \ldots, t_v\}$ be the endpoints of the spine of V_I so that for some structures D_1, D_3 it holds that

$$(D, d_1, d_2) \cong (D_1, d_1, t_{I_1}) \oplus_{t_{I_1}} (V_I, t_{I_1}, t_{I_2}) \oplus_{t_{I_2}} (D_3, t_{I_2}, d_2).$$

The structure D_1 or the structure D_3 has a subgraph isomorphic to $C_{f_{(k'-1)}+2,N}$. Let this be D_3, as the argument is similar for either case. Suppose the colouring of the segment in V^1 switches from white to black ν_1 times, and suppose similarly that the number of times is ν_2 for V^2 and ν_3 for V^3. Let then $\{p_1, \ldots, p_{\nu_1}\}$ be the set of points where the colour switches from white to black in the segment of V^1. The sets $\{q_1, \ldots, q_{\nu_2}\}$ and $\{r_1, \ldots, r_{\nu_3}\}$ are defined similarly for V^2 and V^3 respectively. Since $(|w| + |b|) \mid N$, the strings w and b can be used to colour a string of length N. Let x_1 and x_2 be the top of the two teeth, in the middle of the 3-block V_I, namely the endpoints of the spine of V^2. It holds that

$$(V_I, t_{I_1}, t_{I_2}) = (V^1, t_{I_1}, x_1) \oplus_{x_1} (V^2, x_1, x_2) \oplus_{x_2} (V^3, x_2, t_{I_2}).$$

Assume there exists a vertex p_j for $1 \leq j \leq \nu_1$ such that colouring the next N edges that follow using w and b, the edges of the spine adjacent to x_1 are of the same colour. Then Duplicator replies to Spoiler's move as follows. She defines the vertex p' in the segment of the j-th block of C and the vertex p'' in the $(j+2l)$-th block of C to be such that both of them are at the same distance from the corresponding tooth after them as p_j is from x_1. Let C_1, C_2, C_3 be such that

$$(C, c_1, c_2) = (C_1, c_1, p') \oplus_{p'} (C_2, p', p'') \oplus_{p''} (C_3, p'', c_2).$$

Then $C_1 = [C]_{p'}^{c_1} \cong [D]_{p_j}^{d_1}$ and $C_3 = [C]_{c_2}^{p''} \equiv_k^{\mathrm{MSO}} [D]_{d_2}^{p_j}$ by Lemma 3.1.2. Duplicator's response for C is composed of a colouring of C_1, C_2 and C_3, and therefore for C_1 and C_3 this is given by the strategy she has for the game on $[C]_{p'}^{c_1}$ and $[D]_{p_j}^{d_1}$ and also for the game on $[C]_{c_2}^{p''}$ and $[D]_{d_2}^{p_j}$. By definition, the colour switches from white to black in D at the vertex p_j. The substring u of the spine of C, which is the spine of C_2, with endpoints p' and p'' is $2l \cdot N$ edges and therefore

can be coloured using the substrings b and w, starting with b. According to the assumption, after colouring the string u using b and w, the edges of the spine adjacent to the vertices that are at the top of teeth in u, namely the forks in C_2, are coloured the same. Duplicator then colours the teeth attached to u also using b and w, colouring the top with the opposite colour of what the edges of the spine, attached at the top of the tooth, are coloured with. The resulting white and black substructures of F_i and G_i are $\equiv^{\mathrm{MSO}}_{(k'-1)}$-equivalent. A similar argument holds for any point $q_{j'} \in \{q_1, \ldots, q_{\nu_2}\}$ or $r_{j''} \in \{r_1, \ldots, r_{\nu_3}\}$.

Assume that for every $p_j, q_{j'}$ and $r_{j''}$, this is not the case, and therefore, colouring the next N edges of any such vertex using w and b, causes one of the edges adjacent to the top of the next tooth to be coloured in black and the other one in white. We will show that if this is the case, then Duplicator can simulate the colouring of a 3-block inside a single block or inside a 2-block, depending on the following.

Let $p_I \in \{p_1, \ldots, p_{\nu_1}\}$ be the vertex in the segment of V^1, that is further than $2 \cdot s + 1$ edges away from, but as close as possible to, x_1, the top of the tooth of V^1. Let the case be similar for $q_J \in \{q_1, \ldots, q_{\nu_2}\}$ and $r_K \in \{r_1, \ldots, r_{\nu_3}\}$, for x_2 and t_{I_2} respectively, the top vertices of the respective teeth after them. By assumption, the distance between p_I and x_1 is equal to either $z \cdot (|b| + |w|)$ or $z' \cdot (|b| + |w|) + |b|$ for some $z, z' \in \mathbb{N}$. In other words colouring the substring with endpoints p_I and x_1, starting with the black substring b, will either result in finishing the colouring with b or with w. So is the case for the distances of q_J from x_2 and of r_K from t_{I_2}. Therefore for at least two substrings from the ones starting at p_I, q_J or r_K, and ending at the corresponding teeth after them, will result in finishing the colouring with black or with white. The cases are symmetrical, therefore assume that for two such substrings the colouring finishes using the white substring w, and are hence of length $z_1 \cdot (|b|+|w|)$ and $z_2 \cdot (|b|+|w|)$, respectively, for some $z_1, z_2 \in \mathbb{N}$. For which pair of substrings this is the case, is similar, but it is slightly more interesting for the ones starting at the vertices p_I and r_K.

It will be shown that the lengths of these two substrings are actually equal. Let the substring starting at the vertex p_I be of length $z_1 \cdot (|b| + |w|)$ and that of the substring starting at r_K be of length $z_2 \cdot (|b| + |w|)$ for some $z_1, z_2 \in \mathbb{N}$. Suppose that $z_2 > z_1$. Let b_r be the black string-component adjacent to r_K that is part of the substring from r_K to t_{I_2}, and let w_r be the white string-component that is in the same substring and is adjacent to the black string-component b_r. Let w_r have endpoints r_1 and r_2, with r_1 being the vertex common to both b_r and w_r. Then, r_2 is less than or equal to $2 \cdot s + 1$ edges away from the vertex t_{I_2}, since otherwise r_2 would have been selected instead of r_K, and therefore

$$z_2 \cdot (|b| + |w|) - (|b_r| + |w_r|) \leq 2 \cdot s + 1. \tag{3.1}$$

Let $z_{1,2}$ be the difference in the sizes of the two substrings, namely $z_{1,2} = (z_2 - z_1) \cdot (|b| + |w|)$. Also, since both p_I and r_K are further than $2 \cdot s + 1$ edges away from their respective next teeth, it holds that

$$z_2 \cdot (|b| + |w|) - z_{1,2} > 2 \cdot s + 1. \tag{3.2}$$

Therefore, from (3.1) and (3.2), $(|b_r| + |w_r|) > z_{1,2} = (z_2 - z_1) \cdot (|b| + |w|)$. But w and b are defined to be the largest white and black string-components in W and B respectively, where W and B are the sets of white and black string-components respectively, appearing in the segments of V^1, V^2 and V^3. Hence $|w| \geq |w_r|$ and $|b| \geq |b_r|$, which is a contradiction. Therefore $z_1 \geq z_2$. A similar contradiction can be shown for $z_1 > z_2$, and hence it is the case that $z_1 = z_2$. This means that there are two vertices p_I and r_K, one in the segment of V^1 and the other in the segment of V^3, which have equal distances from the respective teeth in V^1 and V^3, and where the colour switches from white to black in the colouring given by Spoiler.

In this case Duplicator responds as follows. Let C_2 be the j-th block of C, with endpoints y_1 and y_2, and let p be the vertex in C_2 whose distance from y_2 is equal to the distance of p_I from x_1 and of r_K from t_{I_2}. Let $C_1 = [C]^{c_1}_p$ and $C_3 = [C]^p_{c_2}$ such that $C = C_1 \cdot C_3$. Let D_1, D_2 and D_3 be such that $D_1 = [D]^{d_1}_{p_I}$, $D_2 = [D]^{p_I}_{r_K}$, $D_3 = [D]^{r_K}_{d_2}$ and $D = D_1 \cdot D_2 \cdot D_3$. Then it holds that $(C_1, c_1, p) \cong (D_1, d_1, p_I)$ and $(C_3, p, c_2) \cong (D_3, r_K, d_2)$, and the colour switches in D from white to black at both p_I and r_K. Duplicator colours C_1 and C_3 according to the colourings of D_1 and D_3. Since the colouring of V_I appears more than $\mu = 2\lambda n$ times, all components in D_2 appear more than λ times in the colouring of D_1 and D_3, and hence also appear enough times in the colouring of C. Therefore, Duplicator has a reply and the resulting white and black substructures of F_i and G_i are $\equiv^{\mathrm{MSO}}_{(k'-1)}$-equivalent.

As was stated above, Duplicator can simulate the colouring of a 3-block inside a single block. In the case where p_I and q_J have the same distance from their respective next teeth, or where this is the case for q_J and r_K, Duplicator would be able to simulate the colouring of a 3-block coloured with c, inside a 2-block. Then she has to apply the same method on two different 3-blocks of D, and their respective replies in C.

Even more interesting is when Spoiler makes a colouring move on the graph F_i instead of G_i, for this last case. So assume that there exists a colouring c of 3-blocks, such that there are vertices p_I and r_K, with the properties mentioned above. It is known that there are at least $2 \cdot \lambda \cdot n$ such 3-blocks and therefore at least $2l$ such 3-blocks. Let then, $V^C_1, \ldots, V^C_l$ be l 3-blocks coloured using c in C. Let $C_1, \ldots, C_{l+1}$, and $x_1, \ldots, x_{2l}$ be such that the following holds.

$$\begin{aligned}(C, c_1, c_2) \quad = \quad & (C_1, c_1, x_1) \oplus_{x_1} (V^C_1, x_1, x_2) \oplus_{x_2} \ldots \\ & \ldots \oplus_{x_{2j}} (C_{j+1}, x_{2j}, x_{2j+1}) \oplus_{x_{2j+1}} (V^C_{j+1}, x_{2j+1}, x_{2(j+1)}) \oplus_{x_{2(j+1)}} \cdots \\ & \ldots \oplus_{x_{2l-1}} (V^C_l, x_{2l-1}, x_{2l}) \oplus_{x_{2l}} (C_{l+1}, x_{2l}, c_2).\end{aligned}$$

Then, because of the size of C, for at least one $j' \in \{1, \ldots, l+1\}$, $C_{j'}$ has a subgraph isomorphic to $C_{f_{(k'-1)},N}$. Assume that it holds for some fixed $j' = j_0$. For each j the vertices p^j_I, q^j_J and r^j_K are the vertices with the above properties, that are inside the V^C_j 3-block. Duplicator replies according to the following. She chooses subgraphs $D_1, \ldots, D_{l+1}$, single blocks $V^D_1, \ldots, V^D_l$ and vertices $y_1, \ldots, y_{2l}$ such that

$$\begin{aligned}(D, d_1, d_2) \quad = \quad & (D_1, d_1, y_1) \oplus_{y_1} (V^D_1, y_1, y_2) \oplus_{y_2} \ldots \\ & \ldots \oplus_{y_{2j}} (D_{j+1}, y_{2j}, y_{2j+1}) \oplus_{y_{2j+1}} (V^D_{j+1}, y_{2j+1}, y_{2(j+1)}) \oplus_{y_{2(j+1)}} \cdots \\ & \ldots \oplus_{y_{2l-1}} (V^D_l, y_{2l-1}, y_{2l}) \oplus_{y_{2l}} (D_{l+1}, y_{2l}, d_2).\end{aligned}$$

All structures V_j^D for $j \in \{1, \ldots, l\}$ are single blocks. Furthermore, for j_0, the subgraph D_{j_0} is chosen to be $2l+2$ blocks larger than C_{j_0} and for all $j' \neq j_0$, $D_{j'} \cong C_{j'}$. By Lemma 3.1.2, $C_{j_0} \equiv_k^{\text{MSO}} D_{j_0}$. This is possible because we know that D has 2 more blocks than C, and for all $j' \neq j_0$, $C_{j'}$ and $D_{j'}$ comprise the same number of blocks. Only the sizes of V_j^D and V_j^C for $j \in \{1, \ldots, l\}$ need to be considered together with the difference in size of C_{j_0} and D_{j_0}. Each V_j^C is a 3-block, and each V_j^D is a single block. Therefore $3l$ must be two less than $l + 2l + 2$, which is indeed the case. This way Duplicator has a reply for all D_j and V_j^D in D and the components that appear in each V_j^C that are not used, appear more than $\mu = 2\lambda n$ times in C and therefore enough times in the colouring of D given by the strategy described. The resulting white and black subgraphs of F_i and G_i are $\equiv_{(k'-1)}^{\text{MSO}}$-equivalent.

Similarly, if p_I and q_J or if q_J and r_K are the vertices that are guaranteed to be at equal distances from their respective next teeth in the colouring c, then Duplicator chooses $2l$ 3-blocks in C, with appropriate replies in D, and follows a similar strategy. □

We show that the class of forests having only trees with 1 (mod 3) number of leaves is expressible in MSO. Let F be some forest containing the trees $T_1, \ldots, T_n$. Let X be the set of nodes of some tree T_i in F, and let X_0, X_1, X_2 be a partition of X. Consider also a deterministic binary tree automaton A, with 3 states q_0, q_1, q_2, such that a vertex x is assigned the state q_j, by A if and only if x is the root of a subtree of T_i, with j (mod 3) number of leaves. The formula transition(X, X_0, X_1, X_2) below, holds for any partition of the vertices in X, into X_0, X_1, X_2 which agree respectively with the states q_0, q_1, q_2 of A. In other words, for a vertex x, $x \in X_j$ if and only if x is assigned the state q_j. The formula Tree1mod3(X), given the assumption that X is the set of nodes of a tree T_i in F, expresses that the root of T_i is assigned the state q_1 by A, and therefore has 1 (mod 3) number of leaves.

$$
\begin{aligned}
\text{partition}(X, X_0, X_1, X_2) &= \forall x \in X\ (X_0(x) \vee X_1(x) \vee X_2(x)) \wedge \textstyle\bigwedge_{i \neq j}(X_i \cap X_j = \emptyset),\\
\text{transition}(X, X_0, X_1, X_2) &= \forall x, y, z \in X\ (\text{leaf}(x) \rightarrow X_1(x)) \wedge\\
&\quad \Big(E(x,y) \wedge E(x,z) \rightarrow\\
&\quad \rightarrow \textstyle\bigwedge_{i,j}(X_i(y) \wedge X_j(z) \rightarrow X_{i+j \ (\text{mod } 3)}(x))\Big) \wedge\\
&\quad \wedge \textstyle\bigwedge_i (E(x,z) \wedge \text{one-child}(x) \wedge X_i(z) \rightarrow X_i(x)),\\
\text{Tree1mod3}(X) &= \forall X_0, X_1, X_2 \forall x \in X\ (\text{partition}(X, X_0, X_1, X_2) \wedge\\
&\quad \wedge \text{transition}(X, X_0, X_1, X_2) \wedge \text{root}(x) \rightarrow X_1(x)).
\end{aligned}
$$

The formula Tree(X) is true in some forest F, if and only if X is the set of vertices of one of the trees T_i in F. According to the formulae above, the class of forests that contains only trees with 1 (mod 3) number of leaves is defined by the formula Forest1mod3 below.

$$
\begin{aligned}
\text{Tree}(X) &= \forall x, y\ \big(X(x) \wedge X(y) \rightarrow \exists r\ (\text{ExistsPath}(r, x) \wedge \text{ExistsPath}(r, y))\big) \wedge\\
&\quad \wedge \forall z, w\ (X(w) \wedge \text{ExistsPath}(w, z) \rightarrow X(z)),\\
\text{Forest1mod3} &= \forall X\ (\text{Tree}(X) \rightarrow \text{Tree1mod3}(X)).
\end{aligned}
$$

The following Corollary is a direct consequence of the above and of Theorem 3.1.4.

Corollary 3.1.5. *Over the class of forests* GL *is strictly less expressive than* MSO.

Corollary 3.1.6. *Over the following classes of finite simple graphs,* GL *is strictly less expressive than* MS_2.

1. *The class of finite graphs.*
2. *The class of forests.*
3. *The class of planar graphs.*
4. *The class of bounded-degree graphs.*
5. *The class of graphs of bounded tree-width.*
6. *The class of graphs of bounded clique-width.*

Notice that, by Theorem 1.2.4, GL is strictly less expressive than MS_1 over the classes 2-5 given above.

3.2 FO Interpretations and MSO Transductions

In this section we investigate whether GL is closed under First Order interpretations and MSO transductions. FO interpretations and MSO transductions are mappings from structures of some signature to structures of some other signature. Since GL is defined only on graphs, we restrict our attention to graphs regarding these mappings, but we give the definitions below in terms of structures in general. One reason FO interpretations are useful is because they provide a way of encoding a structure into a graph, and then define properties of the original structure in terms of definable properties of the graph that corresponds to the structure. This idea is captured when a logic is closed under some particular type of interpretations. Closure of a logic under such mappings is in general regarded as a natural property, and both FO and MSO are closed under most of these mappings. We give an introduction to these notions and refer the reader to [Hod97],[Ott97] and [EF99] for further details about FO interpretations, and [Cou08] for MSO transductions.

Definition 3.2.1. *Let σ, τ be two relational signatures, with $\sigma = \langle R_1, \ldots, R_s \rangle$ where each R_i is r_i-ary for some $r_i \in \mathbb{N}$. Let Φ be an $(s+1)$-tuple of* FO *formulae over the signature τ, each one with free variables among the ones displayed below, where $\bar{x}, \bar{x}_1, \ldots, \bar{x}_{r_i}, \ldots$ have all length k.*

$$\Phi = (\Phi_{uni}(\bar{x}), \Phi_1(\bar{x}_1, \ldots, \bar{x}_{r_1}), \ldots, \Phi_s((\bar{x}_1, \ldots, \bar{x}_{r_s}))).$$

Then Φ is an FO*-definable interpretation of σ in τ of width k, written as an* FO*-definable (σ, τ)-interpretation.*

Suppose $\mathfrak{A}$ is a τ-structure. An FO-interpretation Φ of σ in τ defines a σ-structure $\mathfrak{A}^\Phi$ over k-tuples of the elements of $\mathfrak{A}$. In particular the universe of $\mathfrak{A}^\Phi$ is equal to $\{\bar{a} \in A^k \mid \mathfrak{A} \models \Phi_{uni}(\bar{a})\}$ and for each $i \in \{1, \ldots, s\}$, the interpretation of R_i is given by $\{(\bar{a}_1, \ldots, \bar{a}_{r_i}) \in A^{k \cdot r_i} \mid \forall i\ \bar{a}_i \in A^k,\ \mathfrak{A} \models \Phi_i(\bar{a}_1, \ldots, \bar{a}_{r_i}) \wedge \bigwedge_{i=1}^{r_i} \Phi_{uni}(\bar{a}_i)\}$.

In what follows, if R is a r-ary query on STRUC$[\sigma]$, and Φ is a (σ, τ)-interpretation of width k, then $R(\Phi(.))$ maps each structure $\mathfrak{A}$ of STRUC$[\tau]$ to an r-ary query on $\Phi(\mathfrak{A})$.

Definition 3.2.2. *Let σ, τ be relational signatures. Then a logic $\mathcal{L}$ is closed under* FO*-definable (σ, τ)-interpretations if the following holds. If Φ is an* FO*-definable (σ, τ)-interpretation, and R is an $\mathcal{L}$-definable query on* STRUC$[\sigma]$*, then $R(\Phi(.))$ is $\mathcal{L}$-definable.*

As explained in [EF99], MS_1 is not closed under FO-definable (σ, τ)-interpretations of width k, for $k \geq 2$, but it is closed under such interpretations of width 1. On the other hand, on graphs represented as structures with the set of vertices as the universe of the structure and with a single binary relation for edges, MS_2 is not closed under FO-definable interpretations of width $k = 1$, either.

The following example will illustrate how FO interpretations work and why MS_2 is not closed under FO-definable interpretations of width 1. Let us consider the signature $\tau = \langle E \rangle$ of graphs. Let Φ be the FO-definable (τ, τ)-interpretation $\Phi = (\Phi_{uni}(x), \Phi_E(x_1, x_2))$ where

$$\begin{aligned} \Phi_{uni}(x) &= x = x, \\ \Phi_E(x_1, x_2) &= x_1 \neq x_2. \end{aligned}$$

The formula Φ_E expresses that there is an edge between any two vertices different from each other. Let $\mathcal{C}$ be the class of graphs with an even number of vertices. This class of graphs is known to be definable neither in MS_1 nor in MS_2.

Lemma 3.2.3. *Let G be a graph. Then G is in $\mathcal{C}$ if and only if $\Phi(G)$ has a perfect matching.*

Proof. First consider the *only if* direction. Fix $G = (V, E)$ and assume that $G \in \mathcal{C}$. The universe of the interpretation is the same as V. Also there is an edge in the interpretation between any two vertices $v_1, v_2 \in V$ such that $v_1 \neq v_2$. If V is of even size, then the set V can be split into pairs, and the set of edges between the vertices of each pair forms a perfect matching in the interpretation.

Consider then the *if* direction. Fix $G = (V, E)$ and assume that $\Phi(G)$ has a perfect matching. Then the number of vertices in $\Phi(G)$ is even and since this set of vertices is the same as V, V is also of even cardinality. □

Corollary 3.2.4. MS_2 *is not closed under* FO*-definable (σ, τ)-interpretations of any width.*

Proof. For $k \geq 2$, the result follows from the fact that MS_1 is not closed under FO-definable (σ, τ)-interpretations of such width. For $k = 1$, note that the property of a graph having an even number of vertices is not definable in MS_2. A perfect matching on the other hand is definable by saying there is a set of edges S and expressing that each vertex is adjacent to exactly one edge in S. The result follows from Lemma 3.2.3 and Definition 3.2.2. □

It should be noted that the property of a graph having a perfect matching is only definable in MS_2, where the second order quantification is over edges and not only over vertices. In the above, we give an interpretation of width 1 from structures of signature $\sigma_{\lfloor G \rfloor}$ to structures of the same signature, where MS_2 is well-defined. But MS_2 is closed under FO $(\tau_{\lceil G \rceil}, \tau_{\lceil G \rceil})$-interpretations, namely when graphs are represented as structures with a universe containing elements for both the vertices and the edges of the graph. This is because, MS_2 over such graph structures is simply MSO.

In the works of Courcelle, a variation of interpretations of width 1, named MSO transductions is considered ([Cou08]). Both MS_1 and MS_2 are closed under these transductions. The formal definition follows. We show that GL is closed neither under FO transductions (and therefore MSO transductions), nor under FO-definable interpretations of width 1 over incidence graphs.

Monadic second-order transductions are transformations of structures specified by monadic second-order formulae. We consider here a simpler form of the monadic second-order transductions defined in [Cou08], since this simpler form is enough to show that GL is not closed under such transductions, and therefore nor under the generalized ones. For simplicity the definition of MSO transductions below is given for signatures with binary relations only. The definition can be extended to r-ary relations for $r \in \mathbb{N}$.

Definition 3.2.5. *A k-copying monadic second-order transduction from structures of signature σ to structures of signature $\tau = \langle R_1, \ldots, R_n \rangle$, for $k \in \mathbb{N}$, is a mapping $f : \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$ that associates with each structure $\mathfrak{A} \in STRUC[\sigma]$ a structure $\mathfrak{B} \in STRUC[\tau]$ and is specified by the definition scheme $((\delta_i)_{i \leq k}, (\theta_{1;i,j})_{i,j \leq k}, \ldots, (\theta_{n;i,j})_{i,j \leq k})$ as follows. For the universe of the structure $\mathfrak{B}$, $B \subseteq A_1 \times \{1\} \cup \ldots \cup A_k \times \{k\}$ where the sets A_i are defined in $\mathfrak{A}$ by the monadic second-order formulae δ_i, and for each relation R_h of $\mathfrak{B}$, $R_h = \{((v,i),(u,j)) \mid (v,u) \in R_{h;i,j}\}$, where the sets $R_{h;i,j}$ are defined in $\mathfrak{A}$ by the monadic second-order formulae $\theta_{h;i,j}$.*

It should be noted that each copy of $a \in A$ of the structure $\mathfrak{A}$ in the resulting structure $\mathfrak{B}$, is essentially marked with $1, \ldots, k$, and that a first-order transduction is defined similarly, with the difference that all the formulae in the definition scheme are first-order instead of monadic second-order. Notice that a 1-copying FO transduction is essentially the same as an FO interpretation of width 1. In fact, k-copying transductions are generalizations of interpretations of width 1. With a k-copying transduction, the universe of the resulting structure consists of k subsets of the universe of the input structure, as permitted by the formulae $(\delta_i)_{i \leq k}$ in the definition scheme. An interpretation of width k, may increase the size of the resulting universe to $|A|^k$, where A is the universe of the input structure $\mathfrak{A}$. On the other hand, for a k-copying transduction, the size of the resulting structure is always linearly bounded by the size of the input structure, namely $|B| \leq k \cdot |A|$, where A and B are the universes of the input structure $\mathfrak{A}$ and the resulting structure $\mathfrak{B}$ respectively. One of the main reasons why MSO transductions are of interest is the following Theorem, adapted from [Cou08].

Theorem 3.2.6 ([Cou08])**.** *If $L \subseteq \mathrm{STRUC}[\tau]$ is* MSO*-definable, and $f : \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$ is a k-copying* MSO *transduction for some $k \in \mathbb{N}$, then $f^{-1}(L)$ is* MSO*-definable.*

To show that GL is closed neither under FO-definable interpretations of any width k, nor under FO k'-copying transductions for any k', we consider the two signatures $\sigma_{\lfloor G \rfloor}$ and $\tau_{\lceil G \rceil}$ defined in Section 1.1.1. The results mainly follow from Lemma 3.2.7 below and Theorem 3.1.4.

Lemma 3.2.7. *Let C be a comb. Then the property of C having* 1 (*mod* 3) *number of leaves is* GL *definable.*

Proof. In the absence of noise-strings, one can express in GL that a comb has 1 (mod 3) number of leaves. It can be expressed using the following formula ϕ. The formula ψ_{comb} expresses that a graph is a comb.

$$\begin{aligned}
\phi &= \psi_{comb} \wedge (\text{SetOf3Blocks} \mid \text{SetOf3Blocks}),\\
\psi_{comb} &= \forall x, y \Big(\text{fork}(x) \wedge \text{fork}(y) \rightarrow \big((\text{Path}(x,y) \mid \top) \vee (\text{Path}(y,x) \mid \top)\big)\Big),\\
\text{forks}(x_1,x_2,x_3) &= \big(\textstyle\bigwedge_{i\neq j}(x_i \neq x_j)\big) \wedge \text{fork}(x_1) \wedge \text{fork}(x_2) \wedge \text{fork}(x_3),\\
\text{SetOf3Blocks} &= \forall x\ \text{root}(x) \rightarrow \exists! x_1, x_2 \left(\text{forks}(x,x_1,x_2) \wedge \left(\begin{array}{l}(\text{Path}(x,x_1) \mid \top)\\ \wedge(\text{Path}(x_1,x_2) \mid \top)\end{array}\right)\right).
\end{aligned}$$

The formula ϕ expresses that a graph is a comb and that it can be split into two subgraphs, each one comprising disconnected components, where all of them contain exactly 3 fork vertices, where for every component one of the 3 forks is the root of the component. □

Let F and G be the graphs constructed in the proof of Theorem 3.1.4 for GL formulae of quantifier rank k for some $k \in \mathbb{N}$, and let H be the structure containing a single directed edge. Then $F \equiv_k^{\text{GL}} G$ and therefore also $F \uplus H \equiv_k^{\text{GL}} G \uplus H$. Let single-edge-root$(x)$ be a first-order formula defining the vertices that are the first vertices of a directed string, and the string consists of a single edge. Essentially it will be used to identify one of the two vertices in H. Let, similarly, first-edge(e) be the first-order formula over the signature of incidence graphs, defining the edges that are incident to roots of trees.

We show that GL is not closed under FO-definable $(\tau_{\lceil G \rceil}, \tau_{\lceil G \rceil})$-interpretations of width 1, and also that it is not closed under FO non-copying transductions from $\sigma_{\lfloor G \rfloor}$ structures to $\sigma_{\lfloor G \rfloor}$ structures. The results for GL not being closed under $(\sigma_{\lfloor G \rfloor}, \sigma_{\lfloor G \rfloor})$-interpretations of width 1 and FO transductions from $\tau_{\lceil G \rceil}$ structures to $\tau_{\lceil G \rceil}$ structures follow similar lines. Notice that showing that GL is not closed under FO transductions implies that it is not closed under MSO transductions either. We proceed with the FO-definable interpretation.

Let $\Psi = (\Psi_{\text{uni}}, \Psi_{\text{inc}_1}, \Psi_{\text{inc}_2})$ be the following FO-definable $(\tau_{\lceil G \rceil}, \tau_{\lceil G \rceil})$-interpretation.

$$\begin{aligned}
\Psi_{\text{uni}}(x) &= x = x,\\
\Psi_{\text{inc}_1}(e,y) &= \big(\neg\text{first-edge}(e) \wedge \text{inc}_1(e,y)\big) \vee \big(\text{first-edge}(e) \wedge \text{single-edge-root}(y)\big),\\
\Psi_{\text{inc}_2}(e,y) &= \text{inc}_2(e,y).
\end{aligned}$$

Let $f : \text{STRUC}[\tau_{\lceil G \rceil}] \rightarrow \text{STRUC}[\tau_{\lceil G \rceil}]$, be the FO transduction with definition scheme $(\delta_f, \theta_{\text{inc}_1}, \theta_{\text{inc}_2})$, along the same lines as the interpretation Ψ. Similarly, we define the MSO

transduction $g : \mathrm{STRUC}[\sigma_{\lfloor G \rfloor}] \to \mathrm{STRUC}[\sigma_{\lfloor G \rfloor}]$, with definition scheme (δ_g, θ_E) according to the following.

$$\begin{aligned} \delta_g(x) &= x = x, \\ \theta_E(x,y) &= \big(\neg\mathrm{root}(x) \wedge E(x,y)\big) \vee \big(\text{single-edge-root}(x) \wedge \exists z\ (\mathrm{root}(z) \wedge E(z,y))\big). \end{aligned}$$

Consider the forest $F \uplus H$, where H is a single edge. Then both transductions f and g and the interpretation Ψ, given the forest $F \uplus H$, produce a structure with the roots of all trees being the root of the single disconnected edge in H.

Using the interpretation Ψ, a graph comprising a comb together with the noise, as described in the proof of Theorem 3.1.4, becomes a tree where the root is essentially connected to all the noise-strings and the comb. Therefore it is possible to remove all the noise-strings to end up with just the comb of the graph, and then reason along the lines of Lemma 3.2.7 defining the combs that have 1 (mod 3) number of leaves.

Theorem 3.2.8. GL *is not closed under:*

1. FO*-definable* $(\tau_{\lceil G \rceil}, \tau_{\lceil G \rceil})$*-interpretations of width 1.*
2. FO *non-copying transductions from* $\sigma_{\lfloor G \rfloor}$*-structures to* $\sigma_{\lfloor G \rfloor}$*-structures.*
3. FO *non-copying transductions from* $\tau_{\lceil G \rceil}$*-structures to* $\tau_{\lceil G \rceil}$*-structures.*

Proof. Let Ψ be the FO-definable interpretation of width 1, presented above. Similarly let f and g be the transductions above. We show that there exists a sentence ψ such that for any sentence ψ^-, there exist $\mathfrak{A}, \mathfrak{B}$ where $\Psi(\mathfrak{A})$ and $\Psi(\mathfrak{B})$ disagree on ψ (respectively, $f(\mathfrak{A})$ and $f(\mathfrak{B})$ disagree on ψ; $g(\mathfrak{A})$ and $g(\mathfrak{B})$ disagree on ψ), but $\mathfrak{A}$ and $\mathfrak{B}$ agree on ψ^-.

Let $\psi = \varphi_D$ displayed below, let ψ^- be any sentence and let k be the quantifier rank of ψ^-. Let F and G be two graphs constructed in the proof of Theorem 3.1.4 for ψ^-, such that the comb in F has 1 (mod 3) number of leaves and the other one has 0 (mod 3) number of leaves. Let $\mathfrak{A} = F \uplus H$ and $\mathfrak{B} = G \uplus H$. In the formulae below, let ϕ be the GL formula from Lemma 3.2.7.

$$\begin{aligned} \mathrm{Spider}(x) &= \mathrm{root}(x) \wedge \forall y\ (x \neq y \to \neg\mathrm{fork}(y) \wedge (\mathrm{Path}(x,y) \mid \top)), \\ \text{first-fork}(x,y) &= \mathrm{root}(x) \wedge \mathrm{fork}(y) \wedge (\mathrm{Path}(x,y) \mid \top) \wedge \\ &\quad \wedge \neg\exists z\ \big((\mathrm{Path}(x,z) \mid \mathrm{Path}(z,y) \mid \top) \wedge \mathrm{fork}(z)\big), \\ \varphi_{\mathcal{D}} &= \exists! x, y\ \big(\text{first-fork}(x,y) \wedge (\mathrm{Spider}(x) \mid \mathrm{Connected} \wedge \mathrm{root}(y) \wedge \mathrm{fork}(y) \wedge \phi)\big). \end{aligned}$$

The formula first-fork(x,y) above holds when x is the root of a tree and y is the unique fork that is closest to the root. The GL formula φ_D defines the class of trees where each tree can be split up into a graph whose non-root vertices are either leaves or have exactly one child, and a comb with a root that is a fork and such that it has 1 (mod 3) number of leaves.

Consider first $\Psi(\mathfrak{A})$. The interpreted structure is a tree, exactly one of the children y of its root having a comb as a subtree, and the subtrees rooted at the rest of the children being

strings. The same holds for exactly one child y' of the root of the tree $\Psi(\mathfrak{B})$. Furthermore, the comb rooted at y of $\Psi(\mathfrak{A})$ has 1 (mod 3) number of leaves and hence $\Psi(\mathfrak{A}) \models \varphi_D$. On the other hand, $\Psi(\mathfrak{B}) \models \neg\varphi_D$. By definition, $\mathfrak{A} \models \psi^- \Leftrightarrow \mathfrak{B} \models \psi^-$, therefore GL is not closed under FO-definable $(\tau_{\lceil G \rceil}, \tau_{\lceil G \rceil})$-interpretations.

The case is similar for the transductions f and g. □

3.3 Separation Logic

Separation Logic is a logic for analyzing programs that involve pointer variables for memory management. It was introduced by Reynolds in [Rey02] and by Ishtiaq and O'Hearn in [IO01], and has been widely studied since then. We give a short introduction here, while the reader may refer to [BDL08] for more information. The two main connectives of Separation Logic are the separation connective, that works similarly to the separation operator of GL, and its adjoint for which something similar is absent from GL. In the latter paper, the authors consider the expressive power of two syntactic fragments of Separation Logic, namely SL($-\!\!*$) and SL($*$) defined below, and show that the former is equi-expressive to second-order logic and they conjecture that the latter is strictly less expressive than MSO, over memory states. We give a positive resolution to this conjecture.

The structures on which Separation Logic works, consist of a partial function representing the memory heap of a program. Let Loc be a set of locations, namely memory addresses, and let Var be a set of assigned variables. For a function f, we denote the domain of the function by $\mathrm{dom}(f)$ and its range by $\mathrm{ran}(f)$.

Definition 3.3.1 ([BDL08])**.** *A memory state is a pair (s, h) such that $s : Var \to Loc$ and h is a partial function of type $h : Loc \to Loc$. The function s is the store and the function h is the heap of the memory state.*

When the domains of two partial functions h_1 and h_2 are disjoint, it is denoted by $h_1 \perp h_2$ and their disjoint union is denoted by $h_1 * h_2$. The syntax of a Separation Logic formula is inductively defined as:

$$\psi := x = y \mid x \hookrightarrow y \mid \psi_1 \wedge \psi_2 \mid \neg\psi_1 \mid \exists x.\psi_1(x) \mid \psi_1 * \psi_2 \mid \psi_1 -\!\!* \psi_2,$$

where $x, y \in \mathrm{Var}$ and ψ_1, ψ_2 are SL formulae. The semantics is as follows.

$$\begin{array}{lll}
(s,h) \models & x = y & \Leftrightarrow \ s(x) = s(y), \\
(s,h) \models & x \hookrightarrow y & \Leftrightarrow \ h(s(x)) = s(y), \\
(s,h) \models & \psi_1 \wedge \psi_2 & \Leftrightarrow \ (s,h) \models \psi_1 \text{ and } (s,h) \models \psi_2, \\
(s,h) \models & \neg\psi_1 & \Leftrightarrow \ \text{not } (s,h) \models \psi_1, \\
(s,h) \models & \exists x.\psi_1 & \Leftrightarrow \ \text{there is } l \in \mathrm{Loc} \text{ such that } (s[x \mapsto l], h) \models \psi_1, \\
(s,h) \models & \psi_1 * \psi_2 & \Leftrightarrow \ \text{there are } h_1, h_2 \text{ such that } h = h_1 * h_2 \text{ and } (s,h_i) \models \psi_i, i \in \{1,2\}, \\
(s,h) \models & \psi_1 -\!\!* \psi_2 & \Leftrightarrow \ \text{for any } h' \text{ disjoint from } h \text{ such that } (s,h') \models \psi_1, (s, h * h') \models \psi_2.
\end{array}$$

Two important syntactic fragments of SL are SL($*$) and SL($-\!\!*$). The latter uses all constructs from the above apart from the separation connective '$*$'. In [BDL08], it is shown that SL($-\!\!*$) is equi-expressive to Second Order Logic over structures represented as memory states. Similarly the syntactic fragment SL($*$) omits the use of the *magic wand* '$-\!\!*$'. In the same paper the authors conjecture that SL($*$) is strictly less expressive than MSO over structures represented by memory states. We prove that this is indeed the case.

Definition 3.3.2. *For any memory heap h with $H = ran(h) \cup dom(h)$, its associated graph is defined to be a graph $G_h = (V, E)$, such that $V = H$ and for any $l, l' \in H$, $E(l, l')$ holds in G_h if and only if $h(l) = l'$.*

Definition 3.3.3. *For any memory state (s, h) where $S = dom(s)$ and $H = ran(s) \cup dom(h) \cup ran(h)$, its associated structure is defined to be the tuple (G_h, ρ) where G_h is the graph associated with the memory heap h, and ρ is the mapping assigning vertices of G_h to first order variables, such that for each $x_i \in S$ and for each $l \in H$, $s(x_i) = l$ if and only if $\rho(x_i) = l$.*

For any SL($*$) formula ψ, we denote with $[\psi]_{\mathrm{GL}}$ its translation into GL, according to the following inductive definition.

$$\begin{aligned}
[x = y]_{\mathrm{GL}} &:= x = y, \\
[x \hookrightarrow y]_{\mathrm{GL}} &:= (E(x, y) \mid \top), \\
[\psi_1 \wedge \psi_2]_{\mathrm{GL}} &:= [\psi_1]_{\mathrm{GL}} \wedge [\psi_2]_{\mathrm{GL}}, \\
[\psi_1 * \psi_2]_{\mathrm{GL}} &:= [\psi_1]_{\mathrm{GL}} \mid [\psi_2]_{\mathrm{GL}}, \\
[\neg\psi']_{\mathrm{GL}} &:= \neg[\psi']_{\mathrm{GL}}, \\
[\exists x.\psi']_{\mathrm{GL}} &:= \exists x\ [\psi']_{\mathrm{GL}}.
\end{aligned}$$

Lemma 3.3.4. *For any memory state (s, h) and any SL($*$) formula ψ, $(s, h) \models_{\mathrm{SL}(*)} \psi$ if and only if $(G_h, \rho) \models_{\mathrm{GL}} [\psi]_{\mathrm{GL}}$, where (G_h, ρ) is the graph and assignment associated with the memory state (s, h).*

Proof. We proceed by induction on the structure of the formula.

$\psi = x = y$. Assume that $(s, h) \models_{\mathrm{SL}(*)} x = y$. Then $s(x) = s(y)$ and by the definition of the graph associated with a memory state, (G_h, ρ), it holds that $\rho(x) = \rho(y)$. Therefore, $(G_h, \rho) \models_{\mathrm{GL}} x = y$ For the opposite direction assume that $(G_h, \rho) \models_{\mathrm{GL}} x = y$ and hence $\rho(x) = \rho(y)$. Then $s(x) = s(y)$ and hence $(s, h) \models_{\mathrm{SL}(*)} x = y$.

$\psi = x \hookrightarrow y$. Assume that $(s, h) \models_{\mathrm{SL}(*)} x \hookrightarrow y$. Then $h(s(x)) = s(y)$ and by the definition of (G_h, ρ) it holds that $(G_h, \rho) \models_{\mathrm{GL}} (E(x, y) \mid \top)$. Similarly, if $(G_h, \rho) \models_{\mathrm{GL}} (E(x, y) \mid \top)$, then $h(s(x)) = s(y)$ and $(s, h) \models_{\mathrm{SL}(*)} x \hookrightarrow y$.

$\psi = \psi_1 \wedge \psi_2$. It holds that $(s, h) \models_{\mathrm{SL}(*)} \psi_1 \wedge \psi_2$ if and only if $(s, h) \models_{\mathrm{SL}(*)} \psi_1$ and $(s, h) \models_{\mathrm{SL}(*)} \psi_2$. By the inductive hypothesis, for $i \in \{1, 2\}$, $(s, h) \models_{\mathrm{SL}(*)} \psi_i$ if and only if $(G_h, \rho) \models_{\mathrm{GL}} [\psi_i]_{\mathrm{GL}}$. Finally this is the case if and only if $(G_h, \rho) \models_{\mathrm{GL}} [\psi_1 \wedge \psi_2]_{\mathrm{GL}}$.

$\psi = \neg\psi'$. By the inductive hypothesis $(s,h) \models_{\mathrm{SL}(*)} \psi'$ if and only if $(G_h,\rho) \models_{\mathrm{GL}} [\psi']_{\mathrm{GL}}$, and therefore $(s,h) \models_{\mathrm{SL}(*)} \neg\psi'$ if and only if $(G_h,\rho) \models_{\mathrm{GL}} [\neg\psi']_{\mathrm{GL}}$.

$\psi = \psi_1 * \psi_2$. Assume that $(s,h) \models_{\mathrm{SL}(*)} \psi_1 * \psi_2$. Then there exist disjoint h_1, h_2 such that for $i \in \{1,2\}$, $(s,h_i) \models_{\mathrm{SL}(*)} \psi_i$. By the inductive hypothesis this is the case if and only if $(G_{h_i},\rho) \models_{\mathrm{GL}} [\psi_i]_{\mathrm{GL}}$. By the definition of a graph associated to a memory state, if h_1 and h_2 are disjoint, then the graphs G_{h_1} and G_{h_2} are disjoint. Therefore, $(G_h,\rho) \models_{\mathrm{GL}} [\psi_1 \mid \psi_2]_{\mathrm{GL}}$.

Similarly, if $(G_h,\rho) \models_{\mathrm{GL}} [\psi_1 \mid \psi_2]_{\mathrm{GL}}$ then there exist G_1, G_2 such that $G_h = G_1 \mid G_2$ and $(G_i,\rho) \models_{\mathrm{GL}} [\psi_i]_{\mathrm{GL}}$, for $i \in \{1,2\}$. These two graphs (G_1,ρ) and (G_2,ρ) are associated with some memory states (s,h_1) and (s,h_2), where h_1 and h_2 are disjoint. By the inductive hypothesis, $(s,h_i) \models_{\mathrm{SL}(*)} \psi_i$ for $i \in \{1,2\}$, and therefore, since h_1 and h_2 are disjoint, it holds that $(s,h) \models_{\mathrm{SL}(*)} \psi_1 * \psi_2$.

$\psi = \exists x.\psi'$. Assume that $(s,h) \models_{\mathrm{SL}(*)} \exists x.\psi'$. Then there a location l such that $(s[x \mapsto l],h) \models_{\mathrm{SL}(*)} \psi'$. By the inductive hypothesis, $(G_h,\rho[x \mapsto l]) \models_{\mathrm{GL}} [\psi']_{\mathrm{GL}}$ and therefore, $(G_h,\rho) \models_{\mathrm{GL}} [\exists x\ \psi']_{\mathrm{GL}}$.

For the opposite direction, if $(G_h,\rho) \models_{\mathrm{GL}} \exists x\ [\psi']_{\mathrm{GL}}$, then there exists l_x in G_h such that $(G_h,\rho[x \mapsto l_x]) \models_{\mathrm{GL}} [\psi']_{\mathrm{GL}}$. By the inductive hypothesis it is the case that $(s[x \mapsto l_x],h) \models_{\mathrm{SL}(*)} \psi'$ and therefore $(s,h) \models_{\mathrm{SL}(*)} \exists x.\psi'$.

□

In addition to the boolean connectives and the atomic cases introduced for SL, MSO formulae use the two syntactic connectives $\exists P.\psi$ and $P(x)$, where $x \in \mathrm{Var}$ and $P \subseteq \mathrm{Var}$. The satisfaction relation is defined similarly for the rest of the connectives and for the two additional ones, it makes use of an additional environment $\mathcal{E}$ and is as shown below.

$$\begin{array}{lll} (s,h),\mathcal{E} \models & \exists P.\psi & \Leftrightarrow \text{ there is } R \subseteq \mathrm{Loc}, \text{ such that } (s,h),\mathcal{E}[P \mapsto R] \models \psi, \\ (s,h),\mathcal{E} \models & P(x) & \Leftrightarrow \ s(x) \in \mathcal{E}(P). \end{array}$$

We associate with any memory state (s,h) and environment $\mathcal{E}$ for set variables, the tuple (G_h,ρ,ζ), where (G_h,ρ) is the graph and assignment associated with (s,h) and ζ is the mapping assigning subsets of the set of vertices H of G_h to second-order variables such that for any $P_i \in \mathrm{dom}(\mathcal{E})$, and any $L \subseteq H$, $\zeta(P_i) = L$ if and only if $\mathcal{E}(P_i) = L$.

Let ψ be some MSO formula over the structures represented as memory states. Then we define $[\psi]_{\mathrm{MSO}}$ to be the translation of ψ over structures as graphs. The translation is defined inductively, identical to the GL case for the boolean connectives, equality and the heap function, and is as follows for the two additional connectives.

$$\begin{array}{rl} [\exists X.\psi_1]_{\mathrm{MSO}} = & \exists X\ [\psi_1]_{\mathrm{MSO}}, \\ [X(x)]_{\mathrm{MSO}} = & X(x). \end{array}$$

Lemma 3.3.5. *For any memory state (s,h), environment $\mathcal{E}$ and any* MSO *formula ψ over structures represented as memory states, $(s,h),\mathcal{E} \models_{\text{MSO}} \psi$ if and only if $(G_h,\rho,\zeta) \models_{\text{MSO}} [\psi]_{\text{MSO}}$, where (G_h,ρ,ζ) is the graph associated with the memory state (s,h) and the environment $\mathcal{E}$.*

Proof. The proof of this lemma is identical to the proof of Lemma 3.3.4 for the relevant cases of boolean connectives and so it only needs to be shown for the two additional cases below.

$\psi = P(x)$. Assume that $(s,h),\mathcal{E} \models_{\text{MSO}} P(x)$. Then $s(x) \in \mathcal{E}(P)$. Therefore $\rho(x) \in \zeta(P)$ and hence $(G_h,\rho,\zeta) \models_{\text{MSO}} P(x)$. Similarly assume that $(G_h,\rho,\zeta) \models_{\text{MSO}} P(x)$. Then $\rho(x) \in \zeta(P)$ which implies that $s(x) \in \mathcal{E}(P)$. Therefore $(s,h),\mathcal{E} \models_{\text{MSO}} P(x)$.

$\psi = \exists P.\psi_1$. Assume that $(s,h),\mathcal{E} \models_{\text{MSO}} \exists P.\psi_1$. Therefore there exists $R \subseteq \text{Loc}$ such that $(s,h),\mathcal{E}[P \mapsto R] \models_{\text{MSO}} \psi_1$. By the inductive hypothesis $(G_h,\rho,\zeta[P \mapsto R]) \models_{\text{MSO}} [\psi_1]_{\text{MSO}}$, and hence $(G_h,\rho,\zeta) \models_{\text{MSO}} \exists P\ [\psi_1]_{\text{MSO}}$.

For the opposite direction assume that $(G_h,\rho,\zeta) \models_{\text{MSO}} [\exists P\ \psi_1]_{\text{MSO}}$. Then there exists $R \subseteq \text{Loc}$ such that $(G_h,\rho,\zeta[P \mapsto R]) \models_{\text{MSO}} [\psi_1]_{\text{MSO}}$. By the inductive hypothesis $(s,h),\mathcal{E}[P \mapsto R] \models_{\text{MSO}} \psi_1$ and therefore $(s,h),\mathcal{E} \models_{\text{MSO}} \exists P.\psi_1$.

□

Definition 3.3.6. *A graph G is compatible to heaps if it is such that for all $v_1,v_2,v_3 \in V(G)$ it is not the case that $E(v_1,v_2)$ and $E(v_1,v_3)$.*

In other words, a graph G is compatible to heaps if it is the graph of a partial unary function.

Lemma 3.3.7. *For any graph G, if G is compatible with heaps, then there exists a heap h, such that G is the graph associated with the heap h.*

Proof. Let G be some graph that is compatible with heaps. Let h be the partial function such that for any $v_1,v_2 \in V(G) = \text{Loc}$, it holds that $h(v_1) = v_2$ if and only if $E(v_1,v_2)$. Then it remains to show that h is indeed a function, which follows from the assumption that G is compatible. □

Similarly there is a translation of GL formulae into SL$(*)$ ones and a translation of MSO formulae over graphs compatible to memory states, to MSO formulae over structures represented as memory states. The proof is very similar to the ones for Lemma 3.3.4 and Lemma 3.3.5 respectively, and it is omitted.

Lemma 3.3.8. *For any* GL *formula ψ, there exists a* SL$(*)$ *formula ψ' over structures represented as memory states such that for any graph G compatible to memory states, $(s,h) \models_{\text{SL}(*)} \psi'$ if and only if $(G,\rho) \models_{\text{GL}} \psi$, where (G,ρ) is the graph associated with (s,h).*

Lemma 3.3.9. *For any* MSO *formula ψ, there exists an* MSO *formula ψ' over structures represented as memory states such that for any graph G compatible to memory states, $(s,h),\mathcal{E} \models_{\text{MSO}} \psi'$ if and only if $(G,\rho,\zeta) \models_{\text{MSO}} \psi$, where $(s,h),\mathcal{E}$ is the memory state and environment associated with (G,ρ,ζ).*

Theorem 3.3.10. SL(∗) *is strictly less expressive than* MSO *over structures represented as memory states.*

Proof. We use Theorem 3.1.4 and the translations from GL to SL(∗) and vice versa that were given above, to show that a particular class of memory heaps resembling the class of forests used in Theorem 3.1.4, is not definable in SL(∗).

Consider the function that maps each vertex to its parent, and let all trees be represented in a way such that the set of edges E of a tree is that function. This way all forests are compatible with memory states, and therefore according to Lemma 3.3.7 for every forest there exists a heap with which it is associated. Let φ_{MSO} be an MSO formula that defines the set of forests in which every tree has 1 (mod 3) number of leaves. Then according to Lemma 3.3.9, there exists an MSO formula ψ_{MSO} over structures represented as memory states, such that for any forest F, $F \models_{\text{MSO}} \varphi_{\text{MSO}}$ if and only if $(s,h),\mathcal{E} \models_{\text{MSO}} \psi_{\text{MSO}}$, where F is the graph associated with $(s,h),\mathcal{E}$.

Suppose that there exists an SL(∗) formula $\psi_{\text{SL}(*)}$ such that for any memory state (s,h), $(s,h) \models_{\text{MSO}} \psi_{\text{MSO}}$ if and only if $(s,h) \models_{\text{SL}(*)} \psi_{\text{SL}(*)}$. According to Lemma 3.3.4, there exists a GL formula φ_{GL}, such that for any memory state (s,h), $(s,h) \models_{\text{SL}(*)} \psi_{\text{SL}(*)}$ if and only if $(G,\rho) \models_{\text{GL}} \varphi_{\text{GL}}$, where (G,ρ) is the graph associated with (s,h).

According to Theorem 3.1.4, there exist forests G_1, G_2 such that $G_1 \models_{\text{MSO}} \varphi_{\text{MSO}}$ and $G_2 \models_{\text{MSO}} \neg\varphi_{\text{MSO}}$, but are such that, $G_1 \models_{\text{GL}} \varphi_{\text{GL}} \Leftrightarrow G_2 \models_{\text{GL}} \varphi_{\text{GL}}$. Let h_1 and h_2 be the heaps associated with the graphs G_1 and G_2 respectively according to Lemma 3.3.7. According to the definition of the GL formula φ_{GL}, $(s,h_1) \models_{\text{SL}(*)} \psi_{\text{SL}(*)} \Leftrightarrow (s,h_2) \models_{\text{SL}(*)} \psi_{\text{SL}(*)}$. Furthermore, by definition of ψ_{MSO}, $(s,h_1) \models_{\text{MSO}} \psi_{\text{MSO}}$ and $(s,h_2) \models_{\text{MSO}} \neg\psi_{\text{MSO}}$, which is a contradiction. □

Chapter 4

Graph Logic with Recursion

The expressive power of GL is greatly increased when a construct for fixed point operations is added to the logic. The resulting logic is called GL_μ and is formally defined in Section 1.6.

It was shown in [DGG07] that PSPACE-complete problems are expressible in GL_μ while the combined complexity remains in PSPACE as is the case for GL (without the recursion operator). We show that GL_μ is strictly more expressive than MSO over words and trees and in the case of words, we compare it to other grammars that strictly contain regular languages.

4.1 On Graphs

The relationship between GL_μ and MSO over graphs in general is still not clear in terms of whether MSO definable graph properties are included in the GL_μ ones. As was mentioned, there are known examples of properties definable in GL_μ and not in MSO, but there are also MSO definable properties, such as 3-colourability, that are not known to be expressible in GL_μ, although, as shown by Corollary 2.1.3 in Section 2.1, 4-colourability is expressible in GL without the use of recursion.

Apart from 3-colourability on graphs, the class of graphs having a Hamiltonian Cycle is thought to be not definable in GL. It should be noted that Hamiltonicity is not definable in MS_1 either, only MS_2, and is an example often used to specify that over graphs in general MS_2 is more expressive than MS_1. Despite that the class of 3-colourable graphs is thought to remain not definable even when recursion is present in the logic, Hamiltonicity of undirected graphs, becomes expressible in GL_μ, and in particular it can be expressed using the formulae below.

$$\begin{aligned} \mathrm{HamCycle} &= \forall x\ \deg_2(x) \wedge \mathrm{Connected}. \\ \varphi_{HC} &= \mu R.\ \Big(\mathrm{HamCycle} \vee \big(\forall y\ \big(\deg_{>1}(y)\big) \wedge \exists z, w\big(E(z,w) \mid R\big)\big)\Big), \end{aligned}$$

Theorem 4.1.1. *A graph G has a Hamiltonian Cycle if and only if G satisfies φ_{HC}.*

Proof. Fix a graph G. For the *only if* direction assume that G has a Hamiltonian Cycle. Then G is either a cycle or contains a cycle through all of its vertices. In the first case, every vertex of G has degree 2, which means that G satisfies HamCycle and consequently $G \models \varphi_{HC}$.

In the second case, where G contains a cycle through all of its vertices, it is the case that all vertices have degree larger than 1. Let G be a graph with a Hamiltonian Cycle C. If there are n edges in G that are not part of the cycle C, we write $G = H_n \uplus C$, where H_n is the graph with all the edges outside C, all of whose endpoints are in C by definition. We prove by induction on k, that for any G such that $G = H_k \uplus C$ for some $k \in \mathbb{N}$, $G \models \varphi_{HC}$. For $k = 0$, $G = C$ and by the argument above $G \models \varphi_{HC}$. Suppose the statement holds for $k = K$ for some $K \in \mathbb{N}$, and consider a graph $G = H_{(K+1)} \uplus C$. Since G has a Hamiltonian Cycle, all of the vertices have degree larger than 1, therefore it only needs to be shown that there exists an edge e such that $G - \{e\} \models \varphi_{HC}$. Let e be any edge in $H_{(K+1)}$. Then $G - \{e\}$ has a Hamiltonian Cycle and therefore satisfies φ_{HC} by the inductive hypothesis.

For the *if* direction, assume that G is a model of φ_{HC}. We prove by induction on the number of recursion steps that G contains a Hamiltonian Cycle. If G satisfies φ_{HC} in the first recursive step, then $G \models \text{HamCycle}$, and hence is a Hamiltonian Cycle. Assume that the statement is true for any graph satisfying φ_{HC} after k recursive steps, and assume that G satisfies φ_{HC} at the $(k+1)$-th recursive step. Then there exists an edge e in G such that $G - \{e\}$ satisfies R and hence satisfies φ_{HC} after k recursive steps. Since $G \models \varphi_{HC}$, all vertices have degree greater than 1, and therefore removing any edge from G, no vertex is removed. Hence G and $G - \{e\}$ have the same set of vertices. By the inductive hypothesis, $G - \{e\}$ has a Hamiltonian Cycle C and therefore, since G has no more vertices, C is also a Hamiltonian Cycle of G. □

4.2 On Words

We have already mentioned that GL has the same expressive power as MSO over words. In this section we examine what can be expressed in GL_μ over words, which intuitively should be strictly more than what can be expressed in MSO. In particular consider the non-regular language $L = \{a^n b^n \mid n \in \mathbb{N}\}$. The language L can be proved to be not regular by applying the Pumping Lemma for regular languages. This means that L is not MSO-definable, but on the other hand it is GL_μ-definable. Let ψ_L be the following GL_μ formula defining this class of words.

$$\begin{aligned} \psi_L &= \mu R.\ \Big(\varphi_{ab} \vee \big(\exists x, y, z, w\ (\text{In}_0(x) \wedge \text{Out}_0(y) \wedge \varphi_R(x, y, z, w))\big)\Big), \\ \varphi_R(x, y, z, w) &= (E_a(x, z) \mid E_b(w, y) \mid \text{Path}(z, w) \wedge R), \\ \varphi_{ab} &= \exists x, y, z\ (E_a(x, y) \mid E_b(y, z)). \end{aligned}$$

The formula is satisfied by a word if that word is ab or if we can recursively remove an edge with label a from the left and an edge labelled with b from the right, until we get the word ab.

4.2.1 PSPACE-complete problems on strings

In [DGG07] the authors show that the model-checking complexity of GL_μ is in PSPACE and that the data complexity of GL_μ on graphs is PSPACE-complete. Here we show that even on strings

the data complexity of GL_μ is PSPACE-complete.

We present an encoding of Quantified Boolean Formulae (QBF) into strings together with a GL_μ formula which is satisfied by the string if and only if the QBF instance has a solution. The encoding of Quantified Boolean Formulae into strings is as follows. First, all variables present in the formula are listed, and then the formula is presented. During the encoding of the latter, there are letters in the alphabet available for denoting the start and finish of clauses, variables and quantifiers. Before giving the details for the QBF case, we give an encoding and a GL_μ formula for the Boolean Satisfiability Problem (SAT) instances.

In both the QBF and SAT cases, the string encoding a particular instance, begins with encoding all the variables that appear in the formula, and the encoding of the actual formula follows.

A variable x_ℓ in the beginning of the string is encoded as $\sharp\mathbf{bin}(\ell)$, where $\mathbf{bin}(\ell)$ denotes the binary representation of ℓ. Each variable appears in at least one clause and in the clauses it appears in positive form or negative. To denote that it appears in positive form we append the symbol $+$ and for the negative form we use $-$. The beginning of a clause is denoted with '$\langle$' and the end of it with '$\rangle$'.

For example, the SAT clause $(x_1 \vee x_2 \vee \neg x_3)$ is encoded as $\langle +1 + 10 - 11\rangle$. The symbol '$\|$' denotes the point where the listing of the variables finishes and the formula starts. Therefore, the formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is encoded as shown below:

$$\sharp 1\sharp 10\sharp 11 \parallel \langle +1 + 10 - 11\rangle\langle -1 + 10 + 11\rangle.$$

We write the *variables listing encoding*, for the first part of the string that simply encodes the variables used in the formula. Similarly the *formula encoding* is the part of the string encoding the actual formula. The delimiter '$\|$' is simply separating the variables listing encoding from the formula encoding.

The method with which GL_μ can express that a SAT formula is satisfiable, is applied with the formulae given below and is as follows. If a SAT instance, encoded into a string s as described earlier, is satisfiable, then one can split this string into two subgraphs, say s_1 and s_2, where one of them (say s_1) contains part of the variables listing encoding, and the other one contains the rest. The idea is that s_1 contains all the variables that are assigned the false value, and s_2 contains all the variables that are assigned the true value together with the actual encoded formula. One then needs to check that for every clause in the SAT formula in s_2, either there exists a positive variable in the clause that also appears in the variables listing encoding subgraph of s_2, or otherwise there exists a negative variable in the clause that does not appear in the latter subgraph of s_2.

Checking whether the encoding of a literal in a clause matches the encoding of a variable in the variables listing encoding, is where the recursion operator of GL_μ is needed, going through each edge one by one ensuring they have the same label.

For each label in the alphabet of the encodings, we define a formula expressing that in the string, this particular label appears in only one position. Let σ below denote a label in the

alphabet.

$$\text{unique}_\sigma = \exists x, y\ \big((E_\sigma(x,y) \mid \top) \wedge \forall w, v\ (x \neq w \wedge y \neq v) \rightarrow \neg(E_\sigma(w,v) \mid \top)\big).$$

Similarly, we define the following two formulae, one to express that a set of strings contains only bits and one to express that such a set of strings contains no bits.

$$\begin{aligned} \text{OnlyBits} &= \forall x, y\ \big((E(x,y) \mid \top) \rightarrow (E_0(x,y) \vee E_1(x,y) \mid \top)\big), \\ \text{NoBits} &= \neg(\top \mid \text{OnlyBits}). \end{aligned}$$

The free variable of the formulae below, expressing a particular encoding, denotes the position at which this encoding starts.

$$\begin{aligned} \text{Var}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w\ ((E(x,w) \mid \top) \rightarrow (E_\sharp(x,w) \mid \text{OnlyBits})), \\ \text{Pos}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w\ ((E(x,w) \mid \top) \rightarrow (E_+(x,w) \mid \text{OnlyBits})), \\ \text{Neg}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w\ ((E(x,w) \mid \top) \rightarrow (E_-(x,w) \mid \text{OnlyBits})), \\ \text{Clause}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w\ ((E(x,w) \mid \top) \rightarrow (E_\langle(x,w) \mid \top)) \wedge \\ &\quad \wedge \text{unique}_\langle \wedge \text{unique}_\rangle, \\ \text{SetOfVars} &= \forall x, y\ \big(\text{first}(x) \wedge \text{last}(y) \rightarrow \neg(\top \mid \text{Path}(x,y) \wedge \neg\text{Var}(x))\big), \\ \text{SetOfClauses} &= \forall x, y\ \big(\text{first}(x) \wedge \text{last}(y) \rightarrow \neg(\top \mid \text{Path}(x,y) \wedge \neg\text{Clause}(x))\big). \end{aligned}$$

The formulae $\text{Var}(x), \text{Pos}(x)$ and $\text{Neg}(x)$ define respectively the set of strings that are encodings of variables in the variables listing encoding, the positive and the negative literals in the clauses. The formula $\text{Clause}(x)$ defines the set of strings that are encodings of a single clause. Finally the formulae SetOfVars and SetOfClauses define the graphs that are sets of disconnected strings, each one of which encodes a single variable and a single clause respectively. The above formulae assist in defining the following ones.

$$\begin{aligned} \text{False} &= (\text{SetOfVars} \mid \text{SetOfVars}), \\ \text{True} &= (\text{SetOfVars} \mid \text{SetOfVars}), \\ \text{RestClause} &= \forall x, y\big(\text{first}(x) \wedge (E(x,y) \mid \top) \rightarrow \\ &\quad (E_\langle(x,y) \vee E_\rangle(x,y) \vee E_+(x,y) \vee E_-(x,y) \mid \top)\big), \\ \text{Rest} &= (\text{SetOfVars} \mid \text{SetOfVars} \mid \text{RestClause}), \\ \text{Equal} &= \exists x, y\ \Big(\text{first}(x) \wedge \text{first}(y) \wedge \\ &\quad \wedge \exists z, w\big(E_\sharp(x,z) \mid (E_+(y,w) \vee E_-(y,w) \mid \text{EqualBit})\big)\Big), \\ \text{EqualBit} &= \mu R.\ \big(\epsilon \vee \exists x, y, z, w\ \text{first}(x) \wedge \text{first}(y) \wedge \\ &\quad \wedge\big((E_0(x,z) \mid E_0(y,w)\ |R) \vee (E_1(x,z) \mid E_1(y,w) \mid R)\big)\big). \end{aligned}$$

The formula False will be used to split in two subgraphs the encodings of the variables that are assigned to false. This is because a string contains encodings of variables if and only if that string can be split into two subgraphs, both comprising disconnected substrings, where each such substring is the encoding of a single variable. The formula RestClause is used to denote

the subgraphs that remain when we pick a variable from a clause. Similarly the formula Rest defines the set of graphs comprising strings satisfying RestClause together with some strings encoding variables.

The formula EqualBit is the one formula using the recursion operator, and is satisfied by a graph consisting of two strings with edges labelled with 1 and 0, if these two strings encode the same bitstrings. The formula EqualBit is used by the formula Equal, that is satisfied by a graph of two strings, when one of these strings encodes a variable in the variables listing encoding and the other encodes the same variable appearing as a literal in the clauses.

Finally the formula expressing satisfiability of a SAT formula encoded into a string according to the above is the one presented below.

$$
\begin{aligned}
\text{Assign} &= \exists x\ (\text{Pos}(x) \vee \text{Neg}(x) \mid \text{Rest}) \wedge \\
&\quad \wedge \Big((\text{Pos}(x) \mid \text{Rest}) \rightarrow \exists y\ \big(\text{Rest} \mid (\text{Pos}(x) \mid \text{Var}(y)) \wedge \text{Equal}\big) \Big) \wedge \\
&\quad \wedge \Big((\text{Neg}(x) \mid \text{Rest}) \rightarrow \forall y\ \neg\big((\text{Rest} \mid (\text{Neg}(x) \mid \text{Var}(y)) \wedge \text{Equal})\big) \Big), \\
\text{TruePart} &= \Big((\text{SetOfClauses} \mid \text{SetOfClauses}) \Mapsto \big(\exists x\ (\text{Clause}(x) \mid \text{True}) \rightarrow \text{Assign}\big) \Big), \\
\text{NPSat} &= \exists x, y\ (E_{\|}(x, y) \mid \text{False} \mid \text{TruePart}),
\end{aligned}
$$

where the operator '$\Mapsto$' is defined in Section 1.5.

Assuming that a subgraph is composed of a single clause together with a substring of the variables listing encoding, the formula Assign expresses then that there either exists a positive literal in the clause that matches the encoding of a variable, or otherwise there exists a negative literal in the clause that matches no variable in the substring of the variables listing encoding that appears in the graph.

This way, if all the variables assigned the value false are removed from a string, one can find for each clause, a literal that makes it true. This is what the formula NPSat expresses with the help of the formula TruePart. The latter formula holds when, for every subgraph comprising a set of variables and a single clause, the formula Assign holds.

Extending the string encoding of the formulae to accommodate for alternating quantifiers of a formula, one can define classes of strings that are complete for any level of the Polynomial Hierarchy. Furthermore, by defining a GL_μ sentence that recursively deals with the different alternating levels of quantification, it is possible to define a class of strings that is PSPACE-complete.

Theorem 4.2.1. *There exists a* GL_μ *formula* φ *that defines a class of strings that is* PSPACE*-complete.*

Checking validity of Quantified Boolean Formulae is complete for PSPACE, and so we give an encoding of QBF instances into strings and a formula φ which a string S satisfies if and only if the QBF instance it encodes is valid. Consider the following encoding in addition to what was described above, to accommodate for quantifiers. The variables listing encoding, and in

particular the encoding of the variables in it, is extended with an additional index corresponding to the quantifier block in which the variable gets bound. The variable index from the encoding described for SAT instances, and which is unique for each variable, also remains in the encoding of QBF instances, with additional symbols in the alphabet to denote the limits of these indices. Let then the alphabet used for the encoding, include the symbols '∀','∃','⌈' and '⌉'.

The encoding initially describes the quantifier blocks appearing in the formula by listing the variables bound and the quantifier type that precedes them, both accompanied by the quantifier index, which is given, again, in binary. The part of the string encoding the variables that appear in the formula, is again called the variables listing encoding. As an example, let ψ be the formula $\forall x_1, x_2\ \exists x_3 (x_1 \vee \neg x_2 \vee x_3)$. Then ψ is encoded into the following string:

$$\underbrace{\forall\lceil 01\rceil\sharp\lceil 01\rceil 01\sharp\lceil 01\rceil 10}_{\forall x_1,x_2}\underbrace{\exists\lceil 10\rceil\sharp\lceil 10\rceil 11}_{\exists x_3} \parallel \underbrace{\langle +\lceil 01\rceil 01 - \lceil 01\rceil 10 + \lceil 10\rceil 11\rangle}_{\text{clause of formula } \psi}.$$

A variable x_n that is bound by the m-th quantifier, is thus encoded as $\sharp\lceil\mathbf{bin}(m)\rceil\mathbf{bin}(n)$, where $\mathbf{bin}(n)$ denotes the number n in binary. Similarly, when the variable appears in a clause instead of the variables listing encoding, the symbol '$\sharp$' is omitted at the beginning of the variable encoding, and instead one of the symbols '$+$' or '$-$' is used to denote whether the variable appears in a positive or negative form.

The formula Index below defines the strings that encode a quantifier index, as a connected sequence of 0's and 1's that starts and ends with the appropriate symbols '⌈' and '⌉'. The formulae given earlier expressing that a string encodes a variable are redefined accordingly:

$$\begin{aligned}
\text{Index} &= \text{Connected} \wedge \exists x_1, x_2, y_1, y_2 \\
&\quad \big(\text{first}(x_1) \wedge \text{last}(y_2) \wedge (E_{\lceil}(x_1, y_1) \mid E_{\rceil}(x_2, y_2) \mid \top)\big) \wedge \\
&\quad \wedge \text{unique}_{\lceil} \wedge \text{unique}_{\rceil}, \\
\text{Var}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w \Big((E(x,w) \mid \top) \rightarrow \big(E_{\sharp}(x,w) \mid \text{OnlyBits} \mid \text{Index}\big)\Big), \\
\text{Pos}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w \Big((E(x,w) \mid \top) \rightarrow \big(E_{+}(x,w) \mid \text{OnlyBits} \mid \text{Index}\big)\Big), \\
\text{Neg}(x) &= \text{first}(x) \wedge \text{Connected} \wedge \forall w \Big((E(x,w) \mid \top) \rightarrow \big(E_{-}(x,w) \mid \text{OnlyBits} \mid \text{Index}\big)\Big).
\end{aligned}$$

The GL_μ formula that checks satisfiability of the encoded Quantified Boolean Formula, needs to be extended to accommodate for the alternating quantifiers. This is done by recursively going through the alternating quantifier blocks and assigning the value true or false to the variables. First, a formula is needed to check that the indices of the variables being assigned at each recursive step match the index of the respective quantifier at hand.

$$\text{EqualIndices} = \Big(\top \Rrightarrow \big((\text{Index} \mid \text{Index}) \rightarrow (\text{NoBits} \mid (\text{OnlyBits} \wedge \text{EqualBit}))\big)\Big).$$

The formula EqualIndices expresses that whenever two indices are separated from the rest of the graph, then the two sequences of bits, between the index delimiter symbols, are equal to each other.

A few more formulae are defined next, essentially used to define that a string is a quantifier together with its index. This is accomplished with QuantE and QuantA as shown below, for the cases where the quantifier is $\exists$ and $\forall$ respectively.

$$\begin{aligned}
\text{FirstE}(x) &= \exists y\ (E_\exists(x,y) \mid \top) \wedge \forall w, w'\ \big((E_\exists(w,w') \vee E_\forall(w,w') \mid \top) \rightarrow (\text{Path}(x,w) \mid \top)\big),\\
\text{FirstA}(x) &= \exists y\ (E_\forall(x,y) \mid \top) \wedge \forall w, w'\ \big((E_\exists(w,w') \vee E_\forall(w,w') \mid \top) \rightarrow (\text{Path}(x,w) \mid \top)\big),\\
\text{QuantE}(x) &= \text{Connected} \wedge \exists y\ (E_\exists(x,y) \mid \text{Index}),\\
\text{QuantA}(x) &= \text{Connected} \wedge \exists y\ (E_\forall(x,y) \mid \text{Index}),\\
\text{NoQuant} &= \neg\exists x, y\ (E_\exists(x,y) \vee E_\forall(x,y) \mid \top).
\end{aligned}$$

The formula FirstE(x) is satisfied at some vertex x, if a quantifier block begins at x, and is such that all the rest of the quantifiers are after this one in the encoding of the string. Essentially, as shown below, at each recursion step, after a quantifier block has been dealt with, the edge containing the quantifier of that block is removed. Thus, the first quantifier in the string after k recursive steps is the $(k+1)$-st quantifier. The definition of the formula Equal is changed to the following one, to accommodate the different encoding of variables.

$$\begin{aligned}
\text{Equal} = \exists x, y\ \Big(&\text{first}(x) \wedge \text{first}(y) \wedge\\
&\wedge\exists z, w\Big(E_\sharp(x,z) \mid \big(E_+(y,w) \vee E_-(y,w)\big) \mid\\
&\big((\text{Index} \mid \text{Index}) \wedge \text{EqualIndices}\big) \mid \text{EqualBit}\Big)\Big).
\end{aligned}$$

The formulae that follow use the bit checking between the indices of variables, that was defined earlier, to make sure that at each recursive step only the appropriate variables are assigned a truth value.

$$\begin{aligned}
\text{TrueClause} &= \text{NoQuant} \wedge \exists x, y\ (E_\parallel(x,y) \mid \text{TruePart}),\\
\text{ExistsAssign} &= \exists x\ \text{FirstE}(x) \wedge \Big(\big((\text{QuantE}(x) \mid \text{False}) \wedge \text{EqualIndices}\big) \mid R\Big),\\
\text{ForallAssign} &= \exists x\ \text{FirstA}(x) \wedge \Big(\big((\text{QuantA}(x) \mid \text{False}) \wedge \text{EqualIndices}\big) \models\!\!\Rightarrow\ R\Big),\\
\text{QBFSat} &= \mu R.\ \big(\text{TrueClause} \vee \text{ExistsAssign} \vee \text{ForallAssign}\big).
\end{aligned}$$

Informally, the formula QBFSat above, using the formulae ExistsAssign and ForallAssign, removes at each recursive step the substring encoding the first quantifier that has not been dealt with yet, and a few substrings encoding variables that are bound by that quantifier. The substrings encoding variables that are removed at each step correspond to the ones that are assigned the truth value *false*. The difference between what ExistsAssign and ForallAssign express, is that the latter uses the connective '$\models\!\!\Rightarrow$' so that any such assignment of truth values to the variables bound by a universal quantifier will do. When no more quantifiers exist in the substring remaining, the formula TrueClause expresses that the assignment given is such that all clauses are satisfied. The recursion operator is necessary for checking that two bit strings are equal and to go through the arbitrarily many quantifier blocks of the Quantified Boolean Formula. Therefore, a string encoding a QBF instance, satisfies QBFSat if and only if the encoded Quantified Boolean Formula is true.

One can also define a formula expressing that a Quantified Boolean Formula encoded in the same way on a string, is not satisfiable, and therefore also define a formula saying that it is valid. Note that in the formula ForallAssign the recursion variable R is under an even number of negations.

4.2.2 Conjunctive Grammars and their Boolean Closure

In this section a brief comparison will be made between the word languages definable in GL_μ, and various classes of formal languages. The simplest of these classes is the class of regular languages, which was shown in [DGG07] to be exactly the languages definable in GL. The next important class of languages is the class of context-free ones which is strictly larger than the regular languages. A famous example for separating the two classes, through the use of the Pumping Lemma, is the language $\{a^n b^n \mid a, b \in \Sigma\}$.

Informally, a language is context-free if it can be generated by a context-free grammar, which is a set of rewriting rules where non-terminal symbols can be replaced by strings of terminal and non-terminal symbols according to the rules of the given grammar. A class that extends the context-free languages is the class of context-sensitive languages, where as the name implies, the rules applicable to any non-terminal symbol depend also on the context in which this symbol appears. We refer the reader to [RS97] for further details.

Various different other classes of languages have been defined that lie between the context-free and the context-sensitive ones, such as the class of conjunctive grammars, the boolean grammars and the deterministic context-sensitive ones. Conjunctive grammars, defined below, were introduced by Okhotin (see [Okh01],[Okh03]) and extend the rules of context-free grammars, by allowing the use of conjunction. An example language given in [Okh01] that separates conjunctive grammars from context-free grammars, is the language $\{wcw \mid w \in \Sigma^*, c \in \Sigma\}$, which can be shown to be undefinable by a context-free grammar using the Pumping Lemma for this class of languages. It is worth noting that in [AK08], Aizikowitz et al. introduce a variant of alternating pushdown automata that accept the same class of languages as the ones that can be derived by conjunctive grammars.

Boolean grammars make use of all boolean connectives, and it is because of negation that the semantics of boolean grammars are defined according to language equations and not rewriting terms as is done with the ones mentioned earlier. Boolean grammars are at least as expressive as the conjunctive ones, but a strict inclusion is not yet known. A thorough introduction in Boolean grammars is given in [Okh04].

The data complexity of both the Boolean Grammars and of the Conjunctive Grammars is PTIME, whereas for the case of the context-sensitive ones, PSPACE-complete problems can be expressed. In particular, the data complexity of context-sensitive grammars is exactly NLINSPACE, and therefore every language accepted by a non-deterministic linear space Turing machine is accepted by some context-sensitive grammar. A subclass of context-sensitive languages, named deterministic context-sensitive, is defined to be exactly the languages accepted by some determin-

istic linear space Turing machine. Although this class is included in the class of context-sensitive languages by definition, the inclusion is not known to be strict. As was shown in the previous section, the languages definable in GL_μ are also PSPACE-complete. A summary of the inclusions between the above languages is shown below:

$$L(\mathrm{reg}) \subsetneq L(\mathrm{CF}) \subsetneq L(\mathrm{ConjCF}) \subseteq L(\mathrm{Bool}) \subseteq L(\mathrm{DetCS}) \subseteq L(\mathrm{CS}),$$

where $L(\mathrm{reg})$ is the class of regular languages, CF is shorthand for context-free and similarly CS is for context-sensitive. Similarly, $L(\mathrm{ConjCF})$ and $L(\mathrm{Bool})$, denote the class of conjunctive grammar languages and Boolean grammar languages respectively. The class of languages $L(\mathrm{DetCS})$ lies between $L(\mathrm{Bool})$ and $L(\mathrm{CS})$ with none of the two inclusions known to be strict. The algorithm for model-checking GL_μ given in [DGG07] uses non-linear space for any fixed GL_μ formula, and so it may be the case that $L(\mathrm{GL}_\mu)$ can define languages that are not contained in $L(\mathrm{DetCS})$ or $L(\mathrm{CS})$.

In this section, a short introduction is given for these grammars and it is shown that any language generated by a conjunctive grammar can be defined in GL_μ, which is also presented in [DGG07]. Hence all languages generated by context-free grammars can also be defined in GL_μ. In particular, since GL_μ is closed under boolean operations, the boolean closure of conjunctive grammars can be defined in GL_μ. The boolean closure of conjunctive grammars is not necessarily the same as the boolean grammars. Furthermore, it has to be noted that the class of languages definable by conjunctive grammars is not known to be closed under complementation. If this class is not closed under complementation then conjunctive grammars are strictly included in the boolean grammars, as the latter class is closed under negation. The language $\{ww \mid w \in \{a,b\}^*\}$ is not known to be definable by a conjunctive grammar, whereas the complement of this language is actually context-free, as shown in [Sip05]. In particular, the language $L_1 = \{x \in \{a,b\}^* \mid x \text{ not of the form } ww\}$, is the union of the words of odd length together with the words of the form $x^i a x^i x^j b x^j$ or $x^i b x^i x^j a x^j$, for $x \in \{a,b\}$, which are both context-free languages. It is easily shown to be GL_μ definable as well, using the following formula.

$$\begin{aligned} \mathrm{Equal} &= \mu R.\ \Big(\epsilon \vee \exists x,y,z,w\ \big(\mathrm{first}(x) \wedge \mathrm{first}(z) \wedge \textstyle\bigvee_{\alpha\in\Sigma}(E_\alpha(x,y) \mid E_\alpha(z,w) \mid R)\big)\Big), \\ \varphi_{ww} &= \exists x,y,z,w,\ \mathrm{first}(x) \wedge \textstyle\bigvee_{\alpha\in\Sigma}(E_\alpha(x,y) \mid E_\alpha(z,w) \mid \mathrm{Equal}). \end{aligned}$$

The formula φ_{ww} expresses that there are two edges such that the two substrings obtained when disregarding these two edges, are equal with respect to both size and labels. The latter is expressed by Equal. The following sequence of inclusions illustrates where GL_μ lies with respect to the above languages.

$$L(\mathrm{reg}) \subsetneq L(\mathrm{CF}) \subsetneq L(\mathrm{ConjCF}) \subseteq L(\mathrm{BoolClosConjCF}) \subseteq \left\{ \begin{array}{c} L(\mathrm{GL}_\mu) \\ L(\mathrm{Bool}) \subseteq L(\mathrm{DetCS}) \subseteq L(\mathrm{CS}) \end{array} \right\},$$

where $L(\mathrm{BoolClosConjCF})$ denotes the boolean closure of conjunctive grammars. Neither direction of inclusion is known between $L(\mathrm{GL}_\mu)$ and the languages $L(\mathrm{Bool})$, $L(\mathrm{DetCS})$ and $L(\mathrm{CS})$.

Definition 4.2.2 (**Context-free Grammar**,[RS97]). *A context-free grammar G is a tuple (Σ, N, S, P) where Σ is the set of terminal symbols, N is the set of non-terminal symbols, S is an element of N and P is a set of production rules of the form $N \to (\Sigma \cup N)^*$.*

The relation of derivability, $\stackrel{G}{\Rightarrow}$, of a grammar G is defined as follows. For all sets of strings $s_1 A s_2$ where $s_1, s_2 \in (\Sigma \cup N)^*$, and for all P of the form $A \to w$ with $w \in (\Sigma \cup N)^*$ and $A \in N$, then $s_1 A s_2 \stackrel{G}{\Rightarrow} s_1 w s_2$ holds. The relation $\stackrel{G}{\Rightarrow}^*$ is the reflexive and transitive closure of $\stackrel{G}{\Rightarrow}$.

A language L is context-free if there exists a context-free grammar G such that for any word $w \in \Sigma^*$ it is the case that $w \in L$ if and only if $S \stackrel{G}{\Rightarrow}^* w$, where S is a distinguished non-terminal acting as the initial one.

Definition 4.2.3 (**Conjunctive Grammar**, [Okh01]). *A conjunctive grammar is a quadruple (Σ, N, S, P), where Σ is the set of terminal symbols, N is the set of non-terminal symbols and S is an element of N representing the start symbol. P is a set of production rules in the form of ordered pairs $(A, \{\alpha_1, \ldots, \alpha_n\})$, respectively of a non-terminal A and a finite subset of $(\Sigma \cup N)^*$. The rules are written in the form $A \to \alpha_1 \& \ldots \& \alpha_n$.*

A conjunctive grammar formula is defined to be an element of $(\Sigma \cup N \cup \{(,), \&\})^*$ and is such that ϵ is a formula, every symbol in $\Sigma \cup N$ is a formula, if s_1, s_2 are formulae then $s_1 s_2$ is a formula, and if $s_1, \ldots, s_n$ are formulae then $(s_1 \& \ldots \& s_n)$ is a formula. The relation of derivability of a grammar G is defined as follows. For all $s_1, s_2 \in (\Sigma \cup N \cup \{(,), \&\})^*$ and for all $A \in N$, if $s_1 A s_2$ is a conjunctive grammar formula, then for all rules of the form $A \to \alpha_1 \& \ldots \& \alpha_n$ in P, it is the case that $s_1 A s_2 \stackrel{G}{\Rightarrow} s_1(\alpha_1 \& \ldots \& \alpha_n) s_2$. Furthermore, for all $s_1, s_2 \in (\Sigma \cup N \cup \{(,), \&\})^*$ and for all $w \in \Sigma^*$, if $s_1(w \& \ldots \& w) s_2$ is a formula then it holds that $s_1(w \& \ldots \& w) s_2 \stackrel{G}{\Rightarrow} s_1 w s_2$.

The relation $\stackrel{G}{\Rightarrow}^*$ is the reflexive and transitive closure of $\stackrel{G}{\Rightarrow}$. For any language L, $L \in L(\text{ConjCF})$ if and only if there exists a conjunctive grammar $G = (\Sigma, N, S, P)$, such that for any word $w \in \Sigma^*$, $w \in L$ if and only if $S \stackrel{G}{\Rightarrow}^* w$. Note that context-free grammars are subsumed by conjunctive grammars since the syntax and the semantics of the latter extend the syntax and semantics respectively of the context-free ones.

An example of a language definable by some conjunctive grammar but no context-free grammar is the language $\{a^n b^n c^n \mid a, b, c \in \Sigma, n \in \mathbb{N}\}$, as is shown in [Okh01]. This is accomplished using the grammar $G = (\Sigma, N, S, P)$, where $N = \{A, C, D, F, H, J, S\}$ and where P contains the following rules.

$$
\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl}
S & \to & F \& D, & D & \to & AJ, & A & \to & \epsilon, \\
F & \to & HC, & J & \to & bJc, & C & \to & Cc, \\
H & \to & aHb, & J & \to & \epsilon, & C & \to & \epsilon. \\
H & \to & \epsilon, & A & \to & Aa, & & &
\end{array}
$$

According to the grammar G, the non-terminal F produces exactly those words of the form $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}$ and similarly D produces the set of words $\{a^n b^m c^m \mid n, m \in \mathbb{N}\}$. Notice

that these two languages are context-free, as are the rules of the grammar G defining them. The rule $S \to F\&D$ simply says that a word has to be derivable from both F and D, or in other words be in the language $L = \{a^n b^n c^n \mid a, b, c \in \Sigma, n \in \mathbb{N}\}$. Using a pumping lemma argument, it can be shown that L is not context-free.

In [Mor89] another class of grammars, named alternating context-free, is defined which is different from the class of grammars defined in [Lan02] with the same name. The latter are essentially the same as conjunctive grammars, whose class of languages is in PTIME, whereas the class of languages defined by the alternating context-free grammars of [Mor89] is in EXPTIME. We refer the reader to [Mor89] for an introduction to the latter grammars.

Fixed Point Logic with Chop (FLC for short) is introduced and studied in [MO99] and is an extension of the modal μ-calculus L_μ, with a sequential composition operator added. The logic FLC is strictly more expressive than L_μ over words, as it can define languages that are not even context-free. Linear Fixed Point Logic with Chop (LFLC), studied in [Lan02], is an extension of linear modal μ-calculus, with the same sequential composition operator as FLC, where linear modal μ-calculus, is the restriction of modal μ-calculus on words. It has been proved in [Lan02] that LFLC over words is equi-expressive to the alternating context-free grammars defined in the same paper, which, as stated earlier, are essentially the same as the conjunctive grammars.

Since GL_μ is closed under all boolean operations, in order to show that $L(\mathrm{BoolClosConjCF}) \subseteq L(\mathrm{GL}_\mu)$, it is sufficient to show that $L(\mathrm{ConjCF}) \subseteq L(\mathrm{GL}_\mu)$. The latter was proved in [DGG07] and it is shown here using a translation from Linear Fixed Point Logic with Chop.

Definition 4.2.4 (**LFLC Formulae**,[Lan02])**.** *Let Σ be an alphabet and let $\boldsymbol{Var}$ be a set of variables. If $a \in \Sigma$, $Z \in \boldsymbol{Var}$ and φ, ψ are* LFLC *formulae, then so are the following:*

$$\chi = \boldsymbol{tt} \mid \boldsymbol{ff} \mid a \mid Z \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \mu Z.\varphi_1 \mid \nu Z.\varphi_1 \mid \varphi; \psi.$$

Notice that for any formula $\mu Z.\varphi_1$, the recursion variable Z always appears positive in φ_1, namely under an even number of negations, since negation is not included in the syntax of LFLC. Therefore the fixed point operator defined by φ_1 is monotone.

Definition 4.2.5 (**LFLC Semantics**, [Lan02])**.** *The semantics of a* LFLC *formula, where $\rho : \boldsymbol{Var} \to \wp(\Sigma^*)$ is the environment interpreting the recursion variables, is defined inductively as follows.*

$$\begin{aligned}
[\![\boldsymbol{tt}]\!]_\rho &\equiv \Sigma^*, \\
[\![\boldsymbol{ff}]\!]_\rho &\equiv \emptyset, \\
[\![a]\!]_\rho &\equiv \{a\}, \\
[\![Z]\!]_\rho &\equiv \rho(Z), \\
[\![\varphi \vee \psi]\!]_\rho &\equiv [\![\varphi]\!]_\rho \cup [\![\psi]\!]_\rho, \\
[\![\varphi \wedge \psi]\!]_\rho &\equiv [\![\varphi]\!]_\rho \cap [\![\psi]\!]_\rho, \\
[\![\mu Z.\varphi]\!]_\rho &\equiv \bigcap \{V \subseteq \Sigma^* \mid [\![\varphi]\!]_{\rho[Z \mapsto V]} \subseteq V\}, \\
[\![\nu Z.\varphi]\!]_\rho &\equiv \bigcup \{V \subseteq \Sigma^* \mid V \subseteq [\![\varphi]\!]_{\rho[Z \mapsto V]}\}, \\
[\![\varphi; \psi]\!]_\rho &\equiv \{v \in \Sigma^* \mid \exists v_1, v_2 \in \Sigma^*,\ s.t.\ v = v_1 v_2, v_1 \in [\![\varphi]\!]_\rho, v_2 \in [\![\psi]\!]_\rho\}.
\end{aligned}$$

Although negation is not included in the syntax of LFLC, any modal μ-calculus formula, negated or not, is equivalent to a formula of LFLC, since all boolean constructs are included in the syntax as well as both the least and greatest fixed point operators, and for any $a \in \Sigma$, $\neg a$ is equivalent to $\bigvee_{b \in \Sigma, b \neq a} b$. In other words, the negation outside a modal μ-calculus formula can be pushed inside preserving the fact that each recursion variable Z appears positive in the formula $\mu Z.\varphi_1$ or $\nu Z.\varphi_1$ where Z gets bound.

A direct translation from LFLC to LFP over words will be given, as well as one from LFLC to GL_μ. The proofs for the correctness of the translations make use of the following three useful lemmas about LFLC.

It should be noted that LFP over words captures PTIME, and therefore a direct translation is not required for Theorem 4.2.9 below, since any LFLC formula can be evaluated in polynomial time. The difficulty of directly translating LFLC formulae into LFP ones is that the recursion in LFLC works over potentially infinite lattices, whereas in LFP the recursion variables are always mapped to tuples of elements of the structure that the LFP recursion operator is working on. Lemma 4.2.7, essentially states that to see whether a word w satisfies some LFLC formula it is enough to consider the set of subwords of w as a lattice in the recursion.

Lemma 4.2.6. *For all sets $V_1, V_2 \in \wp(\Sigma^*)$ such that $V_1 \subseteq V_2$, and for all* LFLC *formulae φ and environments ρ it is the case that $[\![\varphi]\!]_{\rho[Z \mapsto V_1]} \subseteq [\![\varphi]\!]_{\rho[Z \mapsto V_2]}$.*

Proof. It follows from the monotonicity of the formula. □

Lemma 4.2.7. *For any word w and any* LFLC *formula φ and environment ρ, it holds that $[\![\varphi]\!]_{\rho[Z \mapsto V]} \cap S = [\![\varphi]\!]_{\rho[Z \mapsto V \cap S]} \cap S$, where S is the set of all subwords of w.*

Proof. The Lemma essentially states that for any subword w_i of any word w, and for any LFLC formula φ and environment ρ, it holds that $w_i \in [\![\varphi]\!]_{\rho[Z \mapsto V]} \Leftrightarrow w_i \in [\![\varphi]\!]_{\rho[Z \mapsto V \cap S]}$, where S is the set of all subwords of w. The direction $w_i \in [\![\varphi]\!]_{\rho[Z \mapsto V]} \Leftarrow w_i \in [\![\varphi]\!]_{\rho[Z \mapsto V \cap S]}$ follows from Lemma 4.2.6. We proceed on the proof of the opposite direction by induction on the structure of the formula. For what follows, w_i denotes a subword of a word w, and we let $\rho_1 = \rho[Z \mapsto V]$ and $\rho_2 = \rho[Z \mapsto V \cap S]$. We want to show that $w_i \in [\![\varphi]\!]_{\rho_1} \Rightarrow w_i \in [\![\varphi]\!]_{\rho_2}$.

$\varphi =$ a for some $a \in \Sigma$. Assume $w_i \in [\![a]\!]_{\rho_1}$. Then $w_i \in [\![a]\!]_{\rho'}$ for any environment ρ'.

$\varphi =$ Z' for some $Z' \in \mathbf{Var}$. Suppose first that $Z' \neq Z$ and let $w_i \in [\![\varphi]\!]_{\rho_1}$. Then $w_i \in \rho_1(Z')$ and ρ_1 agrees with ρ_2 on recursion variables other than Z and therefore $w_i \in [\![\varphi]\!]_{\rho_2}$.

Similarly suppose that $Z' = Z$ and let $w_i \in [\![\varphi]\!]_{\rho_1}$. By the semantics definition, $w_i \in \rho_1(Z) = V$ and therefore, since $w_i \in S$, it is also the case that $w_i \in \rho_2(Z) = V \cap S$. Hence $w_i \in [\![Z]\!]_{\rho_2}$.

$\varphi =$ $\varphi_1 \wedge \varphi_2$. Let $w_i \in [\![\varphi]\!]_{\rho_1}$. It follows that $w_i \in [\![\varphi_1]\!]_{\rho_1} \cap [\![\varphi_2]\!]_{\rho_1}$, and therefore by the inductive hypothesis $w_i \in [\![\varphi_1]\!]_{\rho_2} \cap [\![\varphi_2]\!]_{\rho_2}$, which means that $w_i \in [\![\varphi]\!]_{\rho_2}$.

$\varphi = \varphi_1 \vee \varphi_2$. Similar case to the one above.

$\varphi = \varphi_1;\varphi_2$. Let $w_i \in [\![\varphi_1;\varphi_2]\!]_{\rho_1}$. Then by the definition of the semantics, $w_i = v_1v_2$, and for $j \in \{1,2\}$ it holds that $v_j \in [\![\varphi_j]\!]_{\rho_1}$. By the inductive hypothesis, and since S is closed under subwords, $[\![\varphi_j]\!]_{\rho_1} \cap S = [\![\varphi_j]\!]_{\rho_2} \cap S$, for $j \in \{1,2\}$, and since $v_j \in S$ and $v_j \in [\![\varphi_j]\!]_{\rho_1}$, it holds that $v_j \in [\![\varphi_j]\!]_{\rho_2}$. Therefore $w_i \in \{v \in \Sigma^* \mid \exists v_1, v_2 \in \Sigma^*, \text{ s.t. } v = v_1v_2, v_1 \in [\![\varphi_1]\!]_{\rho_2}, v_2 \in [\![\varphi_2]\!]_{\rho_2}\}$. Hence $w_i \in [\![\varphi]\!]_{\rho_2}$.

$\varphi = \mu Z'.\psi$. Assume first that $Z' \neq Z$ and that $w_i \in [\![\mu Z'.\psi]\!]_{\rho_1}$. For the following we define F_ℓ to be the function mapping V' to $[\![\psi]\!]_{\rho_\ell[Z' \mapsto V']}$, for $\ell = 1,2$. Note that the inductive hypothesis is that for any ρ', Z', V', $[\![\psi]\!]_{\rho'[Z' \mapsto V']} \cap S = [\![\psi]\!]_{\rho'[Z' \mapsto V' \cap S]} \cap S$. According to this, the following hold by the inductive hypothesis.

$$\begin{aligned} [\![\psi]\!]_{\rho_1[Z' \mapsto V']} \cap S &= [\![\psi]\!]_{\rho_1[Z' \mapsto V' \cap S]} \cap S, \\ [\![\psi]\!]_{\rho_2[Z' \mapsto V']} \cap S &= [\![\psi]\!]_{\rho_2[Z' \mapsto V' \cap S]} \cap S, \\ [\![\psi]\!]_{\rho_3[Z \mapsto V]} \cap S &= [\![\psi]\!]_{\rho_3[Z \mapsto V \cap S]} \cap S, \end{aligned}$$

where $\rho_3 = \rho[Z' \mapsto V']$. Notice that $\rho_3[Z \mapsto V] = \rho[Z' \mapsto V'][Z \mapsto V] = \rho_1[Z' \mapsto V']$, and similarly $\rho_3[Z \mapsto V \cap S] = \rho_2[Z' \mapsto V']$. Therefore using the F_ℓ notation we get:

$$\begin{aligned} F_1(V') \cap S &= F_1(V' \cap S) \cap S, \\ F_2(V') \cap S &= F_2(V' \cap S) \cap S, \\ F_1(V') \cap S &= F_2(V') \cap S. \end{aligned} \tag{4.1}$$

We want to show that $[\![\mu Z'.\psi]\!]_{\rho_1} \cap S \subseteq [\![\mu Z'.\psi]\!]_{\rho_2} \cap S$. It is enough to show then that for every V_2 where $F_2(V_2) \subseteq V_2$, there exists V_1 such that $F_1(V_1) \subseteq V_1$ and $V_1 \cap S = V_2 \cap S$. Let V_2 be any set where $F_2(V_2) \subseteq V_2$. We show that letting V_1 to be equal to $V^\infty = F_1^\infty(V_2)$ satisfies the condition. By definition V^∞ is in $\{V_1 \mid F_1(V_1) \subseteq V_1\}$. We hence only need to show that $V^\infty \cap S = V_2 \cap S$.

We proceed by induction on the number of recursion steps showing that for all ordinals k, $F_1^k(V_2) \cap S = V_2 \cap S$. For $k = 0$, we have that $V_2 \cap S = V_2 \cap S$.

For the case of the successor ordinal assume that for some j, $F_1^j(V_2) \cap S = V_2 \cap S$, and consider $F_1^{j+1}(V_2) \cap S$. This is equal to $F_1(F_1^j(V_2)) \cap S$ and by equation 4.1 this is equal to $F_1(F_1^j(V_2) \cap S) \cap S$. By the inductive hypothesis, $F_1(F_1^j(V_2) \cap S) \cap S = F_1(V_2 \cap S) \cap S$. Finally, equation 4.1 entails that the latter is equal to $F_1(V_2) \cap S = F_2(V_2) \cap S$, which is equal to $V_2 \cap S$, since by assumption V_2 is a fixed point of F_2.

Finally, assume the statement holds for all $j < \lambda$, where λ is a limit ordinal. Then $F^\lambda(V_2) \cap S = (\bigcup_{j<\lambda} F^j(V_2)) \cap S = \bigcup_{j<\lambda}(F^j(V_2) \cap S)$ which by the inductive hypothesis is equal to $\bigcup_{j<\lambda}(V_2 \cap S) = V_2 \cap S$. This completes the inductive proof.

Assume then that $Z' = Z$. We need to show that $[\![\mu Z.\psi]\!]_{\rho_1} \cap S \subseteq [\![\mu Z.\psi]\!]_{\rho_2} \cap S$. In other words, by definition of the semantics for $\mu Z.\psi$, we need to show that

$$\bigcap\{V \mid [\![\psi]\!]_{\rho_1[Z \mapsto V]} \subseteq V\} \cap S \subseteq \bigcap\{V \mid [\![\psi]\!]_{\rho_2[Z \mapsto V]} \subseteq V\} \cap S.$$

But these two sets are the same, since $\rho_1[Z \mapsto V] = \rho_2[Z \mapsto V] = \rho[Z \mapsto V]$.

$\varphi = \nu Z'.\psi$. Let first $Z' \neq Z$ and assume $w_i \in [\![\nu Z'.\psi]\!]_{\rho_1}$. By definition $w_i \in \bigcup\{V' \subseteq \Sigma^* \mid V' \subseteq [\![\psi]\!]_{\rho_1[Z' \mapsto V']}\}$. We have to show that for any $V' \in \{V' \mid V' \subseteq [\![\psi]\!]_{\rho_1[Z' \mapsto V']}\}$ there exists a V'' in $\{V'' \mid V'' \subseteq [\![\psi]\!]_{\rho_2[Z' \mapsto V'']}\}$, such that $V' \cap S = V'' \cap S$.

Let $V' \subseteq [\![\psi]\!]_{\rho_1[Z' \mapsto V']} = F_1(V')$. Letting V'' be equal to $F_2^{\infty}(V')$ and using a similar induction as the one used in the previous case, where $\varphi = \mu Z'.\psi$, we get that $V'' \cap S = V' \cap S$, and that $V'' \subseteq [\![\psi]\!]_{\rho_2[Z' \mapsto V'']}$.

The argument for the case where $Z' = Z$, is similar to the one given earlier for $\varphi = \mu Z'.\psi$.

□

Lemma 4.2.8. *Let w be some word and let S be the set of subwords of w. Then for every $V_1 \subseteq \Sigma^*$ and any* LFLC *formula φ and environment ρ, such that $[\![\varphi]\!]_{\rho[Z \mapsto V_1]} \cap S = V_1 \cap S$, there exists $V_2 \subseteq \Sigma^*$, such that $V_1 \cap S = V_2 \cap S$ and $[\![\varphi]\!]_{\rho[Z \mapsto V_2]} = V_2$.*

Proof. Fix w, φ and ρ, and let V_1 be some subset of Σ^*, such that $[\![\varphi]\!]_{\rho[Z \mapsto V_1]} \cap S = V_1 \cap S$. For convenience we will denote with F_φ the function that maps a subset V of Σ^* to the set $[\![\varphi]\!]_{\rho[Z \mapsto V]}$ for the fixed environment ρ. Note that F_φ is monotone and let V^∞ be such that for some k, $V^\infty = F_\varphi^k(V_1) = F_\varphi^{k+1}(V_1)$, where $F_\varphi^k(V_1) = F(F_\varphi^{k-1}(V_1))$ and $F_\varphi^1(V_1) = F_\varphi(V_1)$.

By definition, $F_\varphi(V^\infty) = V^\infty$ and therefore $[\![\varphi]\!]_{\rho[Z \mapsto V^\infty]} = V^\infty$. It remains to be shown that $V^\infty \cap S = V_1 \cap S$. Defining V_2 to be equal to V^∞ is sufficient for the Lemma to hold.

We prove by induction on k that $F_\varphi^k(V_1) \cap S = V_1 \cap S$. For $k = 1$, we have that $F_\varphi(V_1) \cap S = V_1 \cap S$ by the assumption of the statement. Suppose that the statement holds for some j, namely $F_\varphi^j(V_1) \cap S = V_1 \cap S$, and consider $F_\varphi(F_\varphi^j(V_1)) \cap S$. This is equal to $F_\varphi\big(F_\varphi^j(V_1) \cap S\big) \cap S$ by Lemma 4.2.7, which is then equal to $F_\varphi(V_1 \cap S) \cap S$ by the inductive hypothesis. Again using the Lemma 4.2.7, we have that $F_\varphi(V_1 \cap S) \cap S = F_\varphi(V_1) \cap S$ which is equal to $V_1 \cap S$ by the assumption of the statement.

Finally, suppose the statement holds for all $j < \lambda$, where is λ is some limit ordinal. Then $F^\lambda(V_1) \cap S = (\bigcup_{j<\lambda} F^j(V_1)) \cap S = \bigcup_{j<\lambda}(F^j(V_1) \cap S)$ which by the inductive hypothesis is equal to $\bigcup_{j<\lambda}(V_1 \cap S) = V_1 \cap S$. □

The above three Lemmas will assist in proving the correctness of the two translations from LFLC to LFP and GL_μ. The inductive translation from LFLC to LFP is given first. The second translation and its proof is along similar lines. In the following, for any LFLC formula φ, its translation into an LFP formula is denoted by $[\varphi]_T$.

A word w satisfies an LFLC formula φ if and only if it satisfies the LFP formula $\exists x, y\ \mathrm{first}(x) \wedge \mathrm{last}(y) \wedge [\varphi]_T(x, y)$. A word structure for this case in LFP is represented as a structure with a binary relation for each letter in the alphabet, as is done for words in GL_μ. More formally, for an alphabet $\Sigma = \{d_1, \ldots, d_n\}$ a word structure is defined as $\mathfrak{W} = (A, R_{d_1}, \ldots, R_{d_n})$ where A is the universe, and each R_{d_i} is binary. In what follows, for a word $\mathfrak{W}$ with vertices a and b, $a \prec b$

expresses that a precedes b in the word $\mathfrak{W}$. Notice that this property is expressible in LFP. The translation of LFLC formulae to LFP ones is given below.

$$\begin{array}{lcl}
[d]_T(x,y) & = & R_d(x,y), \\
[Z]_T(x,y) & = & S_Z(x,y), \\
[\varphi_1 \wedge \varphi_2]_T(x,y) & = & [\varphi_1]_T(x,y) \wedge [\varphi_2]_T(x,y), \\
[\varphi_1 \vee \varphi_2]_T(x,y) & = & [\varphi_1]_T(x,y) \vee [\varphi_2]_T(x,y), \\
[\varphi_1;\varphi_2]_T(x,y) & = & \exists z\ ([\varphi_1]_T(x,z) \wedge [\varphi_2]_T(z,y)), \\
[\mu Z.\varphi]_T(x,y) & = & [\mathbf{lfp}_{S_Z,x,y}([\varphi]_T \wedge x \prec y)(S_Z,x,y)](x,y), \\
[\nu Z.\varphi]_T(x,y) & = & [\mathbf{gfp}_{S_Z,x,y}([\varphi]_T \wedge x \prec y)(S_Z,x,y)](x,y).
\end{array}$$

In the following, for each LFLC word w, we define $\mathfrak{W} = (A, R_{d_1}, \ldots, R_{d_n})$ with universe A, to be the word structure in LFP corresponding to w and if $a, b \in A$, are two vertices in $\mathfrak{W}$ then $w_{a,b}$ corresponds to the subword of $\mathfrak{W}$ that starts at a and ends at b. Furthermore, for any environment $\rho : \mathbf{Var} \to \wp(\Sigma^*)$ interpreting recursion variables, we define ρ' to be the function assigning subsets of A^2 to second order variables, such that for any $a, b \in A$, and any second order variable S_Z, $(a,b) \in \rho'(S_Z)$ if and only if $w_{a,b} \in \rho(Z)$.

Theorem 4.2.9. *Let φ be any* LFLC *formula, w any* LFLC *word, $\rho : \boldsymbol{Var} \to \wp(\Sigma^*)$ any environment interpreting recursion variables and let $\mathfrak{W}$ be the word structure for* LFP *with universe A corresponding to w, and ρ' the assignment corresponding to ρ. Then for any $a, b \in A$, $(\mathfrak{W}, \rho') \models_{\text{LFP}} [\varphi]_T(a,b)$ if and only if for the subword $w_{a,b}$ of w, it holds that $(w_{a,b}, \rho) \models_{\text{LFLC}} \varphi$.*

Proof. We proceed by induction on the structure of the formula.

$\varphi =$ d for some $d \in \Sigma$. It is clear that for any a, b it holds that $(\mathfrak{W}, \rho') \models R_d(a,b)$ if and only if $w_{a,b}$ is simply the one letter word d.

$\varphi =$ Z for some $Z \in \mathbf{Var}$. Then for any vertices a, b, $(\mathfrak{W}, \rho') \models S_Z(a,b)$ if and only if $(a,b) \in \rho'(S_Z)$ if and only if $w_{a,b} \in \rho(Z)$.

$\varphi =$ $\varphi_1 \wedge \varphi_2$. Then for any a, b, $(w_{a,b}, \rho) \models \varphi_1 \wedge \varphi_2$ if and only if $(w_{a,b}, \rho) \models \varphi_1$ and $(w_{a,b}, \rho) \models \varphi_2$. By the inductive hypothesis, this holds if and only if $(\mathfrak{W}, \rho') \models [\varphi_1]_T(a,b)$ and $(\mathfrak{W}, \rho') \models [\varphi_2]_T(a,b)$. The required result follows.

$\varphi =$ $\varphi_1 \vee \varphi_2$. The case is similar to the one above.

$\varphi =$ $\varphi_1;\varphi_2$. Then by definition of the semantics, for any a, b, $(w_{a,b}, \rho) \models \varphi_1;\varphi_2$ if and only if $w_{a,b} \in \{v \in \Sigma^* \mid \exists v_1, v_2 \in \Sigma^*, \text{ s.t. } v = v_1 v_2,\ v_1 \in [\![\varphi_1]\!]_\rho, v_2 \in [\![\varphi_2]\!]_\rho\}$. Consider first the *only if* direction. We want to prove that $(\mathfrak{W}, \rho') \models [\varphi]_T(a,b) \Rightarrow (w_{a,b}, \rho) \models \varphi$. Assume $(\mathfrak{W}, \rho') \models \exists z\ ([\varphi_1]_T(a,z) \wedge [\varphi_2]_T(z,b))$. Let c be such that $(\mathfrak{W}, \rho') \models [\varphi_1]_T(a,c) \wedge [\varphi_2]_T(c,b)$. By the inductive hypothesis we know that the following hold.

$$\begin{array}{rl}
 & (\mathfrak{W}, \rho') \models [\varphi_1]_T(a,c) \Leftrightarrow (w_{a,c}, \rho) \models \varphi_1 \\
\text{and} & (\mathfrak{W}, \rho') \models [\varphi_2]_T(c,b) \Leftrightarrow (w_{c,b}, \rho) \models \varphi_2.
\end{array}$$

Then it is the case that $w_{a,b}$ satisfies φ, by the definition of the semantics shown above, since $w_{a,b} = w_{a,c}w_{c,b}$.

For the *if* direction, assuming $(w_{a,b}, \rho) \models \varphi$, we want to show that $(\mathfrak{W}, \rho') \models [\varphi]_T(a, b)$. Then by the definition of the semantics there are v_1, v_2, such that $w_{a,b} = v_1v_2$ and $(v_j, \rho) \models \varphi_j$, for $j \in \{1, 2\}$. If there are such v_1, v_2 then there exists c in $w_{a,b}$ such that $w_{a,c} = v_1$ and $w_{c,b} = v_2$. Therefore, by the inductive hypothesis, it holds that $(\mathfrak{W}, \rho') \models [\varphi_1]_T(a, c)$ and $(\mathfrak{W}, \rho') \models [\varphi_2]_T(c, b)$. The required result then follows.

$\varphi = \mu Z.\psi$. By definition, it holds that for any a, b, $(w_{a,b}, \rho) \models \mu Z.\psi$ if and only if $w_{a,b} \in \bigcap\{V \subseteq \Sigma^* \mid [\![\psi]\!]_{\rho[Z\mapsto V]} \subseteq V\}$. It needs to be shown that this is the case if and only if $(\mathfrak{W}, \rho') \models [\mathbf{lfp}_{S_Z,x,y}([\psi]_T \wedge x \prec y)(S_Z, x, y)](a, b)$. By Lemma 4.2.7 we have that for all Z, V, w and any subword w_i of w, it holds that $w_i \in [\![\psi]\!]_{\rho[Z\mapsto V]} \Leftrightarrow w_i \in [\![\psi]\!]_{\rho[Z\mapsto V\cap S]}$, where S is the set of subwords of w. By monotonicity of the mapping $[\![\psi]\!]_{\rho[Z\mapsto V]}$, the equation below holds.

$$\begin{aligned} &w_{a,b} \in \bigcap\{V \mid [\![\psi]\!]_{\rho[Z\mapsto V]} \subseteq V\} && \Leftrightarrow \\ &w_{a,b} \in \bigcap\{V \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\} && \Leftrightarrow \\ &w_{a,b} \in S \cap \bigcap\{V \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\} && \Leftrightarrow \\ &w_{a,b} \in \bigcap\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\}. \end{aligned}$$

Now, $\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\} \subseteq \{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S = V \cap S\}$, since every V in the first set, also belongs to the second set. So clearly if $w_{a,b}$ is in the intersection of the V's of the second set, it is also in the intersection of the V's of the first set. Therefore it holds that

$$w_{a,b} \in \bigcap\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S = V \cap S\} \Rightarrow w_{a,b} \in \bigcap\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\}.$$

Consider the opposite direction and assume that $w_{a,b} \in \bigcap\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\}$. For contradiction assume that $w_{a,b} \notin \bigcap\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S = V \cap S\}$. Then there must be some V_1 in the second set that does not belong to the first one, such that $w_{a,b} \notin V_1$. We need to show that there exists a set $V_2 \in \{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\}$ such that $V_1 \cap S = V_2 \cap S$. If this holds, then $w_{a,b} \notin V_2$ and therefore $w_{a,b} \notin \bigcap\{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\}$, which is a contradiction.

It holds that $V_1 \in \{V \cap S \mid [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S = V \cap S\}$. Then $[\![\psi]\!]_{\rho[Z\mapsto V_1\cap S]} \cap S = V_1 \cap S$ by definition. From Lemma 4.2.7 we also have that $[\![\psi]\!]_{\rho[Z\mapsto V_1\cap S]} \cap S = [\![\psi]\!]_{\rho[Z\mapsto V_1]} \cap S$. Therefore, the set V_2 we are looking for, is the one given by Lemma 4.2.8. Thus

$$\begin{aligned} &w_{a,b} \in \bigcap\{V \mid [\![\psi]\!]_{\rho[Z\mapsto V]} = V\} \Leftrightarrow \\ &w_{a,b} \in \bigcap\{V \mid [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S = V \cap S\}. \end{aligned} \tag{4.2}$$

Therefore we need to show that $w_{a,b} \in \bigcap\{V \mid [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S = V \cap S\}$ if and only if $(\mathfrak{W}, \rho') \models [\mathbf{lfp}_{S_Z,x,y}([\psi]_T \wedge x \prec y)(S_Z, x, y)](a, b)$. By the inductive hypothesis and the above, it holds that, for any a, b,

$$w_{a,b} \in [\![\psi]\!]_{\rho[Z\mapsto V\cap S]} \cap S \Leftrightarrow (\mathfrak{W}, \rho'_V) \models [\psi]_T(S_Z, a, b), \tag{4.3}$$

where ρ'_V corresponds to $\rho[Z \mapsto V \cap S]$. Let W be any set of tuples (x, y) of positions in the word $\mathfrak{W}$ such that each tuple can be seen as denoting the first and last elements of a subword of $\mathfrak{W}$ and let F_ψ to be the function according to the mapping $W \mapsto \{(x, y) \mid (\mathfrak{W}, \rho_W) \models ([\psi]_T \wedge x \prec y)(S_Z, x, y), \rho_W = \rho'[S_Z \mapsto W]\}$. By the inductive hypothesis represented by the equation 4.3, we have that for any such set W, $(a, b) \in F_\psi(W)$ if and only if $w_{a,b} \in [\![\psi]\!]_{\rho[Z \mapsto V \cap S]} \cap S$, where $V \cap S$ is the set of subwords $w_{a,b}$ such that $(a, b) \in W$. Hence $F_\psi(W) = W$ if and only if $[\![\psi]\!]_{\rho[Z \mapsto V \cap S]} \cap S = V \cap S$. By the definition of the semantics of LFP we have the following.

$$\begin{array}{c}(\mathfrak{W}, \rho') \models [\mathbf{lfp}_{S_Z,x,y}([\psi]_T \wedge x \prec y)(S_Z, x, y)](a, b) \Leftrightarrow \\ (a, b) \in \mathbf{lfp}(F_\psi) = \bigcap\{W \mid F_\psi(W) = W\}.\end{array} \tag{4.4}$$

From the equations 4.2, 4.3 and 4.4 the required result holds, namely that $(\mathfrak{W}, \rho') \models [\mathbf{lfp}_{S_Z,x,y}([\psi]_T \wedge x \prec y)(S_Z, x, y)](a, b)$ if and only if $(w_{a,b}, \rho) \models \mu Z.\psi$.

$\varphi = \nu Z.\psi$. The case is similar to the one above. It can be shown, using Lemma 4.2.8, that for any $V_1 \in \{V \cap S \mid [\![\psi]\!]_{\rho[Z \mapsto V \cap S]} \cap S = V \cap S\}$, there exists a set $V_2 \in \{V \cap S \mid [\![\psi]\!]_{\rho[Z \mapsto V]} = V\}$ such that $V_1 \cap S = V_2 \cap S$ and therefore have that the following holds.

$$w_{a,b} \in \bigcup\{V \cap S \mid [\![\psi]\!]_{\rho[Z \mapsto V]} = V\} \Leftrightarrow w_{a,b} \in \bigcup\{V \cap S \mid [\![\psi]\!]_{\rho[Z \mapsto V \cap S]} \cap S = V \cap S\}.$$

□

An immediate consequence of Theorem 4.2.9 is the following Corollary, which establishes that there is a translation from LFLC formulae to LFP ones on words.

Corollary 4.2.10. *For any* LFLC *word* w *and its corresponding* LFP *word* $\mathfrak{W}$*, and any* LFLC *sentence* φ*,* $w \models_{\text{LFLC}} \varphi$ *if and only if* $\mathfrak{W} \models_{\text{LFP}} \exists x, y\ (\mathit{first}(x) \wedge \mathit{last}(y) \wedge [\varphi]_T(x, y))$.

In what follows we give an explicit translation from LFLC to GL_μ, which follows similar lines as the translation presented above, from LFLC to LFP. This translation however does not pose the same difficulty, as the recursion operator in both logics works over the infinite lattice of all words. For any LFLC formula ψ, the notation $[\psi]_\mu$ is used to denote the translation of this formula into a GL_μ one. The inductive translation is given below.

$$\begin{array}{lcl}
[a]_\mu & = & \exists x, y\ (\text{first}(x) \wedge \text{last}(y) \wedge \text{Connected} \wedge E_a(x, y)), \\
[Z]_\mu & = & \text{Connected} \wedge R_Z, \\
[\varphi_1 \wedge \varphi_2]_\mu & = & [\varphi_1]_\mu \wedge [\varphi_2]_\mu \wedge \text{Connected}, \\
[\varphi_1 \vee \varphi_2]_\mu & = & [\varphi_1]_\mu \vee [\varphi_2]_\mu \wedge \text{Connected}, \\
[\varphi_1; \varphi_2]_\mu & = & \exists x\ \text{first}(x) \wedge \left(([\varphi_1]_\mu \wedge \text{first}(x)) \mid [\varphi_2]_\mu\right) \wedge \text{Connected}, \\
[\mu Z.\psi]_\mu & = & \text{Connected} \wedge \mu R_Z.[\psi]_\mu, \\
[\nu Z.\psi]_\mu & = & \text{Connected} \wedge \nu R_Z.[\psi]_\mu.
\end{array}$$

In the translation above, the formulae of GL_μ, with the use of the formula Connected, are simulating the chop operator that splits a word in two connected subwords. In the following,

lowercase w will denote a word structure as it is represented in LFLC and uppercase W will denote a word structure for GL_μ. Similarly, if $\rho : \mathbf{Var} \to \wp(\Sigma^*)$ is an environment interpreting recursion variables in LFLC, then ρ' denotes the corresponding environment interpreting recursion variables in GL_μ, such that for any LFLC word w' and the corresponding GL_μ word W', $w' \in \rho(Z)$ if and only if $W' \in \rho'(R_Z)$.

Theorem 4.2.11. *For any* LFLC *formula* φ *and any word structures* w, W *that represent the same word, and any environment* ρ *interpreting variables in* LFLC *and its corresponding environment* ρ' *in* GL_μ*, it holds that* $(w, \rho) \models_{\mathrm{LFLC}} \varphi$ *if and only if* $(W, \rho') \models_{\mathrm{GL}_\mu} [\varphi]_\mu$.

Proof. The statement is proved by induction on the structure of the formula.

$\varphi =$ a for some $a \in \Sigma$. Then it holds that $(w, \rho) \models a$, if and only if the word w is simply the letter a, which finally is the case if and only if $(W, \rho') \models \exists x, y\ (\mathrm{first}(x) \wedge \mathrm{last}(y) \wedge \mathrm{Connected} \wedge E_a(x, y))$.

$\varphi =$ Z for some $Z \in \mathbf{Var}$. Then $(w, \rho) \models Z$ if and only if $w \in \rho(Z)$, which holds if and only if $W \in \rho'(R_Z)$. Thus $(w, \rho) \models Z$ if and only if $W \models \mathrm{Connected} \wedge R_Z$.

$\varphi =$ $\varphi_1 \wedge \varphi_2$. Then $(w, \rho) \models \varphi_1 \wedge \varphi_2$ if and only if $(w, \rho) \models \varphi_1$ and $(w, \rho) \models \varphi_2$. By the inductive hypothesis, this is the case respectively if and only if $(W, \rho') \models [\varphi_1]_\mu$ and $(W, \rho') \models [\varphi_2]_\mu$. Finally this holds if and only if $(W, \rho') \models [\varphi_1]_\mu \wedge [\varphi_2]_\mu \wedge \mathrm{Connected}$.

$\varphi =$ $\varphi_1 \vee \varphi_2$. The case is similar to the one above.

$\varphi =$ $\varphi_1; \varphi_2$. Consider the *only if* direction first. We want to prove that $(w, \rho) \models \varphi \Rightarrow (W, \rho') \models [\varphi]_\mu$. Assume $(w, \rho) \models \varphi_1; \varphi_2$. Then $w \in \{v \mid v = w_1 w_2 \text{ s.t. } (w_j, \rho) \models \varphi_j, j = 1, 2\}$. By the inductive hypothesis $(w_1, \rho) \models \varphi_1$ if and only if $(W_1, \rho') \models [\varphi_1]_\mu$. Similarly for w_2 and W_2. Therefore $(W, \rho') \models \exists x\ \mathrm{first}(x) \wedge \big(([\varphi_1]_\mu \wedge \mathrm{first}(x)) \mid [\varphi_2]_\mu\big) \wedge \mathrm{Connected}$ as required.

For the *if* direction, assume that $(W, \rho') \models \exists x\ \mathrm{first}(x) \wedge \big(([\varphi_1]_\mu \wedge \mathrm{first}(x)) \mid [\varphi_2]_\mu\big) \wedge \mathrm{Connected}$. Then it is the case that W can be split into two connected subwords W_1, W_2 such that W_1 is the one appearing first in the original W and $(W_1, \rho') \models [\varphi_1]_\mu$ and $(W_2, \rho') \models [\varphi_2]_\mu$. The required result then follows from the inductive hypothesis.

$\varphi =$ $\mu Z.\psi$. By the LFLC and GL_μ semantics this case is equivalent to showing the following.

$$w \in \bigcap\{V \subseteq \Sigma^* \mid [\![\psi]\!]_{\rho[Z \mapsto V]} \subseteq V\} \Leftrightarrow$$
$$W \in \bigcap\{H \subseteq \mathcal{G} \mid [\![\ [\psi]_\mu \wedge \mathrm{Connected}\ ]\!]^\mu_{\rho'[R_Z \mapsto H]} \subseteq H\}.$$

Since all subwords are also subgraphs, $\{V \subseteq \Sigma^* \mid [\![\psi]\!]_{\rho[Z \mapsto V]} \subseteq V\} \subseteq \{H \subseteq \mathcal{G} \mid [\![[\psi]_\mu \wedge \mathrm{Connected}]\!]^\mu_{\rho'[R_Z \mapsto H]} \subseteq H\}$. Hence the direction that needs some attention is the one from left to right in the equivalence above.

Suppose there exists a word $w \in \bigcap\{V \subseteq \Sigma^* \mid [\![\psi]\!]_{\rho[Z \mapsto V]} \subseteq V\}$ and suppose for contradiction that there is an H such that $[\![\ [\psi]_\mu \wedge \mathrm{Connected}\ ]\!]^\mu_{\rho'[R_Z \mapsto H]} \subseteq H$ and $W \notin H$.

Let V_H be the subset of H comprising the connected components of H. By the inductive hypothesis, for any words w', W' it holds that $w' \in [\![\psi]\!]_{\rho[Z\mapsto V_H]}$ if and only if $W' \in [\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto V_H]}$. Everything in the latter is a connected component, and since $[\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto V_H]} \subseteq H$ we have that it is also a subset of V_H. Therefore $[\![\psi]\!]_{\rho[Z\mapsto V_H]} \subseteq V_H$ and V_H is an element of $\{V \subseteq \Sigma^* \mid [\![\psi]\!]_{\rho[Z\mapsto V]} \subseteq V\}$. If $W \notin H$ and $V_H \subseteq H$, then $w \notin V_H$ and therefore also $w \notin \bigcap\{V \subseteq \Sigma^* \mid [\![\psi]\!]_{\rho[Z\mapsto V]} \subseteq V\}$, which is a contradiction.

$\varphi = \nu Z.\psi$. By the LFLC and GL_μ semantics this case is equivalent to showing the following.

$$w \in \bigcup\{V \subseteq \Sigma^* \mid V \subseteq [\![\psi]\!]_{\rho[Z\mapsto V]}\} \Leftrightarrow$$
$$W \in \bigcup\{H \subseteq \mathcal{G} \mid H \subseteq [\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto H]}\}.$$

Since all subwords are also subgraphs, $\{V \subseteq \Sigma^* \mid V \subseteq [\![\psi]\!]_{\rho[Z\mapsto V]}\} \subseteq \{H \subseteq \mathcal{G} \mid H \subseteq [\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto H]}\}$. Hence the direction that needs some attention is the one from right to left in the equivalence above. We want to show that for any H in the second set there exists a V in the first one such that the subset of H comprising exactly the connected components, is equal to V.

Fix such an H, and let V_H be the subset of H comprising the connected words of H. By the inductive hypothesis, for any words w', W' it holds that $w' \in [\![\psi]\!]_{\rho[Z\mapsto V_H]}$ if and only if $W' \in [\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto V_H]}$. Notice that $H = V_H$ since by definition $H \subseteq [\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto H]}$ and therefore every graph in H is connected. Hence also $V_H \subseteq [\![[\psi]_\mu \wedge \text{Connected}]\!]^\mu_{\rho'[R_Z\mapsto V_H]}$ and therefore $V_H \subseteq [\![\psi]\!]_{\rho[Z\mapsto V_H]}$ and V_H is an element of $\{V \subseteq \Sigma^* \mid V \subseteq [\![\psi]\!]_{\rho[Z\mapsto V]}\}$.

□

Corollary 4.2.12. *For any language L generated by some conjunctive grammar G, there exists a GL_μ formula φ such that the word language $L_\mu = \{w \mid w \models \varphi\}$ is equal to L.*

Proof. It follows from Theorem 4.2.11 and the equivalence between LFLC and conjunctive grammars. □

4.3 On Trees

An important thing to notice is that GL_μ can be easily seen to be different from MSO in expressive power over trees. This follows from the result on words above, but there are also examples of properties on trees, that are not simply words, that establish the claim. Consider the class of trees that contains trees of height 1 with the root having arbitrarily many children. Over the class of these trees, as shown in [Lib04], MSO cannot express that a tree has a root with an even number of children, but GL_μ can do so using the following formula given in [DGG07].

$$\text{EvenStar} \quad = \quad \exists x \left(\text{root}(x) \wedge \mu R.\ \Big(\epsilon \vee \big(\exists y, z\ (E(x,y) \mid E(x,z) \mid R)\big)\Big)\right).$$

An immediate question is whether the properties definable in MSO over trees form a subset of the ones defined in GL_μ. We give a positive answer to this, and we do so in two different ways, one through STL, a tree logic defined below, and one through a proof in similar lines to the one presented in [GK04] for monadic Datalog. A short introduction into STL will be presented here, while the reader can refer to [BTT05] for further details. The result regarding GL_μ and STL is also given in [DGG07].

Spatial Tree Logic (STL) is the quantifier-free fragment of TQL, which is presented in [CG01] and [CG04]. The latter is a spatial query language on trees which makes use of spatial primitives, a least fixed point operator and quantification over labels and trees. The spatial primitives of STL are of two types. The first allows one to define a tree whose root has a single child, which in turn is the root of some tree satisfying an STL formula. The second primitive allows one to fuse the roots of two trees into one.

Let Λ be the alphabet of edge labels of the trees. Let ξ denote a recursion variable and let α be a subset of Λ. The syntax and semantics of STL formulae are the following.

$$\phi \quad := \quad \mathbf{0} \mid \alpha[\phi_1] \mid \phi_1 | \phi_2 \mid \top \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \xi \mid \mu\xi.\phi_1.$$

where for any formula $\mu\xi.\phi_1$, ξ appears positive in ϕ_1, which ensures monotonicity.

Multisets are used for the representation of unordered and unranked trees. A single node is denoted using the empty multiset $\{\!\{\}\!\}$, and multiset union is denoted with $\uplus$. The set of all trees **Tree** is the smallest set containing the empty multiset and is such that if $t, t' \in \mathbf{Tree}$, then $t \uplus t' \in \mathbf{Tree}$ and $\alpha[t] \in \mathbf{Tree}$. A multiset $\{\!\{t_1, \ldots, t_n\}\!\}$, where $t_1, \ldots, t_n \in \mathbf{Tree}$, can be seen as a tree with n children, where each child is the root of a subtree isomorphic to one of $t_1, \ldots, t_n$.

Let δ be a valuation function associating a set of trees to each recursion variable. The interpretation of an STL formula φ under δ is denoted by $[\![\varphi]\!]_\delta$. It is recursively defined as follows.

$$\begin{aligned}
[\![\mathbf{0}]\!]_\delta &= \{\{\!\{\}\!\}\}, \\
[\![\alpha[\varphi]]\!]_\delta &= \{\{\!\{a[t]\}\!\} \mid a \in \alpha, t \in [\![\varphi]\!]_\delta\}, \\
[\![\varphi_1 \mid \varphi_2]\!]_\delta &= \{t_1 \uplus t_2 \mid t_1 \in [\![\varphi_1]\!]_\delta, t_2 \in [\![\varphi_2]\!]_\delta\}, \\
[\![\top]\!]_\delta &= \mathbf{Tree}, \\
[\![\neg\varphi]\!]_\delta &= \mathbf{Tree} \setminus [\![\varphi]\!]_\delta, \\
[\![\varphi_1 \vee \varphi_2]\!]_\delta &= [\![\varphi_1]\!]_\delta \cup [\![\varphi_2]\!]_\delta, \\
[\![\xi]\!]_\delta &= \delta(\xi), \\
[\![\mu\xi.\varphi]\!]_\delta &= \bigcap\{S \mid [\![\varphi]\!]_{\delta[\xi \mapsto S]} \subseteq S\}.
\end{aligned}$$

The authors of [BTT05] compare the expressive power of MSO with STL and prove that a syntactic fragment of STL is equi-expressive to MSO over trees. In particular let a STL formula ψ be a guarded STL formula if the occurrence of any recursion variable appears under the operator $\alpha[.]$. Let the set of STL formulae that are guarded be denoted by gSTL.

Theorem 4.3.1 (Boneva et al., [BTT05])**.** *For any set of trees $\mathcal{T}$, $\mathcal{T}$ is* MSO *definable if and only if it is* gSTL *definable.*

An example that separates STL from MSO, is the same as the one given in the beginning of Section 4.3 for GL_μ. In particular the same class of trees of height 1 with an even number of edges is definable in STL by $\mu\xi.(\alpha[\mathbf{0}] \mid \alpha[\mathbf{0}] \mid \xi \vee \mathbf{0})$.

Theorem 4.3.2 (Boneva et al., [BTT05])**.** STL *is strictly more expressive than* MSO*.*

Using the above two theorems, and a translation of STL formulae into GL_μ ones, we get the required result, namely that over unranked, unordered trees, GL_μ is strictly more expressive than MSO. Let $\|.\|_G$ denote the inductive translation of STL formulae into GL_μ ones.

$$
\begin{array}{lcl}
\|\mathbf{0}\|_G & = & \mathbf{0}, \\
\|\alpha[\psi]\|_G & = & \exists x, y\Big(\mathrm{root}(x) \wedge \bigvee_{a\in\alpha}\big(E_a(x,y) \mid (\mathrm{Connected} \wedge \mathrm{root}(y) \wedge \|\psi\|_G)\big)\Big), \\
\|\psi_1 \mid \psi_2\|_G & = & \exists x\ \Big(\mathrm{root}(x) \\
 & & \wedge\big(\mathrm{root}(x) \wedge \|\psi_1\|_G \wedge \mathrm{Connected} \mid \mathrm{root}(x) \wedge \|\psi_2\|_G \wedge \mathrm{Connected}\big)\Big), \\
\|\top\|_G & = & \top, \\
\|\neg\psi\|_G & = & \neg\|\psi\|_G, \\
\|\psi_1 \vee \psi_2\|_G & = & \|\psi_1\|_G \vee \|\psi_2\|_G, \\
\|\xi\|_G & = & R_\xi, \\
\|\mu\xi.\psi\|_G & = & \mu R_\xi.\|\psi\|_G.
\end{array}
$$

For what follows, if δ is a valuation function associating a set of trees to each recursion variable for STL, we define ρ to be the valuation function in GL_μ, that corresponds to δ and maps each recursion variable to a set of trees such that for any tree T and any recursion variable ξ, $T \in \delta(\xi)$ if and only if $T \in \rho(R_\xi)$. The translation above leads to the following theorem.

Theorem 4.3.3. *For any* STL *formula φ and any valuation function δ in* STL*, the set of trees defined by φ according to δ is equal to the set of trees defined by $\|\varphi\|_G$ according to ρ, the valuation function in* GL_μ *corresponding to δ.*

Proof. The proof is by induction on the structure of the formula. The notation $[\![.]\!]^G_{\sigma;\rho}$ is used to denote the satisfaction interpretation of GL_μ formulae, with the image of this interpretation restricted to trees instead of graphs in general.

$\varphi = \mathbf{0}$. The statement is true for this case. Similarly for the case $\varphi = \top$.

$\varphi = \xi$. Then $[\![\xi]\!]_\delta = \delta(\xi)$, where the latter over trees is equal to $\rho(R_\xi)$, and therefore, for any tree T, $T \in [\![\xi]\!]_\delta$ if and only if $T \in [\![R_\xi]\!]^G_{\sigma;\rho}$.

$\varphi = \psi_1 \vee \psi_2$. Then $[\![\psi_1 \vee \psi_2]\!]_\delta = [\![\psi_1]\!]_\delta \cup [\![\psi_2]\!]_\delta$ and by the inductive hypothesis we have that $[\![\psi_1]\!]_\delta = [\![\|\psi_1\|_G]\!]^G_{\sigma;\rho}$ and $[\![\psi_2]\!]_\delta = [\![\|\psi_2\|_G]\!]^G_{\sigma;\rho}$. Also $[\![\|\psi_1 \vee \psi_2\|_G]\!]^G_{\sigma;\rho} = [\![\|\psi_1\|_G \vee \|\psi_2\|_G]\!]^G_{\sigma;\rho} = [\![\|\psi_1\|_G]\!]^G_{\sigma;\rho} \cup [\![\|\psi_2\|_G]\!]^G_{\sigma;\rho}$ and therefore $[\![\psi_1 \vee \psi_2]\!]_\delta = [\![\|\psi_1 \vee \psi_2\|_G]\!]^G_{\sigma;\rho}$

$\varphi = \neg\psi$. Then $[\![\neg\psi]\!]_\delta = \mathbf{Tree} \setminus [\![\psi]\!]_\delta$. By inductive hypothesis it holds that $[\![\psi]\!]_\delta = [\![\|\psi\|_G]\!]^G_{\sigma;\rho}$ and hence $[\![\|\neg\psi\|_G]\!]^G_{\sigma;\rho} = [\![\neg\psi]\!]_\delta$.

$\varphi = \alpha[\psi]$. Then $[\![\alpha[\psi]]\!]_\delta = \{\{\!|\, a[t] \,|\!\} \mid a \in \alpha, t \in [\![\psi]\!]_\delta\}$. By the inductive hypothesis it is the case that $[\![\psi]\!]_\delta = [\![\|\psi\|_G]\!]^G_{\sigma;\rho}$. Therefore the formula Connected $\wedge$ root$(y) \wedge \|\psi\|_G$ is satisfied exactly by the trees $t \in [\![\psi]\!]_\delta$, having root y. By the translation of STL formulae in GL_μ, we have that $\|\alpha[\psi]\|_G$ defines the set of trees that have some root x, an edge from x to some node y with a label from the set α, and y is the root of a tree $t \in [\![\psi]\!]_\delta$. Therefore $[\![\alpha[\psi]]\!]_\delta = [\![\|\alpha[\psi]\|_G]\!]^G_{\sigma;\rho}$.

$\varphi = \psi_1 \mid \psi_2$. In this case, $[\![\psi_1 \mid \psi_2]\!]_\delta = \{t_1 \uplus t_2 \mid t_1 \in [\![\psi_1]\!]_\delta, t_2 \in [\![\psi_2]\!]_\delta\}$ and by the inductive hypothesis it holds for $i = 1, 2$ that, $[\![\psi_i]\!]_\delta = [\![\|\psi_i\|_G]\!]^G_{\sigma;\rho}$. According to the definition of the multiset representation for trees, the union of two multisets represents the fusion of the roots of the trees represented by each of the two multisets. This is exactly what the translation $\|\psi_1 \mid \psi_2\|_G$ expresses in GL_μ, and hence $[\![\psi_1 \mid \psi_2]\!]_\delta = [\![\|\psi_1 \mid \psi_2\|_G]\!]^G_{\sigma;\rho}$.

$\varphi = \mu\xi.\psi$ Then $[\![\mu\xi.\psi]\!]_\delta = \bigcap\{S \mid [\![\psi]\!]_{\delta[\xi\mapsto S]} \subseteq S\}$. By the inductive hypothesis, it is true that for any set of trees S, $[\![\psi]\!]_{\delta[\xi\mapsto S]} = [\![\|\psi\|_G]\!]^G_{\sigma;\rho[R_\xi\mapsto S]}$. Therefore $\bigcap\{S \mid [\![\psi]\!]_{\delta[\xi\mapsto S]} \subseteq S\} = \bigcap\{S \mid [\![\|\psi\|_G]\!]^G_{\sigma;\rho[R_\xi\mapsto S]} \subseteq S\}$.

□

By Theorem 4.3.2, STL is strictly more expressive than MSO over trees which leads to the following Corollary.

Corollary 4.3.4. GL_μ *is strictly more expressive than* MSO *on trees.*

We proceed by establishing this same result for ranked trees using a proof of similar lines to the one used in [GK04] for Monadic Datalog or, more generally, the one used for MSO in the literature, namely MSO types, or similarly, states of automata. The proof presented below is closer to the one for Monadic Datalog.

Theorem 4.3.5 (Gottlob et al., [GK04])**.** *A tree language is regular if and only if it is definable in Monadic Datalog.*

More precisely the above authors prove that for every unary MSO definable query there exists a Monadic Datalog query such that the two queries return the same result on all ranked trees. The proof for the latter uses rules that recursively define MSO k-types of trees by working both bottom-up and top-down, which aids into selecting nodes in the tree towards the definition of any unary query. In other words, suppose that x is a vertex of some tree T, and let T_x be the subtree of T that is rooted at x. Then the MSO rank-k type $\mathrm{tp}_k(T, x)$, of the node x in T depends on the MSO rank-k type $\mathrm{tp}_k(T_x)$ of the tree T_x and the type $\mathrm{tp}_k(T \setminus T_x)$ of the tree $T \setminus T_x$ (see [Mak04]). The types of these trees are calculated recursively working bottom-up in the first case and top-down in the second one, in the proof of the corresponding Theorem in

[GK04]. For the proof of a result such as Theorem 4.3.5, the rank-k 0-types of trees are required only, and therefore one needs to recursively define the type of a tree by working bottom-up only. We proceed by establishing a similar result for GL_μ.

In order for this to work for GL_μ, a useful result should be given first, regarding simultaneous fixed points. Let the notation $\varphi(R)$ denote a GL_μ formula φ with the recursion variable R appearing in it free and when R is a recursion variable appearing in a formula φ and $\mathcal{F}$ is a set of graphs, $[\![\varphi]\!]_{\sigma;\rho[R\mapsto\mathcal{F}]}$ will sometimes be denoted with $[\![\varphi(\mathcal{F})]\!]_{\sigma;\rho}$. Let $\varphi_1(R_1,\ldots,R_m),\ldots,\varphi_m(R_1,\ldots,R_m)$ be a sequence of formulae of vocabulary $\tau\cup\{R_1,\ldots,R_m\}$. The next two lemmas are almost identical to Lemma 10.9 and Theorem 10.8 in [Lib04] and Lemma 3.3.41 and Theorem 3.3.42 in [GKL$^+$07]. A more general introduction can be found in [AN01].

$$S := \begin{cases} R_1 & \leftarrow & \varphi_1(R_1,\ldots,R_m) \\ & \vdots & \\ R_m & \leftarrow & \varphi_m(R_1,\ldots,R_m) \end{cases} .$$

Let the above be a system of update rules, that define a tuple of operators $S=(S_1,\ldots,S_m)$ such that for each tuple of sets of graphs $(\mathcal{G}_1,\ldots,\mathcal{G}_m)$ it is the case that $S_i(\mathcal{G}_1,\ldots,\mathcal{G}_m)=\mathcal{G}_i'$, where $\mathcal{G}_i'=[\![\varphi_i(\mathcal{G}_1,\ldots,\mathcal{G}_m)]\!]_{\sigma;\rho}$, for all $1\leq i\leq m$. These operators can be combined into a single one,

$$S:\wp(\mathcal{G})\times\ldots\times\wp(\mathcal{G})\to\wp(\mathcal{G})\times\ldots\times\wp(\mathcal{G}),$$

which is monotone and has a least fixed point. For each of the R_i in the above system of update rules S, we define $[\![\mu R_i:S]\!]_{\sigma;\rho}$ to be the i-th component of the least fixed point of S.

Let us denote by S-GL_μ the logic that is obtained by adding this form of simultaneous recursion to GL_μ. We will prove that adding simultaneous fixed points in GL_μ does not increase its expressive power. In particular, it is shown that every S-GL_μ formula $[\![\mu R_i:S]\!]_{\sigma;\rho}$, where S is a system of simultaneous fixed points, is equivalent to a GL_μ formula, by simulating S using nested fixed points as explained in the proof of Lemma 4.3.7 below.

For simplicity we will consider a system of just two update rules.

$$S := \begin{cases} R & \leftarrow & \varphi_R(R,T) \\ T & \leftarrow & \varphi_T(R,T) \end{cases} .$$

The above two update rules define two monotone operators $\mathcal{R}:\wp(\mathcal{G})\times\wp(\mathcal{G})\to\wp(\mathcal{G})$ and $\mathcal{T}:\wp(\mathcal{G})\times\wp(\mathcal{G})\to\wp(\mathcal{G})$, where $\mathcal{G}$ is the set of all graphs. The system of the two operators define a combined, monotone one, $S=(\mathcal{R},\mathcal{T}):\wp(\mathcal{G})\times\wp(\mathcal{G})\to\wp(\mathcal{G})\times\wp(\mathcal{G})$, whose fixed point is denoted by S^∞ or $(\mathcal{R}^\infty,\mathcal{T}^\infty)$. For any fixed set of graphs $X\in\wp(\mathcal{G})$, we define the operator $\mathcal{T}_X:\wp(\mathcal{G})\to\wp(\mathcal{G})$ with $\mathcal{T}_X(Y)=\mathcal{T}(X,Y)$. The operator $\mathcal{T}_X$ is monotone for each fixed X, and hence has a least fixed point, denoted by $\mathcal{T}_X^\infty$.

Lemma 4.3.6 ([Lib04],[GKL$^+$07])**.** *The operator $\mathcal{V}:\wp(\mathcal{G})\to\wp(\mathcal{G})$ defined by $\mathcal{V}(X)=\mathcal{R}(X,\mathcal{T}_X^\infty)$ is monotone whose least fixed point ($\mathcal{V}^\infty$) is equal to $\mathcal{R}^\infty$.*

Proof. Suppose $X \subseteq X'$. Then for all stages α of the operators $\mathcal{T}_X$ and $\mathcal{T}_{X'}$, it is the case that $\mathcal{T}_X^\alpha \subseteq \mathcal{T}_{X'}^\alpha$. Hence $\mathcal{T}_X^\infty \subseteq \mathcal{T}_{X'}^\infty$. Therefore $\mathcal{V}(X) = \mathcal{R}(X, \mathcal{T}_X^\infty) \subseteq \mathcal{R}(X', \mathcal{T}_{X'}^\infty) = \mathcal{V}(X')$ and $\mathcal{V}$ is monotone.

For the second part of the statement, and in particular the direction of $\mathcal{V}^\infty \subseteq \mathcal{R}^\infty$ consider the following. First, notice that $\mathcal{T}^\infty$ is a fixed point of $\mathcal{T}_{\mathcal{R}^\infty}$, since $\mathcal{T}_{\mathcal{R}^\infty}(\mathcal{T}^\infty) = \mathcal{T}(\mathcal{R}^\infty, \mathcal{T}^\infty) = \mathcal{T}^\infty$. Hence $\mathcal{T}^\infty$ contains the least fixed point of $\mathcal{T}_{\mathcal{R}^\infty}$ and therefore $\mathcal{T}_{\mathcal{R}^\infty}^\infty \subseteq \mathcal{T}^\infty$.

Notice also that $\mathcal{V}(\mathcal{R}^\infty) = \mathcal{R}(\mathcal{R}^\infty, \mathcal{T}_{\mathcal{R}^\infty}^\infty)$. By the above this is a subset of or equal to $\mathcal{R}(\mathcal{R}^\infty, \mathcal{T}^\infty) = \mathcal{R}^\infty$. As the least fixed point of $\mathcal{V}$ is $\bigcap\{X |\ \mathcal{V}(X) \subseteq X\}$, we have that $\mathcal{V}^\infty \subseteq \mathcal{R}^\infty$.

For the direction $\mathcal{R}^\infty \subseteq \mathcal{V}^\infty$ we proceed by induction on the stage α of the operator to prove that $\mathcal{R}^\alpha \subseteq \mathcal{V}^\infty$ and $\mathcal{T}^\alpha \subseteq \mathcal{T}_{\mathcal{V}^\infty}^\infty$. For stage 0 it is clear since $\mathcal{R}^0 = \emptyset$ and $\mathcal{T}^0 = \emptyset$.

Assume that the case is so for stage α. Then $\mathcal{R}^{\alpha+1} = \mathcal{R}(\mathcal{R}^\alpha, \mathcal{T}^\alpha)$ and by the induction hypothesis the latter is a subset of or equal to $\mathcal{R}(\mathcal{V}^\infty, \mathcal{T}_{\mathcal{V}^\infty}^\infty) = \mathcal{V}(\mathcal{V}^\infty) \subseteq \mathcal{V}^\infty$.

Similarly we have that $\mathcal{T}^{\alpha+1} = \mathcal{T}(\mathcal{R}^\alpha, \mathcal{T}^\alpha)$, where again the latter is, by the inductive hypothesis, a subset of or equal to $\mathcal{T}(\mathcal{V}^\infty, \mathcal{T}_{\mathcal{V}^\infty}^\infty) = \mathcal{T}_{\mathcal{V}^\infty}(\mathcal{T}_{\mathcal{V}^\infty}^\infty) = \mathcal{T}_{\mathcal{V}^\infty}^\infty$.

For a limit ordinal λ, it is clear. □

Lemma 4.3.7 ([Lib04],[GKL$^+$07])**.** S-GL$_\mu \equiv$ GL$_\mu$.

Proof. Only one direction of equivalence needs to be shown, and in particular that any formula of S-GL$_\mu$ representing a system of simultaneous fixed point operators can be expressed by a GL$_\mu$ formula. This is done by simulating a simultaneous fixed point by a simple nested one, using a technique known as the Bekic principle ([AN01]). For simplicity we consider the following system of only two update rules.

$$S := \begin{cases} R & \leftarrow & \varphi_1(R,T) \\ T & \leftarrow & \varphi_2(R,T). \end{cases}$$

We claim that the following two equivalences hold:

$$\begin{aligned} \mu R : S &\equiv \mu R.\varphi_1\big(R, (\mu T.\varphi_2)/T\big), \\ \mu T : S &\equiv \mu T.\varphi_2\big((\mu R.\varphi_1)/R, T\big). \end{aligned}$$

In the above, the notation $(\mu T.\varphi_2)/T$ is used to denote the substitution of T with the formula $\mu T.\varphi_2$. The proof of the first equivalence is shown. We define the two monotone operators $\mathcal{R}$ and $\mathcal{T}$ where $\mathcal{R}(X,Y) = [\![\varphi_1]\!]_{\sigma;\rho[R\mapsto X, T\mapsto Y]}$ and similarly $\mathcal{T}(X,Y) = [\![\varphi_2]\!]_{\sigma;\rho[R\mapsto X, T\mapsto Y]}$. We denote their least fixed points with $\mathcal{R}^\infty$ and $\mathcal{T}^\infty$ respectively, as earlier. According to this, a graph G is a model of $\mu R : S$ if and only if $G \in \mathcal{R}^\infty$.

Similarly we define $\mathcal{V} : X \mapsto [\![\varphi_1]\!]_{\sigma;\rho[R\mapsto X, T\mapsto Z]}$ where $Z = [\![\mu T.\varphi_2]\!]_{\sigma;\rho[R\mapsto X]}$. A graph G belongs to the least fixed point of $\mathcal{V}$ if and only if $G \models \mu R.\varphi_1\big(R, (\mu T.\varphi_2)/T\big)$. By Lemma 4.3.6 we have that $\mathcal{V}^\infty = \mathcal{R}^\infty$ and it is hence the case that $G \models \mu R.\varphi_1\big(R, (\mu T.\varphi_2)/T\big) \Leftrightarrow G \models \mu R : S$. □

The above two Lemmas, by giving a way of transforming a set of simultaneous recursion formulae into a single nested one, provide what is needed for the following result.

Theorem 4.3.8. *Every regular tree language of ranked trees is* GL_μ *definable.*

Proof. Let L be some regular tree language of trees of rank M and alphabet Σ. Then this language is definable by some MSO formula of rank k, for some $k \in \mathbb{N}$. Let this MSO formula be φ_L, and let $T_L = \{\tau_0, \ldots, \tau_N\}$ be an enumeration of all MSO k-types. The MSO k-type of a tree t having a root with m children, is determined by the k-types of the trees rooted at these m children together with the labels along the edges connecting them to the root of the tree t.

Let S be the set of multisets of at most M elements, where each element is of the form (τ, σ) where $\tau \in T_L$ and $\sigma \in \Sigma$. Let $f : S \to T_L$ be the function that returns the type of a tree according to the types of the trees beneath its root and the labels on the edges connecting the root to these trees. Let $<_T$ be a linear order on the elements of T_L and let $\tau_j \in T_L$ for some $1 \leq j \leq N$. Suppose $f^{-1}(\tau_j) = \{e_1, \ldots, e_\ell\}$. For each $r \leq \ell$, we define a GL_μ formula of the following form.

$$\begin{aligned}\theta_j^r \quad = \quad & \exists x\ \mathrm{root}(x) \wedge \exists x_1, \ldots, x_i \\ & \Big(\big(a_{n_1}(x, x_1) \mid (R_{n_1} \wedge \mathrm{Connected} \wedge \mathrm{root}(x_1))\big) \mid \ldots \\ & \ldots \mid \big(a_{n_i}(x, x_i) \mid (R_{n_i} \wedge \mathrm{Connected} \wedge \mathrm{root}(x_i))\big)\Big).\end{aligned}$$

In the above formula, i is the size of the multiset e_r and (τ_{n_h}, a_{n_h}) for $1 \leq h \leq i$, are elements of this multiset e_r to which the formula corresponds. We then define a formula for each type τ_j as shown below.

$$\theta_j \quad = \quad \theta_j^1 \vee \ldots \vee \theta_j^\ell.$$

Let S be the following system of update rules according to the formulae θ_j.

$$S := \begin{cases} R_1 & \leftarrow \quad \theta_1 \\ & \vdots \\ R_N & \leftarrow \quad \theta_N \end{cases}.$$

Any MSO formula φ of rank k is equivalent to the disjunction of some of the types. We claim that any MSO formula φ is equivalent to a S-GL_μ formula of the following form.

$$(\mu R_{n_1} : S) \vee (\mu R_{n_2} : S) \vee \ldots \vee (\mu R_{n_p} : S).$$

But due to Lemma 4.3.7 any formula of the form $\mu R_j : S$ for some $1 \leq j \leq N$, is equivalent to a nested GL_μ formula. It remains to be proved that for any tree t, t is of type τ_j if and only if $t \models \mu R_j : S$. We proceed by induction on the height of the tree.

The base case is clear for the type of a single node. Assume the statement holds for all trees of height less than h, for some $h \in \mathbb{N}$, and let t be of height h and of type τ_j. The types of the

trees rooted at the children nodes of the root of t together with the edges connecting them to it, will be given by some $e_r \in f^{-1}(\tau_j)$. For each tuple $(\tau_{j_\ell}, \sigma_{j_\ell})$ in e_r, we have by the induction hypothesis that the subtree t_{j_ℓ} rooted at the appropriate child of the root of t satisfies $\mu R_{j_\ell} : S$. The tree t satisfies θ_j^r, and therefore, according to the update rules S, t is in the least fixed point of R_j of S. Hence $t \models \mu R_j : S$.

For the other direction assume that $t \models \mu R_j : S$ for some $j \in [N]$. In order for t to be in the least fixed point of R_j, it must satisfy θ_j under some assignment ρ of sets of trees to recursion variables, and hence t must satisfy at least one of the formulae θ_j^i in the disjunction. Let $(t, \rho) \models \theta_j^\ell$. The latter formula describes the subtrees rooted at the children of the root of t, together with the edges that connect them to it, and therefore, by the inductive hypothesis and the definition of θ_j^ℓ, the tree t is of type τ_j. □

Notice that Theorem 4.3.8 and its proof can be adapted for unary queries in a similar way as it is done for Monadic Datalog in [GK04]. It should also be noted, that by Lemma 4.3.7, where it is shown that S-GL$_\mu \equiv$ GL$_\mu$, it is possible to obtain a more straightforward and direct translation from conjunctive grammars to GL$_\mu$ formulae.

Chapter 5

Conclusion

5.1 GL

We investigated the expressive power of GL over finite graphs. In Chapter 2 we focused on properties that are expressible in this logic over words, over trees and over graphs in general. We showed that the result of [DGG07] which states that over words GL and MSO are equi-expressive, extends to structures that are formed by taking the disjoint union of word structures. Over trees, a natural candidate property for non-expressibility was investigated, namely whether the class of binary trees with an even number of leaf vertices is definable in GL. It was shown that such a property is expressible in GL, although it is speculated that this does not necessarily extend to a more general result, such as whether the class of binary trees with 0 (mod k) number of leaves, for some arbitrary $k \in \mathbb{N}$, is definable in GL. More specifically, it was shown that any language of binary trees accepted by a product of deterministic bottom-up binary tree automata with 2 states is definable in GL. Consequently, this excludes some natural candidates for separating GL from MSO over trees.

A comparison of GL with other logics and machine models over trees, that are properly included in MSO, was made in an attempt to separate the logic from MSO, but it was shown that GL is not included and possibly incomparable to the other logics considered. In other words, GL is able to express properties that the other logics cannot, and possibly vice versa. In particular, we established that there are GL-definable properties that are not definable by tree-walking automata and others that are not definable in Antichain Logic or Chain Logic.

Over graphs in general, it was shown that 4-colourability, and more generally 2^k-colourability, for any $k \in \mathbb{N}$, is definable in GL. This result does not seem to be easily extended to 3-colourability, and the question of whether the latter is definable in GL remains open.

In Chapter 3, we showed that GL is strictly less expressive than MSO over graphs, and this is the case even when restricted over the class of unlabelled directed forests. This was done by considering forests comprising binary trees and additional disjoint strings that distracted Spoiler in the GL game between two such forests. The property shown to be inexpressible in GL over such structures is whether the binary tree inside such forests has 1 (mod 3) number of leaves.

When the graph is decomposed into subgraphs during the various rounds of the game, Spoiler finds it difficult to distinguish between leaves of the additional disjoint strings in the forest, and leaves of the binary tree. With appropriately constructed structures, Duplicator wins the game.

The importance of the additional disjoint strings in the structures on which the GL game is played, is shown using first-order interpretations and monadic second-order transductions. Due to the above separating result, it was established that GL is closed under neither of them. The forests used in the separating result, were interpreted into single trees comprising both the binary tree and the strings acting as noise, all connected to the root of the tree. Furthermore, this was done using first-order formulae, and it was shown that over such structures, the property about the leaves of the binary tree mentioned above, is GL definable. Thus, the importance of the disjoint strings in the structures constructed for separating GL from MSO, is clearly illustrated.

As a corollary of GL being strictly less expressive than MSO over forests, we obtained a result regarding Separation Logic, and more specifically one of its fragments, namely SL($*$), that is considered by Brochenin et al. in [BDL08] and is the fragment lacking the magic wand of Separation Logic. We give a positive answer to a conjecture of the authors, namely that SL($*$) is strictly less expressive than MSO over memory heaps, structures similar to graphs of unary functions.

5.2 GL_μ

In a similar manner, we examined the expressive power of GL_μ, the extension of GL with a recursion operator. It was shown in [DGG07] that PSPACE-complete problems are expressible in GL_μ over graphs, and we similarly showed that the same holds even when restricted over the class of words, which was done by encoding QBF instances into labelled words. Consequently, we compared GL_μ to grammars that derive word languages that strictly contain the regular ones, and showed in particular that GL_μ is at least as expressive as conjunctive grammars introduced in [Okh01], using a different method that the one in [DGG07] where the same result is shown.

It was established though, that languages thought to be not definable by conjunctive grammars, are definable in GL_μ, which is also closed under complementation, unlike conjunctive grammars where this is not known. Conjunctive grammars define languages in PTIME, and GL_μ defines PSPACE-complete languages, and therefore the inclusion in GL_μ of more expressive grammars than conjunctive ones is possible.

Over trees, it was established through Spatial Tree Logic introduced in [BTT05], that GL_μ is strictly more expressive than MSO. In particular an inductively defined translation was given for formulae from STL to GL_μ. Furthermore, it was shown that S-GL_μ, the version of GL_μ with simultaneous fixed-points, is equivalent to GL_μ and a direct proof was given of the containment of MSO in GL_μ. The equivalence between GL_μ and S-GL_μ provides an easier way to translate other logics and grammars to GL_μ, such as conjunctive grammars and Monadic Datalog.

Bibliography

[AD09] Timos Antonopoulos and Anuj Dawar. Separating graph logic from MSO. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2009.

[AK08] Tamar Aizikowitz and Michael Kaminski. Conjunctive grammars and alternating pushdown automata. In Wilfrid Hodges and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 5110 of *Lecture Notes in Computer Science*, pages 44–55. Springer, 2008.

[AN01] A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

[AU71] Alfred V. Aho and Jeffrey D. Ullman. Translations on a context-free grammar. *Information and Control*, 19(5):439–475, 1971.

[BC08] Mikolaj Bojanczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM J. Comput.*, 38(2):658–701, 2008.

[BDL08] Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. In Michael Kaminski and Simone Martini, editors, *Proceedings of the 16th Annual EACSL Conference on Computer Science Logic (CSL'08)*, volume 5213 of *Lecture Notes in Computer Science*, pages 323–338. Springer, September 2008.

[BTT05] Iovka Boneva, Jean-Marc Talbot, and Sophie Tison. Expressiveness of a spatial logic for trees. In *LICS*, pages 280–289. IEEE Computer Society, 2005.

[Büc60] J. Richard Büchi. Weak second order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[CG00] Luca Cardelli and Andrew D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *POPL*, pages 365–377, 2000.

[CG01] Luca Cardelli and Giorgio Ghelli. A query language based on the ambient logic. In David Sands, editor, *ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[CG04] Luca Cardelli and Giorgio Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004.

[CGG02] Luca Cardelli, Philippa Gardner, and Giorgio Ghelli. A spatial logic for querying graphs. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.

[Com89] Kevin J. Compton. A logical approach to asymptotic combinatorics ii: Monadic second-order properties. *J. Comb. Theory, Ser. A*, 50(1):110–131, 1989.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

[Cou94] Bruno Courcelle. The monadic second order logic of graphs VI: on several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2-3):117–149, 1994.

[Cou97] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars*, pages 313–400. World Scientific, 1997.

[Cou03] Bruno Courcelle. The monadic second-order logic of graphs XIV: uniformly sparse graphs and edge set quantifications. *Theor. Comput. Sci.*, 1-3(299):1–36, 2003.

[Cou08] Bruno Courcelle. Graph algebras and monadic second-order logic. 2008.

[DGG04] Anuj Dawar, Philippa Gardner, and Giorgio Ghelli. Adjunct elimination through games in static ambient logic. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004.

[DGG07] Anuj Dawar, Philippa Gardner, and Giorgio Ghelli. Expressiveness and complexity of graph logic. *Inf. Comput.*, 205(3):263–310, 2007.

[Die05] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.

[Don70] John Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[dR87] Michel de Rougemont. Second-order and inductive definability on finite structures. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 33:47–63, 1987.

[EF99] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, second edition edition, 1999.

[Elg61] Calvin Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity and Computation*, 7:43–73, 1974.

[FG04] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

[FV59] Feferman and Vaught. The first-order properties of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

[GHO02] Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *ACM Trans. Comput. Log.*, 3(3):418–463, 2002.

[GK04] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[GKL$^+$07] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, 2007.

[Hal76] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976.

[Hod97] Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104, 1986.

[IO01] Samin S. Ishtiaq and Peter W. O'Hearn. Bi as an assertion language for mutable data structures. In *POPL*, pages 14–26, 2001.

[Lan02] Martin Lange. Alternating context-free languages and linear time mu-calculus with sequential composition. *Electr. Notes Theor. Comput. Sci.*, 68(2), 2002.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2004.

[Mak04] Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.

[Mar06] Jerzy Marcinkowski. On the expressive power of graph logic. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 2006.

[MO99] Markus Müller-Olm. A modal fixpoint logic with chop. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 510–520. Springer, 1999.

[Mor89] Etsuro Moriya. A grammatical characterization of alternating pushdown automata. *Theor. Comput. Sci.*, 67(1):75–85, 1989.

[Okh01] Alexander Okhotin. Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4):519–535, 2001.

[Okh03] Alexander Okhotin. An overview of conjunctive grammars, formal language theory column. *Bulletin of the EATCS*, 79:145–163, 2003.

[Okh04] Alexander Okhotin. Boolean grammars. *Inf. Comput.*, 194(1):19–48, 2004.

[Ott97] Martin Otto. *Bounded Variable Logics and Counting*, volume 9 of *Lecture Notes in Logic*. Springer-Verlag, Berlin, 1997.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PT93] Andreas Potthoff and Wolfgang Thomas. Regular tree languages without unary symbols are star-free. In Zoltán Ésik, editor, *FCT*, volume 710 of *Lecture Notes in Computer Science*, pages 396–405. Springer, 1993.

[Rey02] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

[RS71] R. McNaughton and S. Papert. *Counter-free Automata*. MIT Press, Cambridge, USA, 1971.

[RS86] Robertson and Seymour. Graph minors. II. Algorithmic aspects of tree-width. *ALGORITHMS: Journal of Algorithms*, 7:309–322, 1986.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages (3 volumes)*. Springer, 1997.

[Sip05] M. Sipser. *Introduction to the Theory of Computation.* CourseTechnology, 2005.

[Tho84] Wolfgang Thomas. Logical aspects in the study of tree languages. In *CAAP*, pages 31–50, 1984.

[Tho96] W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1996.

[Tho97] Wolfgang Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In Jan Mycielski, Grzegorz Rozenberg, and Arto Salomaa, editors, *Structures in Logic and Computer Science*, volume 1261 of *Lecture Notes in Computer Science*, pages 118–143. Springer, 1997.

[Tur84] György Turán. On the definability of properties of finite graphs. *Discrete Mathematics*, 49(3):291–302, 1984.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982.