Fink Amores, Michael Christian:

The Graph Isomorphism Problem and the GI-Completeness of Selected Problems from Context-Free Grammars and Rewrite Systems

Bachelorarbeit, Wintersemester 2021

Gutachter*in: Sabel, David ; Merkl, Franz

Fakultät für Mathematik, Informatik und Statistik

Mathematik, Informatik

Ludwig-Maximilians-Universität München

https://doi.org/10.5282/ubm/epub.78037

INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
Lehr- und Forschungseinheit für Theoretische Informatik

MATHEMATISCHES INSTITUT
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Bachelor Thesis

The Graph Isomorphism Problem and the GI-Completeness of Selected Problems from Context-Free Grammars and Rewrite Systems

Michael Christian Fink Amores

Supervisors: Prof. Dr. David Sabel & Prof. Dr. Franz Merkl
Deadline: March 31st, 2021

I hereby assure that I have written the present bachelor thesis independently and that I have not used any sources or resources other than those stated.

München, March 31st, 2021

. . . .

(Michael Christian Fink Amores)

Abstract

Graph isomorphisms are adjacency and label (equivalence class) preserving one-to-one correspondences between vertex sets of possibly labelled graphs. The graph isomorphism problem is then the task of deciding whether two given graphs are isomorphic or not and basis of complexity class **GI**, containing all problems with polynomial reduction to former problem. We highlight history and state of the art research on graph isomorphism related problems, with special focus on categorisation of **GI** in the complexity hierarchy and its relation to the unsolved $\mathbf{P} = \mathbf{NP}$ problem, where strong theoretic and heuristic evidence suggests that **GI** is **NP**-intermediate.
The core of the thesis is study of **GI**-completeness of selected problems from context-free grammars and term rewriting systems. We show polynomial equivalence of the graph isomorphism problem and decision problems on context-free and regular grammars, involving grammar isomorphisms and isomorphic strict interpretations, describing bijective mappings between nonterminals and terminals of different symbol sets.
Grammar concepts are specialised to term rewriting systems, formalising substitution based on a function/variable symbol term model. Isomorphism on such finite structures involve local, on a per rule basis, and global rewriting of respective symbol sets, including arbitrary combinations. Completely local term rewriting system isomorphism can be determined in polynomial time by introduction of local templates, subject to different restrictions. Explicit pseudocode algorithms are subsequently provided to support those claims. In contrast, different graph encodings are used to prove that global rewriting of atleast one symbol set already leads to **GI**-completeness of the related decision problem.

Acknowledgements

I thank my supervisors Dr. David Sabel and Dr. Franz Merkl for their continuous input and support, my sister Yasmina Fink Amores for proof reading and patiently listening to my ideas, and lastly my parents for their motivating words.

Contents

1 Introduction

Graph theory is one of the most prominent fields of research in all of mathematics and (theoretical) computer science with a wide variety of applications. This is owed to the extremely simple concept of graphs, abstracting structural relations, and a big catalogue of open, unanswered questions, inspiring researchers all over the globe. In particular finite structures can be encoded as graphs and allow us to fall back on graph theoretical methods to solve certain problems. One of these problems is the famous *graph isomorphism problem*, involving the task of deciding whether between two given graphs exists an adjacency-preserving one-to-one correspondence on respective vertex sets or not, which can then be juxtaposed to isomorphism between initial, non-graph objects. Security tools like fingerprint scanners, facial scanners, retina scanners, or analysis of social/business structures, especially in context of social networks, would greatly benefit of fast isomorphism-algorithms and profit of efficient tracking of relationship-properties. *Graph isomorphism* and its exact computational categorisation was long deemed a stale topic, with research content in mostly improving known results with regard to their time complexity bounds. In 2015 however, Hungarian researcher László Babai sparked a new flame in the cold lands of graph isomorphism complexity, when he provided the first quasipolynomial time algorithm for determining isomorphisms between two given graphs.
Our main goal is to study isomorphisms on finite structures, their relation to different graph isomorphism types and methods to encode those structures property-preserving into graphs. We are particularly interested in corresponding decision problems and possible polynomial equivalence to the graph isomorphism problem. That is, if a polynomial solution of such problem yields a polynomial decision algorithm for determining graph isomorphisms and vice versa. This will not be done in an exclusively theoretical way, but rather we work on two explicit types of structures, namely formal grammars, more specifically context-free grammars, and term rewriting systems, on which we will present general methods and ideas. In both cases, the aim is to analyse syntactical properties, induced by symbol manipulation, with regard to their computationally complexity and their relation to the graph isomorphism problem. The presented methods can then be generalised to tackle similar problems. This can be of interest in the context of compilers, where formal grammars/languages appear and detection of isomorphism may aid in performance optimisation. Similar for term rewriting systems, who see usage in computer-aided applications and yearly termination competitions. In the latter, new, preferably non-isomorphic, examples are yearly added to a *termination problems data base*. This prevents participating tools from gaining an advantage by tailoring or overfitting to previous data bases.

We will proceed as follows:

Chapter 2: Starts by giving a quick overview of key definitions regarding graph theory and provides a complexity theoretical categorisation of the computational effort to determine isomorphisms between two given graphs. Furthermore, **GI**-completeness of different graph subclasses is discussed.

Chapter 3: Tackles **GI**-completeness of determining grammar isomorphisms and isomorphic strict interpretations on context-free and regular grammars.

Chapter 4: Specialises results from the previous chapter by studying computational properties of isomorphisms on term rewriting systems.

At all points in this thesis we try to uphold balance between mathematical rigorousness and practical application. In particular, implementation and runtime analysis of certain algorithms will not be explained in too much detail to keep the focus of the reader on the core results and presented methods. If needed, we provide appropriate references to further literature.

2 The Graph Isomorphism Problem

2.1 Mathematical Notations and Graph Theoretical Basics

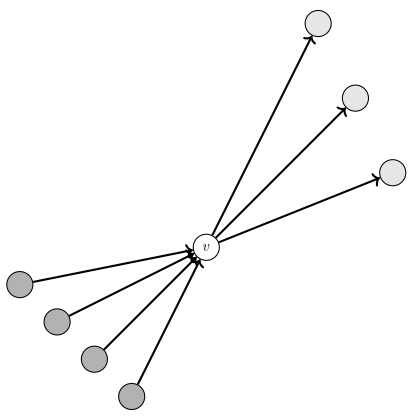
Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ be the set of *positive integers* and consequently $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ the set of *non-negative integers*. *Cardinality* or *order* of a set A is denoted by $|A|$ and we write $A \subseteq B$, if A is a *subset* of an other set B. For distinct elements $a_1, \ldots, a_n$ and elements $b_1, \ldots, b_n$, denote with $\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ the map $f : \{a_1, \ldots, a_n\} \to \{b_1, \ldots, b_n\}$ mapping a_i onto b_i. Moreover, if A and B are disjoint sets, $f : A \to C$ and $g : B \to D$ maps defined on these sets, let $f \sqcup g : A \cup B \to C \cup D$ be the unique map so that $(f \sqcup g)|_A = f$ and $(f \sqcup g)|_B = g$. This can be generalised to arbitrary many, pairwise disjoint, sets. For $f, g : \mathbb{N}_0^m \to \mathbb{R}$, we say that $f = \mathcal{O}(g)$ or f *is in, or of,* $\mathcal{O}(g)$, if we find $C > 0$ and $N \in \mathbb{N}_0$ such that for every $n_1, \ldots, n_m \geq N$, $|f(n_1, \ldots, n_m)| \leq Cg(n_1, \ldots, n_m)$. Lastly, denote for $n \in \mathbb{N}$ with $\mathfrak{S}_n$ the *symmetric group of order* n, i.e. the set of bijections $\{1, \ldots, n\} \to \{1, \ldots, n\}$. The following section gives a quick overview of the most important definitions and terms regarding graph theory that will be used throughout the whole thesis. Acronyms *i.e.*, *w.r.t.* and *w.l.o.g.* and refer to “id est”, “with respect to” and “without loss of generalisation” respectively.

Definition 2.1 (Labelled Directed Graph). A *labelled directed Graph (LDG)* is a tuple $G = (V, E, L, \text{lab})$, where V is a finite set of vertices, $E \subseteq V \times V \times L$ are directed labelled edges between vertices, L is a finite set of labels, sets V, E are disjoint, and $\text{lab} : V \to L$ is a labelling function.

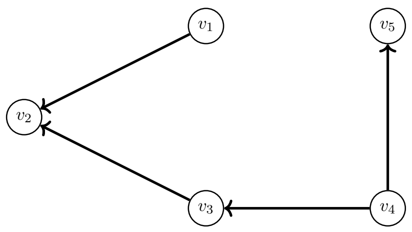
If $|L| = 1$, we say that G is a *directed graph (DG)* and omit L in the definition, $G = (V, E)$. We call G a *labelled undirected graph (LUG)* or simply *labelled graph*, if $(v, w, l) \in E \iff (w, v, l) \in E$. The number of vertices $|V|$ of G is the *order* of G, whilst $|E|$ is called the *size* of G. In order to avoid confusion, we denote an undirected labelled edge by $\langle v, w, l\rangle$. In contrary to other popular literature, we do not allow LDGs to have loops or parallel edges, i.e. for $(v, w, l), (v, w, l') \in E$, $v \neq w$ and already $l = l'$.

Definition 2.2 ((In-/Out-)Degree). For fix vertex $v \in V$, denote with $\text{indeg}(v) = |\{w \in V : (w, v, l) \in E\}|$ and $\text{outdeg}(v) = |\{w \in V : (v, w, l) \in E\}|$ the *indegree* and *outdegree* of v respectively, i.e. the number of ingoing or outgoing edges. Vertices of indegree 0 (and outdegree 1) are called (simple) roots and of outdegree 0 (and indegree 1) (simple) leafs.

If G is undirected, $\text{indeg}(v) = \text{outdeg}(v)$ and we simply call $\deg(v) = \text{indeg}(v)$ the *degree* of v. In this case, the notion of leaf and root agree, and we demand $\deg(v) = 1$.



(a) Vertex v has indegree $\text{indeg}(v) = 4$ and outdegree $\text{outdeg}(v) = 3$, vertices with ingoing edges are light shaded, vertices with outgoing edges dark shaded



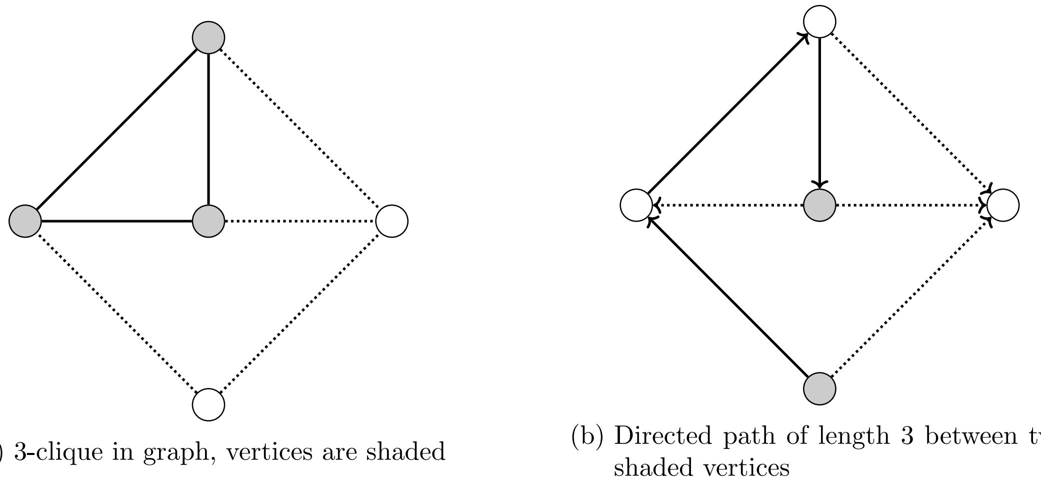
(b) Vertex v_2 is a leaf, v_5 a simple leaf, v_4 a root, v_1 a simple root and v_3 neither

Figure 2.1: Example of (in-/out-)degree and leafs/roots in a directed graph

Definition 2.3 (Complete Graph). G is *complete*, if all vertices are connected, i.e. if we consider the edges to be unlabelled, $E = V \times V$, or more formally, for all $v, w \in V$, $v \neq w$, there is $l, l' \in L$ with $(v, w, l), (w, v, l') \in E$.

Definition 2.4 (Subgraph, Clique). A *subgraph* G' of G is a graph (V', E', L, lab') consisting of a subset $V' \subseteq V$ of V and corresponding edges in E, i.e. $E' = E \cap V' \times V' \times L$, or equivalently, $E' = \{(v, w, l) \in E : v, w \in V'\}$. Vertex labels are preserved, $\text{lab}' = \text{lab}|_{V'}$. A *k-clique* in G is a complete subgraph of order k.

Definition 2.5 (Path). A *(directed) path* in G of *length* $k \in \mathbb{N}$, or *k-path*, is a family of $k + 1$ connected vertices $(v_1, \dots, v_{k+1}) \in V^k$, i.e. we find edges $(v_i, v_{i+1}, l_i) \in E$, $1 \leq i \leq k$. We say that vertices $v, w \in V$ are *k-connected*, if we find a k-path starting in v and ending in w, i.e. $v_1 = v$ and $v_{k+1} = w$. *Adjacent* vertices are 1-connected vertices. For simplicity sake we assume, as long as not stated otherwise, that all vertices in a path are unique, or in other words, every vertex is only visited once.


(a) 3-clique in graph, vertices are shaded

(b) Directed path of length 3 between two shaded vertices

Figure 2.2: Example of k-clique and k-path ($k = 3$)

We will make heavy use of *breadth-first search*, *lexicographical sorting* and *maps* to prove polynomial properties of certain algorithms. Breadth-first search on a graph $G = (V, E)$ visits every vertex (and edge) in at most $\mathcal{O}(|V| + |E|)$, by exploring G starting at an arbitrary vertex. Any set A of strings, can be sorted lexicographically in $\mathcal{O}(cn \log n)$, where n is the size of A and c is an upper bound on the length of the strings in A. Consequently, a map $f : B \to A$ into A can be managed as heap, with insertion (i.e. assigning values) and access being possible in $\mathcal{O}(c \log m)$, where c is again an upper bound on the length of the strings in A, and m is the size of B. Refer to [CLRS09] for concrete implementation of said algorithms, including a comprehensive analysis of their runtimes.

2.2 The Graph Isomorphism Problem and Complexity Class GI

2.2.1 Outline on basic Complexity Theory

Object of the thesis will be so called *decision problems.* Formally, a decision problem is the task to decide, whether a given object a belongs to a certain set A or not, i.e. if $a \in A$. Such a problem can be posed as a yes-no question on the input values and be solved in the form of an *algorithm.* All of the here presented decision problems will be based on finding pairs of finite structures correlated by bijective functions with certain structure preserving properties, and thus can be solved for all input values in finite time by simply checking all such possible functions of which exist only finitely many. Due to theoretical computer science working with countable structures, all decision problems can be regarded as integer subsets, $A \subseteq \mathbb{N}$. This can be accomplished by, for example, *Gödel numbering.* We are however not interested in decidability of decision problems, but rather their *computational (time) complexity*, that is, how computationally difficult they are. Decision problems then can be agglomerated to so called *complexity classes*, i.e. sets of problems. Formal definition of a complexity class $\mathcal{C}$ is object to three things:

(1) A type of computational problem,

(2) a model of computation and

(3) a bounded computational resource, which will be time.

Our computation model of choice will be *Turing machines* represented by simple pseudocode algorithms. This is supported by the *Church-Turing thesis.* Time is then measured by the number of individual, elementary steps dependent on input size n in those algorithms. We do not want to go too in-depth in formal definitions, since satisfactory study of those subjects would blow up the capacity of this thesis. Instead we recommend [Sch08] for a more rigorous approach. From now on, (complexity) classes refer to time complexity classes, i.e. the bounded computational resource will be time if not explicitly stated otherwise. We say problem A is in class $\mathcal{C}$, if $A \in \mathcal{C}$ and in $\mathbf{co}\,\mathcal{C}$, if its complement $\overline{A}$ is in $\mathcal{C}$, $\overline{A} \in \mathcal{C}$.

For function $f : \mathbb{N} \to \mathbb{N}$, let $\mathbf{TIME}(f(n))$ be the class of all problems which can be deterministically solved in time $\mathcal{O}(f(n))$, i.e. $A \in \mathbf{TIME}(f(n))$ if and only if we find a deterministic algorithm which solves A (can decide whether $a \in A$ or not) in at most $\mathcal{O}(f(n))$ steps, where n is the size of the input value, for example the length of a word, or order/size of a graph. Consequently, let $\mathbf{NTIME}(f(n))$ be the class of all problems which can be nondeterministically solved in time $\mathcal{O}(f(n))$. The most prominent representatives of complexity classes are $\mathbf{P} = \bigcup_{k\in\mathbb{N}_0} \mathbf{TIME}(n^k)$ and $\mathbf{NP} = \bigcup_{k\in\mathbb{N}_0} \mathbf{NTIME}(n^k)$, the classes of problems decidable in (non)deterministic polynomial time. Definitions then can be extended to include multivariate polynomials, to cover algorithms with multiple inputs, as $|V|, |E|$ in the case of graphs. Clearly $\mathbf{P} \subseteq \mathbf{NP}$, $\mathbf{TIME}(1) \subseteq \mathbf{TIME}(n) \subseteq \mathbf{TIME}(n^2) \subseteq \ldots \subseteq \mathbf{P}$ and $\mathbf{NTIME}(1) \subseteq \mathbf{NTIME}(n) \subseteq \mathbf{NTIME}(n^2) \subseteq \ldots \subseteq \mathbf{NP}$. The latter two hierarchies do not collapse in the sense that there is no integer $k \in \mathbb{N}$ such that $\mathbf{TIME}(n^k) = \mathbf{P}$ or $\mathbf{NTIME}(n^k) = \mathbf{NP}$ and a direct consequence of a collection of various statements regarding time-bounded computation on Turing machines, called the *Time hierarchy theorem* [HS65, Coo72]. If inclusion $\mathbf{P} \subseteq \mathbf{NP}$ is strict or not, is widely known as the famously unsolved $\mathbf{P} = \mathbf{NP}$ *problem*, first introduced in 1971 by Stephen Cook [Coo71] and an important consequence of the *Cook–Levin theorem*, which states the *Boolean satisfiability problem* is $\mathbf{NP}$-complete.

2.2.2 Polynomial Reductions and Complete Problems

Reduction of one problem to another and *hardness/completeness* of problems relative to complexity classes play a major role in not only theoretical research, but specifically in this thesis. We are primarily interested in problems which are “equivalent” to deciding, whether two given graphs are isomorphic, or not.

Definition 2.6 ((Polynomial) Reduction). Let $A, B \subseteq \mathbb{N}$ be two integer-valued sets (or decision problems). We say that A can be *(many-one-)reduced* to B, in terms $A \preceq B$, if we find a *computable* total function $\rho : \mathbb{N} \to \mathbb{N}$ such that $a \in A \iff \rho(a) \in B$. If f is even computable in polynomial time, i.e. $\rho \in \mathbf{P}$, then we write $A \preceq^{\mathbf{P}} B$. Consequently, A and B are polynomially equivalent, $A \cong^{\mathbf{P}} B$, if $A \preceq^{\mathbf{P}} B$ and $B \preceq^{\mathbf{P}} A$. Both $\preceq$ and $\preceq^{\mathbf{P}}$ are clearly reflexive and transitive.

Note that the notion $\rho \in \mathcal{C}$, for some complexity class $\mathcal{C}$ and function $\rho : \mathbb{N} \to \mathbb{N}$, is no contradiction to our earlier characterisation, but rather an abuse of notation. That is, $\rho \in \mathcal{C}$ if and only if $\{(n, \rho(n)) : n \in \mathbb{N}\} \in \mathcal{C}$, i.e. we identify the function ρ with its function graph. For our purposes, ρ will be mostly given either directly or indirectly by a pseudocode algorithm, and thus ρ is computable in polynomial time if and only if the algorithm itself has runtime of $\mathcal{O}(p)$ for some, possibly multivariate, polynomial p.
The driving motivation behind reductions is twofold. On the one hand, there is strict complexity theoretical information gain. For $A \preceq B$, every instance of problem A can be solved by an algorithm of B, after applications of suitable, computable encoding. If we additionally demand time constraints on the runtime of such an encoding, we can derive a wide facet of information about complexity classification of A just from algorithmic properties of B. On the other hand, reductions can be informally used to convert practical and concrete tasks into mathematically manageable problems on which we can then apply theoretical results. An obvious example is graph theoretical abstraction and modelling of traffic-related problems. Cities, villages or other relevant traffic nodes can be interpreted as vertices, connected by suitably chosen edges.

Definition 2.7 (Hardness, Completeness). Let $\mathbf{P} \subseteq \mathcal{C}$ be a complexity class and $A \subseteq \mathbb{N}$ a decision problem. Then A is *$\mathcal{C}$-hard*, if $B \preceq^{\mathbf{P}} A$ for all $B \in \mathcal{C}$, i.e. every problem B in $\mathcal{C}$ can be polynomially reduced to A. If additionally $A \in \mathcal{C}$, then A is called *$\mathcal{C}$-complete*. Notions of *hardness* and *completeness* can be generalised to other type of classes, requiring reductions of differing complexity. For our purposes the above definition is adequate.

$\mathcal{C}$-complete problems can be understood as computational representatives of a class, that is to say, solving a $\mathcal{C}$-complete problem A, i.e. stating an algorithm, already solves all other problems in $\mathcal{C}$. Properties of complete problems then can be easily generalised to properties of the whole class. Theoretical computer science is interested in **NP**-complete problems, where we additionally demand reductions to be polynomial. Richard Karp's paper [Kar72] published in 1972 was a major step-stone in studying the relation between classes **P** and **NP**, building on Cook's results [Coo71] from one year prior and proving **NP**-completeness of in total 21 combinatorial and graph theoretical computational problems.

2.2.3 GI and its Relation to other Complexity Classes

Definition 2.8 ((Strong) Graph Isomorphism)**.** Given two labelled directed graphs $G_i = (V_i, E_i, L_i, \mathrm{lab}_i)$, $i = 1, 2$, and a bijection $\psi : L_1 \to L_2$, a *graph isomorphism* $\phi : V_1 \to V_2$ w.r.t. ψ, is an adjacency and label preserving bijection, i.e. $\psi(\mathrm{lab}_1(v)) = \mathrm{lab}_2(\phi(v))$ for all $v \in V_1$ and $(v, w, l) \in E_1$ if and only if $(\phi(v), \phi(w), \psi(l)) \in E_2$. We call G_1 and G_2 *isomorphic* w.r.t. ψ. If $L_1 = L_2 = L$ and ψ is the identity on L, then we call ϕ a *strong isomorphism* and G_1 and G_2 *strongly isomorphic.*

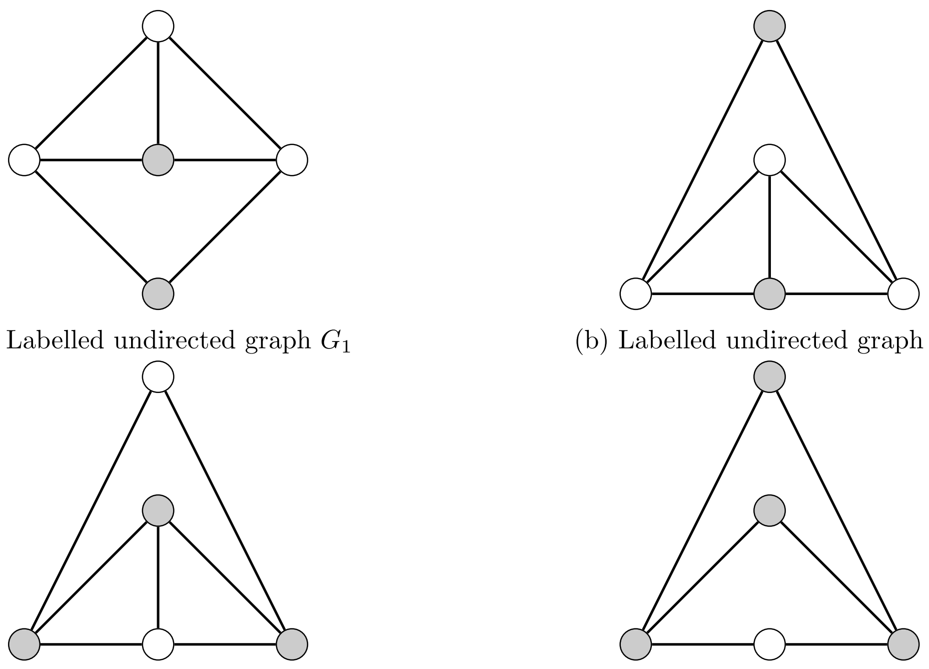
The notion of (strong) isomorphism is appropriately generalised to other subclasses of graphs, like unlabelled (un)directed graphs. See Figure 2.3 for a few examples. Intuitively, two graphs are strongly isomorphic if they are the same after "moving around" vertices, preserving adjacency and labels. Of course, this assumes a certain geometrical abstraction and approach that makes determining (strong) isomorphism of given graphs of arbitrary order and size so difficult and computationally intensive in the first place. This is the case for G_2 in Figure 2.3b, which can be built out of G_1 in Figure 2.3a by swapping top and middle vertex and rotating the resulting graph by 180 degrees. Graph Isomorphism then generalises this concepts by only demanding preservation of equivalence classes of labels. For example G_2 in Figure 2.3b and G_3 in Figure 2.3c are the same graph up to (global) recolouring of vertices, i.e. swapping labels. Just from Definition 2.8, Definition 2.4 and Definition 2.5, we immediately see that images of k-cliques or $k+1$-paths under a (strong) graph isomorphism are again k-cliques or $k+1$-paths, and that restriction of (strong) graph isomorphisms to subgraphs are also (strong) graph isomorphisms. This will be (implicitly) used all throughout this thesis. At this point we can finally discuss the *graph isomorphism problem* and correlated complexity class.

Definition 2.9 (Graph Isomorphism Problem (**GI**))**.** The *graph isomorphism problem (***GI***)* is the task to decide whether two (unlabelled undirected) graphs are isomorphic. Moreover, denote with **GI** the decision problem itself, i.e

$$\mathbf{GI} = \{(G_1, G_2) : G_1, G_2 \text{ isomorphic graphs}\},$$

and the complexity class of to graph isomorphism polynomially reducible problems, that is, the set of problems A such that $A \preceq^{\mathbf{P}} \mathbf{GI}$. Which of the two definitions is used will be clear by context. Since $\mathbf{P} \subseteq \mathbf{GI}$, we can apply Definition 2.7 and discuss **GI**-complete problems.

The complexity class **GI** is only known to be between **P** and **NP**. The latter inclusion is shown pretty easily, as one can guess an one-to-one correspondence between vertex sets and then verify in polynomial time whether adjacency property holds or not. Just from Definition 2.9 we can derive two important relations in regard to the exact classification of **GI**: If **GI** is solvable in polynomial time, then already $\mathbf{GI} = \mathbf{P}$. On the other hand, $\mathbf{GI} = \mathbf{NP}$ if it were to be shown that **GI** is additionally in fact



(a) Labelled undirected graph G_1

(b) Labelled undirected graph G_2

(c) Labelled undirected graph G_3

(d) Labelled undirected graph G_4

Figure 2.3: Example of (strongly) isomorphic labelled undirected graphs

(Edges are unlabelled, vertex-labels are visualised by different shading) G_1 and G_2 are strongly isomorphic, G_1 and G_3 are isomorphic, G_1 and G_4 are not (strongly) isomorphic

NP-complete. The latter assumption is one of the 12 open problems listed in Garey's and Johnson's famous textbook *Computers and Intractability: A Guide to the Theory of NP-Completeness* [GJ79] published in 1979. Just a few years prior in 1975, Richard E. Ladner proved *Landner's theorem* [Lad75], stating that if $\mathbf{P} \neq \mathbf{NP}$, there indeed exist problems neither in **P** nor **NP**. As still unclassified problem, **GI** is thus a prime candidate for a so called **NP**-*intermediate problem*, with strong evidence suggesting that indeed $\mathbf{GI} \neq \mathbf{P}$ and $\mathbf{GI} \neq \mathbf{NP}$. A wide variety of important special cases have efficient, polynomial time solutions, like trees [ZKT85], planar graphs [HW74], permutation graphs [Col81] or graphs with bounded parameters [Bod90], [Luk82], for example bounded (in-/out-)degree, which mostly exploit the combinatorial structure of said classes by using combinatorial heuristics such as individualisation and refinement. There is however no hope of generalising those achievements in favour of $\mathbf{GI} = \mathbf{P}$, as shown in 1992 by Cai, Fürer & Immermann [CFI92], who state that combinatorial refinement methods can not succeed in less than exponential time for the general graph isomorphism problem. Moreover, graph isomorphism can be considered an *universal problem*, in the sense that "isomorphism" between finite combinatorial structures can be remodelled to isomorphism between graphs, and thus solving **GI** with a fast polynomial algorithm

would also solve all those problems relatively fast. We will see some examples of this in Chapter 3 and Chapter 4, when we prove **GI**-completeness of certain isomorphism between *formal grammars* and *term rewriting systems* respectively. Since this kind of *universality* usually implies computational"hardness", it is relatively unlikely for **GI** to be in **P**.
Other algorithmic approaches interconnect graph theory with group theory, build on the fact that computation of the automorphism group of a graph is computationally equivalent to **GI**. This is supported by Frucht [Fru39], who proved that every finite group is isomorphic to the automorphism group of some graph. In 1983, Babai & Luks [BL83] published an algorithm with runtime $2^{\mathcal{O}(\sqrt{n \log n})}$ for graphs of order n, based on a combination of Luks's algorithm for graphs of bounded degree [Luk82] from 1982 and Zemlyachenko's degree reduction technique [ZKT85], which relies on *classification of finite simple groups (CFSG)*. The latter states that every finite simple group, that is, a finite group whose normal subgroups are the trivial group and itself, is either a member of one of three infinite classes (cyclic groups, alternating groups, groups of Lie type) or one of 27 so called sporadic groups. This classification theorem however was announced in 1983 but not completely proven until much later in 2004 when Aschbacher & Smith solved the last remaining special case and 2008 when Harada & Solomon filled a minor gap. This worst case-bound was improved on for a lot of subclasses, for example in 1996 by Spielman [Spi96], demonstrating that isomorphism of *strongly regular graphs* may be tested in $n^{\mathcal{O}(n^{1/3} \log n)}$, or in 2008 by Babai & Codenotti [BC08] for *hypergraphs of bounded rank*.
The biggest break-trough in recent times came in November 2015 from Hungarian researcher László Babai [Bab16] (last revised in January 2016), who announced a quasipolynomial time algorithm for **GI** and related problems of *String Isomorphism*, apparently improving the worst case-bound down to $2^{\mathcal{O}((\log n)^c)}$ for some fixed $c > 0$. This claim however was invalidated by an error in the analysis of the graph isomorphism test, first pointed out by Harald Helfgott, whereupon on January 4, 2017, Babai retracted his paper. Just a few days later on January 9, after "discovering a replacement for the recursive call in the 'Split-or-Johnson' routine that had caused the problem" on January 7, Babai published an updated, revised version [Bab17] restoring his claim, with Helfgott himself confirming the fix. This was furthermore improved on by Helfgott [HBD17], claiming that one can choose in fact $c = 3$. As of now, both (updated) proofs have not been fully peer-reviewed yet. Babai's result is one of three main arguments against **NP**-completeness of **GI**.

Argument 1: **GI** is solvable in quasipolynomial time

As mentioned before, **GI** was shown to be solvable in quasipolynomial time by Babai. Formally, the complexity class of subexponential time is given as **SUBEXP** $= 2^{o(n)}$, $\bigcap_{\varepsilon>0}$ **TIME**$(2^{n^{\varepsilon}})$ or simply **TIME**$(2^{poly(\log n)})$. That is, an algorithm of subexponential time, runs longer than polynomial time, but still not as long as exponential time, indicated by the small o in the exponent of the first definition. Now **GI** being **NP**-complete would immediately imply **NP** $\subseteq$ **SUBEXP**, i.e. all **NP** problems are solvable in quasipolynomial time. This however, would contradict the yet unproven but widely popular*Exponential time hypothesis* formulated by Impagliazzo & Paturi [IP99] in 1999. The hypothesis states that **3-SAT**, i.e. the problem of deciding whether a boolean formula in conjunctive normal form with at most three literals in each clause is satisfiable, and other **NP**-complete problems cannot be solved in subexponential time. One of the best currently known deterministic bounds for **3-SAT** is only in $\mathcal{O}(1.439^n)$ [KS10] with randomised algorithms even cracking $\mathcal{O}(1.321^n)$ [HMS11].

Argument 2: The polynomial time hierarchy would collapse

For some complexity class $\mathcal{C}$ and some decision problem A, denote with $\mathcal{C}^A$ the class of problems solvable by an algorithm in class $\mathcal{C}$ with an *oracle* for A, generalised to $\mathcal{C}^\mathcal{D} = \bigcup_{A\in\mathcal{D}} \mathcal{C}^A$ for class $\mathcal{D}$. Conceptually, oracle machines, or algorithms with an oracle for that matter, can use oracles to solve certain problems in *one step.* For example an oracle for **GI** could decide in one step whether two given graphs are isomorphic or not. Now the *polynomial hierarchy* is inductively defined the following way: $\Delta_0^\mathbf{P} = \Sigma_0^\mathbf{P} = \Pi_0^\mathbf{P} = \mathbf{P}$ and $\Delta_{i+1}^\mathbf{P} = \mathbf{P}^{\Sigma_i^\mathbf{P}}$, $\Sigma_{i+1}^\mathbf{P} = \mathbf{NP}^{\Sigma_i^\mathbf{P}}$, $\Pi_{i+1}^\mathbf{P} = \mathbf{coNP}^{\Sigma_i^\mathbf{P}}$. For example, $\Sigma_1^\mathbf{P} = \mathbf{NP}^\mathbf{P} = \mathbf{NP}$ and $\Delta_2^\mathbf{P} = \mathbf{P}^\mathbf{NP}$. This immediately implies relations like $\Sigma_i^\mathbf{P} \subseteq \Delta_{i+1}^\mathbf{P} \subseteq \Sigma_{i+1}^\mathbf{P}$ or $\Pi_i^\mathbf{P} \subseteq \Delta_{i+1}^\mathbf{P} \subseteq \Pi_{i+1}^\mathbf{P}$, which are widely believed to proper inclusions. In 1988 however, Uwe Schöning [Sch88] proved that if **GI** is **NP**-complete, this hierarchy collapses to the second level, forcing above inequalities to be in fact equalities for $i \geq 1$. Refer to [vM50] by van Melkebeek and [Sto76] by Stockmeyer for a more rigorous introduction to oracle machines and polynomial hierarchy respectively.

Argument 3: **GI** is in **coAM**

Complexity class **AM** is the set of decision problems that can be decided in polynomial time by an *Arthur–Merlin protocol* with two messages, which was first introduced by Babai [Bab85] in 1985. Conceptually, this is comparable to a *blind taste test* or *zero-knowledge proof*, with two participants *Arthur* and *Merlin.* Arthur is assumed to be a standard computer, i.e. the user itself, equipped with a random number generator, for example a fix number of coins that can be tossed, and Merlin represents an all-knowing oracle, i.e. a computer with infinite computing power. Now messages between the two are exchanged. Arthur tosses his coins (not necessarily all) and sends the outcome of all tosses to Merlin, without conveying additional information apart from the values itself. Merlin then answers with a supposed proof that is deterministically verified by Arthur, i.e. he either accepts or rejects the given proof. Problems are then solvable by such protocols if possibility of acceptance exceeds a certain threshold in the case of a yes-instance. No such protocol however has been found for any known **NP**-complete problem, again pointing towards **GI** not being **NP**-hard.

2.3 GI-Complete Subclasses of Graphs

We will discuss a wide variety of different graph subclasses and show that in fact (strong) isomorphism on these subclasses is already **GI**-complete. In particular labelled directed graphs will be of use in the coming chapters, since encoding finite structures into graphs tends to be much easier if you are allowed to actually to use symbols instead of trying to encode everything via isomorphically unique subgraphs.

Theorem 2.10. *The following sets are polynomially equivalent:*

(*i*) $\mathbf{GI} = \{(G_1, G_2) : G_1, G_2 \textit{ isomorphic undirected graphs}\}$

(*ii*) $\mathbf{DG} = \{(G_1, G_2) : G_1, G_2 \textit{ isomorphic directed graphs}\}$

(*iii*) $\mathbf{Pseudo} = \{(G_1, G_2) : G_1, G_2 \textit{ isomorphic pseudographs}\}$

(*iv*) $\mathbf{StrLUG} = \{(G_1, G_2) : G_1, G_2 \textit{ strongly isomorphic LUGs}\}$

(*v*) $\mathbf{StrLDG} = \{(G_1, G_2) : G_1, G_2 \textit{ strongly isomorphic LDGs}\}$

(*vi*) $\mathbf{LDG} = \{(G_1, G_2) : G_1, G_2 \textit{ isomorphic LDGs}\}$

(*vii*) $\mathbf{StrOOLDG} = \{(G_1, G_2) : G_1, G_2 \textit{ strongly isomorphic OOLDGs}\}$

(*viii*) $\mathbf{OOLDG} = \{(G_1, G_2) : G_1, G_2 \textit{ isomorphic OOLDGs}\}$

Proof. Immediately follows by a chain of polynomial reductions as depicted in Figure 2.4. Unlabelled implications are trivial implications based on subclasses and identity encoding. Definitions of pseudograph and OOLDG will be given later in the section, refer to Definition 2.12 and Definition 2.15. ■

$$\begin{tikzcd}
(iv)\ \mathbf{StrLUG} \arrow[r, Rightarrow, "\text{Proposition 2.14}"] \arrow[d, Leftrightarrow, "\text{Proposition 2.17}"] & (iii)\ \mathbf{Pseudo} \arrow[r, Rightarrow, "\text{Proposition 2.13}"] & (ii)\ \mathbf{DG} \arrow[d, Rightarrow, "\text{Proposition 2.16}"] \arrow[dl, Leftrightarrow, "\text{Proposition 2.11}"] \arrow[dd, Rightarrow, bend left, "\text{Proposition 2.16}"] \\
(v)\mathbf{StrLDG} & (i)\ \mathbf{GI} \arrow[u, Rightarrow] & (vii)\ \mathbf{StrOOLDG} \arrow[ll, Rightarrow, bend left] \\
(vi)\ \mathbf{LDG} \arrow[u, Rightarrow, "\text{Proposition 2.18}"] & & (viii)\ \mathbf{OOLDG} \arrow[ll, Rightarrow]
\end{tikzcd}$$

Figure 2.4: Structure of the proof of Theorem 2.10

2.3.1 Edge Replacement Techniques

For the first part of the proof of Theorem 2.10 we will take a look at so called *edge replacement techniques* to encode edge-direction and/or edge-labels. The basic idea is always the same: To encode the original graph into a graph of different class, replace edges $e = (v, w)$ with isomorphically unique substructures consisting of a linearly-edge-depending number of vertices $v_{e,1}, \dots, v_{e,K}$ connected to v and w. That is, the edge replacement is chosen in such a way that the original vertices can be recovered uniquely up to isomorphism. This can be done by traversing the graph breadth-first (always terminates since our graphs are finite) and then gradually replacing edges connecting old vertices. For that, we need to extend standard breadth-first search by adding more labels to distinguish between newly inserted vertices $v_{e,i}$ and original vertices. Since our substructures are of either constant (Proposition 2.11, Proposition 2.16) or bounded size (Proposition 2.13) and only depend linearly on given edge (Proposition 2.17), this can be done it at worst $\mathcal{O}(|V| + c|E|)$ where c is some fix constant. Refer to [CLRS09, pp. 594–600] for an in-depth analysis of breadth-first search and Algorithm 2.1.

Algorithm 2.1: Computation of linear edge replacement encoding λ

```
input   : Labelled directed graph $G = (V, E, L, \mathrm{lab})$, edge replacement enc. $\lambda$
output  : Labelled directed graph $\lambda(G) = (V', E', L', \mathrm{lab}')$ according to $\lambda$
runtime: $\mathcal{O}(|V| + c|E|)$, $c$ upper bound on size of replacement structure
1  Initialise $E' \leftarrow E$, $V' \leftarrow V$, $\mathrm{lab}' \leftarrow \mathrm{lab}$
2  Initialise $L'$ according to $\lambda$
3  Initialise Visit $\leftarrow V$                                // Unseen vertices
4  while Visit $\neq \emptyset$ do                                // $\mathcal{O}(|V| + c|E|)$
5      Choose $w \in$ Visit arbitrary
6      Initialise empty queue $Q$
7      Remove($w$, Visit)
8      Push($w$, $Q$)
9      while $Q$ is not empty do
10         Set $v \leftarrow$ Pop($Q$)
11         for edge $(v, v', l) \in E$ do
12             Replace $(v, v', l)$ according to $\lambda$        // bounded by constant $c$
13             Update $V'$, $E'$, $\mathrm{lab}'$ accordingly
14             if $v' \in$ Visit then
15                 Remove($v'$, Visit)
16                 Push($v'$, $Q$)
17             end
18         end
19     end
20 end
21 return $\lambda(G) = (V', E', L', \mathrm{lab}')$
```

Proposition 2.11 ($(ii) \implies (i)$, motivated by [BC79]). $\mathbf{DG} \preceq^{\mathbf{P}} \mathbf{GI}$

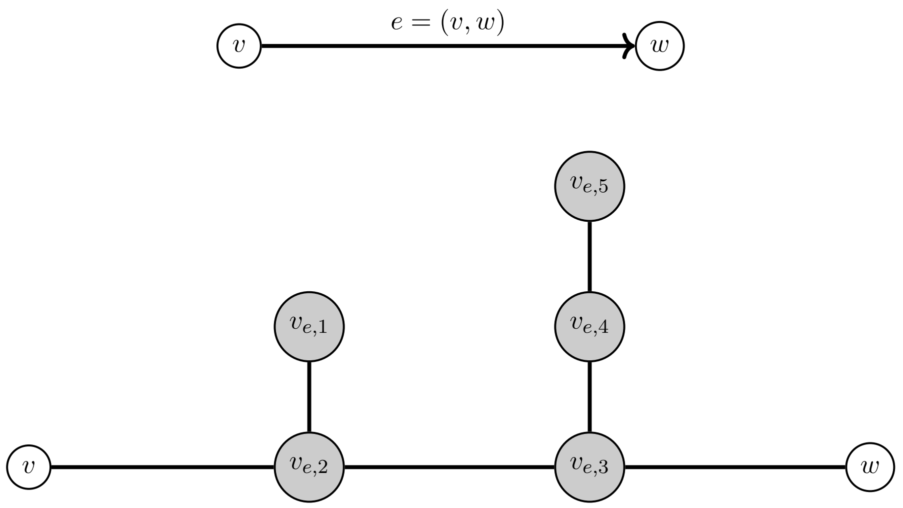
Proof. Define an encoding η of a directed graph into an undirected graph the following way: Given a directed graph $G = (V, E)$, replace every directed edge $e = (v, w) \in E$ by an undirected 5-vertex-structure as seen in Figure 2.9a. That is, introduce paths of length 4 consisting of fresh vertices $v_{e,1}, v_{e,2}, v_{e,3}, v_{e,4}, v_{e,5}$ such that tail v is connected to $v_{e,2}$ and head w to $v_{e,3}$, i.e. the second and third vertex in the path respectively. More formally, $\eta(G) = (V', E')$, where $V' = V \cup \{v_{e,i} : e \in E, 1 \leq i \leq 5\}$ and

$$E' = \{\langle v_{e,i}, v_{e,i+1}\rangle : e \in E, 1 \leq i \leq 4\} \cup \{\langle v, v_{e,2}\rangle, \langle w, v_{e,3}\rangle : e = (v, w) \in E\}. \qquad (2.1)$$

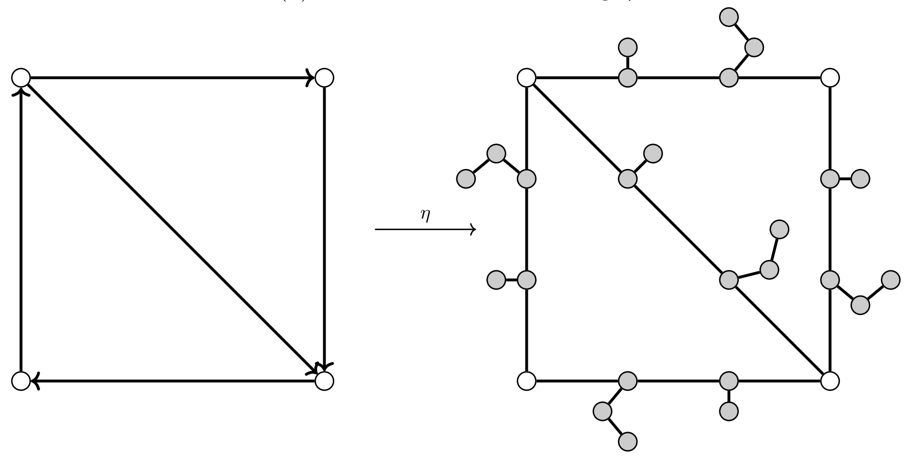
First note that every leaf, whose neighbour has degree 2, is of the form $v_{e,5}$ for some edge $e \in E$. Indeed, if $\deg(v) = 1$ for some vertex $v \in V'$, and $v \neq v_{e,5}$, then either already $v \in V$ and $\text{indeg}(v) = 1$ or $\text{outdeg}(v) = 1$, i.e. a simple leaf/root in the original graph G, or $v = v_{e,1}$. Adjacent vertices are either $v_{e,2}$ or $v_{e,3}$, both of degree 3 in $\eta(G)$. Moreover, $v_{e,4}$ is the unique neighbour of $v_{e,5}$, $v_{e,3}$ the unique neighbour of $v_{e,4}$, and $v_{e,2}$ the unique neighbour of $v_{e,3}$ of degree 3 adjacent to a leaf in $\eta(G)$. Vertex $v_{e,1}$ is not uniquely recoverable however. If $e = (v, w) \in E$ is the unique outgoing edge of some simple root $v \in V$, then v and $v_{e,1}$ are both adjacent vertices of $v_{e,2}$ of degree 1, refer to Figure 2.9a. Now two directed graphs G_1, G_2 are isomorphic if and only if their undirected encodings $\eta(G_1), \eta(G_2)$ are isomorphic.
The first implication is clear. Extended an isomorphism $\phi : V_1 \to V_2$ between G_1 and G_2 to an isomorphism $\phi' : V_1' \to V_2'$ between $\eta(G_1)$ and $\eta(G_2)$ via $\phi'(v_{e,i}) = v_{e',i}$, where $e' = (\phi(v), \phi(w))$ if $e = (v, w) \in E$, $1 \leq i \leq 5$.
On the other hand, let an isomorphism $\phi' : V_1' \to V_2'$ be given. By earlier argumentation, $v_{e,i}$-vertices are almost uniquely identifiable. Let

- $R_i = \{v \in V_i : \text{outdeg}(v) = 1, \text{indeg}(v) = 0\}$ be the set of simple roots in G_i, and
- $S_i = \{v_{e,1} : e \in E_i \text{ unique outgoing edge of some } v \in R_i\}$ the corresponding set of outgoing edges, $i = 1, 2$.

Denote with f_i the canonical one-to-one correspondence between R_i and S_i, i.e. for $v \in R_i$, $f_i(v) = v_{e,1} \in S_i$, such that $e \in E_i$, is the unique outgoing edge of v. For any vertex $v \in R_1$, we have that $\{\phi'(v), \phi'(f_1(v))\} = \{v', f_2(v')\}$ for some $v' \in V_2$, but not necessarily $\phi'(v) = v'$. This can easily be corrected by considering map $\phi'' : V_1' \to V_2'$, where $\phi''(v) = v'$ for $v \in R_1$ with same notation as before, i.e. map simple roots in G_1 onto simple roots in G_2 according to ϕ'. This is still an isomorphism between $\eta(G_1)$ and $\eta(G_2)$. Hence there is an, by ϕ'' induced, one-to-one correspondence between the sets $F_i = \{(v_{e,1}, \ldots, v_{e,5}) : e \in E_i\}$, $i = 1, 2$. Then $|V_1| = |V_2|$ and $\phi = \phi''|_{V_1} : V_1 \to V_2$, is well-defined and bijective. Moreover, $|E_1| = |F_1| = |F_2| = |E_2|$. For fix $e = (v, w) \in E_1$, we have $\langle v, v_{e_2}\rangle, \langle w, v_{e_3}\rangle \in E_1'$ by (2.1) and thus $\big\langle \phi(v), v_{e',2}\big\rangle, \big\langle \phi(w), v_{e',3}\big\rangle \in E_2'$ for some edge $e' \in E_2$,by aforementioned correspondence. But this is already equivalent to $(\phi(v), \phi(w)) \in E_2$ and we are done (in particular $e' = (\phi(v), \phi(w))$). ■



(a) Visualisation of encoding η



(b) Graph example of encoding η

Figure 2.5: Example of encoding η in the proof of Proposition 2.11 (new vertices are shaded)

Definition 2.12 (Pseudograph)**.** A *pseudograph* $G = (V, E)$ is an unlabelled undirected graph with possibly multiple edges between vertices and loops, so called *multi-edges/-loops*. Formally, $E \subseteq V \times V \times \mathbb{N}$ and $(v, w, k) \in E$ if and only if $(w, v, k) \in E$, where we allow $v = w$, but not $(v, w, k), (v, w, k') \in E$ for distinct $k, k' \in \mathbb{N}$. Denote edges by $\langle v, w \mid k\rangle$. Then k is the so called *multiplicity* of an edge $\langle v, w\rangle$. Consequently, two pseudographs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if we find adjacency preserving bijection $\phi : V_1 \to V_2$, i.e. $\langle v, w \mid k\rangle \in E_1$ if and only if $\langle \phi(v), \phi(w) \mid k\rangle \in E_2$.

Technically, a pseudograph G is nothing else than an integer-labelled undirected graph. The edge-labels are exactly the multiplicities and loops are transformed to label-function lab : $V \to \mathbb{N}_0$, $\mathrm{lab}(v) = k$, if $\langle v, v \mid k\rangle \in E$, 0 otherwise. The important distinction now comes in the form of geometrical interpretation:

Proposition 2.13 ((*iii*) $\Longrightarrow$ (*ii*), Booth & Colbourn 1979 [BC79]). **Pseudo $\preceq^{\mathbf{P}}$ DG**

Proof. We encode a pseudograph $G = (V, E)$ into a directed graph $\iota(G) = (V', E')$ by replacing multi-edges and multi-loops with directed edges the following way: For each edge $\langle v, w \mid n\rangle = e \in E$, introduce n fresh vertices $v_{e,1}, \dots, v_{e,n}$ to replace e by $2n$ directed edges $(v_{e,i}, v), (v_{e,i}, w)$, or n directed edges $(v_{e,i}, v)$, $1 \le i \le n$, depending on if $v = w$ or not. That is, $V' = V \cup \{v_{e,i} : e \in E, 1 \le i \le n\}$ and $E' = E^{(1)} \cup E^{(2)}$, where

$$E^{(1)} = \{(v_{e,i}, v), (v_{e,i}, w) : \langle v, w \mid n\rangle = e \in E, v \ne w, 1 \le i \le n\},$$
$$E^{(2)} = \{(v_{e,i}, v) : \langle v, v \mid n\rangle = e \in E, 1 \le i \le n\}.$$

Then the original vertex set V is exactly the subset of vertices in $\iota(G)$ of outdegree 0 and every multi-edge $\langle v, w \mid n\rangle \in E$ can be recovered by $n = \{v' \in V' : (v', v), (v', w) \in E'\}$, i.e. the number of vertices placed between to vertices $v, w \in V$, and every multi-loop $\langle v, v \mid n\rangle \in E$ by $n = \{v' \in V' : (v', v) \in E', \mathrm{outdeg}(v') = 1\}$ (if $n = 0$ we omit the edge/loop). One then easily checks that two pseudographs G_1 and G_2 are isomorphic if and only if their respective directed encodings $\iota(G_1)$ and $\iota(G_2)$ are isomorphic. ■

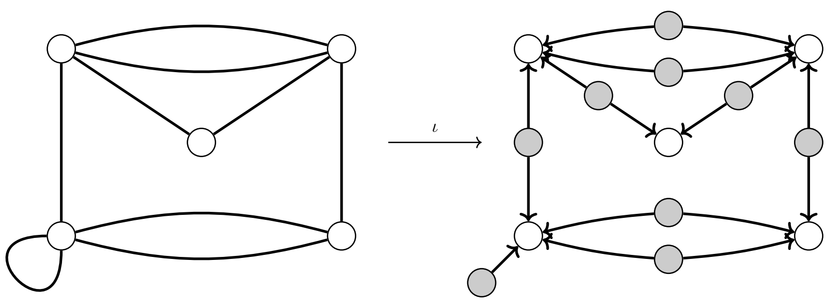


Figure 2.6: Example of encoding ι in the proof of Proposition 2.13 (new vertices are shaded)

Proposition 2.14 ((*iv*) $\Longrightarrow$ (*iii*), Booth & Colbourn 1979 [BC79]). **StrLUG $\preceq^{\mathbf{P}}$ Pseudo**

Proof. This is pretty straight forward. For a labelled undirected graph $G = (V, E, L, \mathrm{lab})$ with ordered label set $L = \{l_1, \dots, l_m\}$, consider $\kappa(G) = (V, E')$ with edges $E' = \{\langle v, w \mid j\rangle : \langle v, w, l_j\rangle \in E\}$. Clearly, κ is an one-to-one encoding from labelled undirected graphs into pseudographs if we extend L to $L' = L \cup \{l_0\}$ for some fresh label l_0 to ensure decoding of vertices without loops. If G is given by an adjacency matrix, κ is as easy as just renaming labels according to their ordering in the given label set, which can be done in $\mathcal{O}(|V|^2 \log |L|)$. Then ϕ is a strong isomorphism between labelled undirected graphs G_1 and G_2 if and only if it is an isomorphism between pseudographs $\kappa(G_1)$ and $\kappa(G_2)$, under the assumption that both label sets are equal and ordered the same way. ■

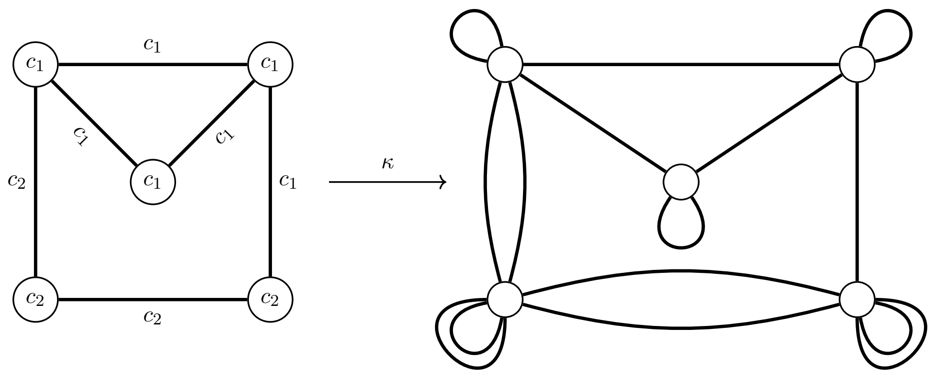


Figure 2.7: Example of encoding κ in the proof of Proposition 2.14 (Label set $L = \{c_1, c_2\}$)

Definition 2.15 (Outgoing-Ordered LDG (OOLDG)). A labelled directed graph $G = (V, E, L, \text{lab})$ is called *outgoing-ordered* or *outgoing-ordered labelled directed graph (OOLDG)* if and only if labels of outgoing edges are unique for each vertex, i.e. whenever $(v, v_1, l), (v, v_2, l) \in E$ then already $v_1 = v_2$.

We refine encoding ι from Proposition 2.13 by introducing labelled directed edges encoding old vertex set membership and original edge direction.

Proposition 2.16 ($(ii) \implies (vii)$ and $(ii) \implies (viii)$, Schmidt-Schauß, Rau & Sabel 2013 [SSRS13]). $\mathbf{DG} \preceq^{\mathbf{P}} \mathbf{StrOOLDG}$ *and* $\mathbf{DG} \preceq^{\mathbf{P}} \mathbf{OOLDG}$

Proof. Recall encoding ι in the proof of of Proposition 2.13. If $G = (V, E)$ is a directed graph, replace every edge $e = (v, w) \in E$ by edges $(v_e, v, 1)$ and $(v_e, w, 2)$, where v_e is a fresh vertex, uniquely introduced for this very edge. Additionally, introduce a single 1-labelled edge $(w_1, w_2, 1)$ between fresh vertices w_1 and w_2. In other words, an encoding ρ is given by $\rho(G) = (V', E', L, \text{lab})$, where $V' = V \cup \{v_e : e \in E\} \cup \{w_1, w_2\}$, $E' = \{(v_e, v, 1), (v_e, w, 2) : (v, w) = e \in E\} \cup \{(w_1, w_2, 1)\}$, $L = \{1, 2\}$ and lab is considered to be trivial.
Two directed graphs G_1 and G_2 are then isomorphic if and only if $\rho(G_1)$ and $\rho(G_2)$ are (strongly) isomorphic. The first part of the proof is trivial, we can even choose a graph isomorphism between $\rho(G_1)$ and $\rho(G_2)$ to be strong. For the if part, consider an isomorphism $\phi' : V_1' \to V_2'$ and a bijection $\psi \in \mathfrak{S}_2$. Unique edges $(w_1, w_2, 1)$ in E_1' and $(w_1', w_2', 1)$ in E_2' force ψ to be the identity, due to necessarily mapping w_i onto w_i' to preserve adjacency (w_1 is the only vertex of outdegree 1).

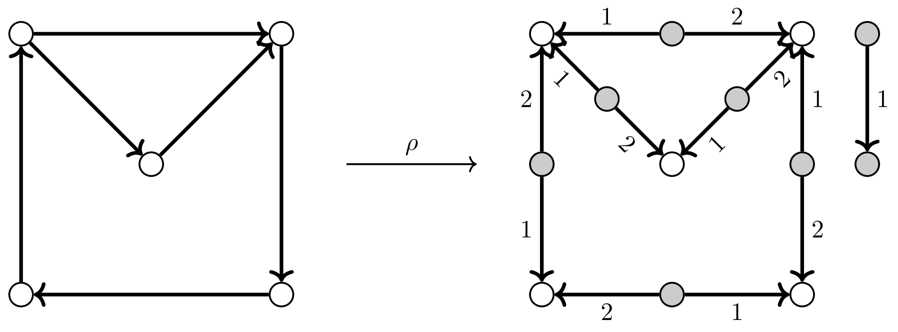


Figure 2.8: Example of encoding ρ in the proof of Proposition 2.16 (new vertices are shaded)

In other words, $\rho(G_1)$ and $\rho(G_2)$ are isomorphic if and only if they are strongly isomorphic and we can proceed by omitting trivial ψ entirely. Now ϕ' maps old vertices onto old vertices. Indeed, $V_1 \cup \{w_2\}$ and $V_2 \cup \{w_2'\}$ are exactly the sets of vertices with outdegree 0, and thus $\phi'(V_1) = V_2$, since already $\phi'(w_2) = w_2'$. For fix $(v_1, v_2) = e \in E_1$, $(v_e, v_1, 1), (v_e, v_2, 2) \in E_1'$ and thus there is an unique $v_{e'} \in V_2' \setminus V_2$ with $\phi'(v_e) = v_{e'}$ and $(v_{e'}, \phi'(v_1), 1), (v_{e'}, \phi'(v_2), 1) \in E_2'$, which corresponds to edge $(\phi'(v_1), \phi'(v_2)) \in E_2$. Moreover, $|E_1| = |V_1' \setminus V_1| - 1 = |V_2' \setminus V_2| - 1 = |E_2|$ and hence $\phi : V_1 \to V_2$, $\phi = \phi'|_{V_1}$ indeed describes an isomorphism between G_1 and G_2. ■

By omitting vertex-labelling entirely, we have in fact proved that isomorphism of unlabelled directed graphs can be reduced to (strong) isomorphism of only-edge-labelled OOLDGs.

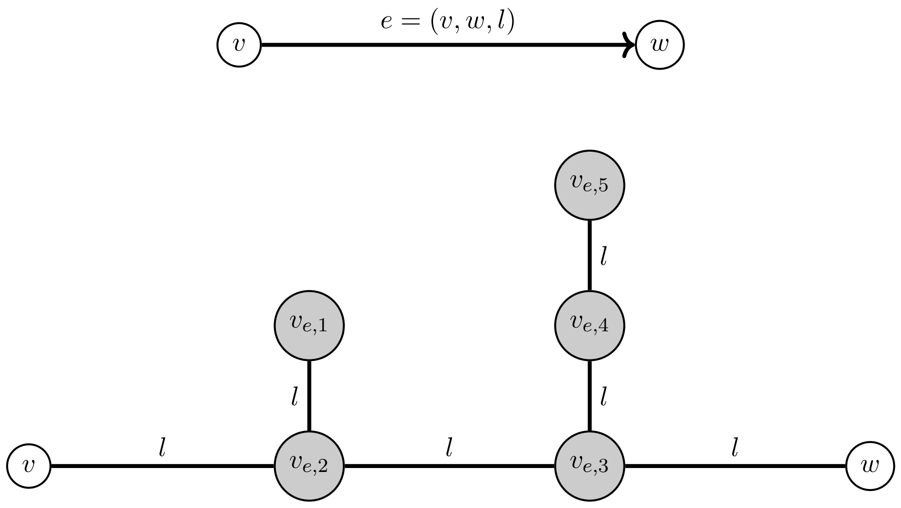
Proposition 2.17 ($(v) \implies (iv)$, motivated by [BC79]). **StrLDG $\preceq^{\mathbf{P}}$ StrLUG**

Proof. Let $G = (V, E, L, \text{lab})$ be an arbitrary labelled directed graph. Recall the proof of Proposition 2.11, where we described an encoding η given a directed graph into an undirected graph. Replace every edge (v, w, l) by the same 5-vertex-structure (depicted in Figure 2.9a), whose edges are additionally l-labelled. That is, $\sigma(G) = (V', E', L \cup \{h\}, \text{lab}')$, where $V' = V \cup \{v_{e,i} : e \in E, 1 \leq i \leq 5\}$,

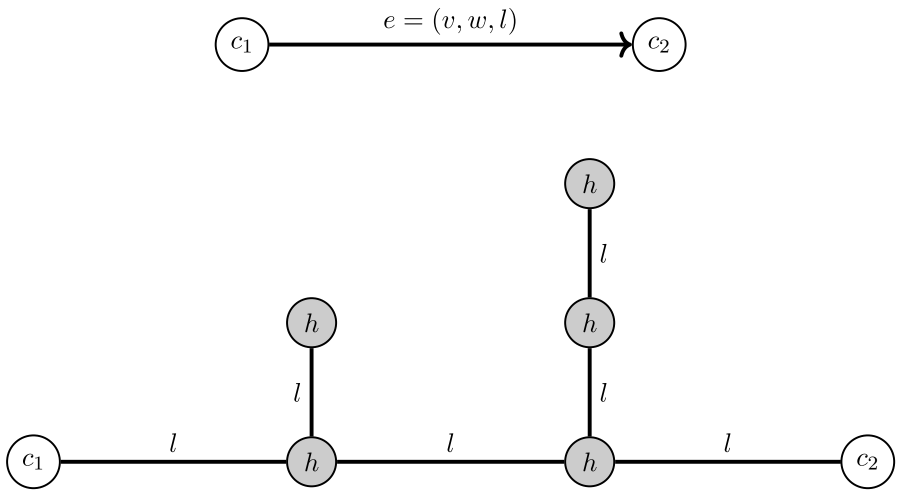
$$\begin{aligned} E' = \{&\langle v_{e,i}, v_{e,i+1}, l\rangle : e = (v, w, l) \in E, 1 \leq i \leq 4\} \\ &\cup \{\langle v, v_{e,2}, l\rangle, \langle w, v_{e,3}, l\rangle : (v, w, l) \in E\}, \end{aligned} \tag{2.2}$$

h is a fresh label, and lab' is extended to all of V' via $\text{lab}'(v_{e,i}) = h$, for each $e \in E$, $1 \leq i \leq n$. Alternatively, endow edge-vertices $v_{e,i}$ with corresponding edge-label, and leave out edge-labels entirely.
We now claim that two labelled directed graphs G_1, G_2 are strongly isomorphic if and only if their LUG-encodings $\sigma(G_1), \sigma(G_2)$ are strongly isomorphic. The first implication is clear. Extend a strong isomorphism $\phi : V_1 \to V_2$ between G_1 and G_2 to a strong isomorphism $\phi' : V_1' \to V_2'$ between $\sigma(G_1)$ and $\sigma(G_2)$ via $\phi'(v_{e,i}) = v_{e',i}$, where $e' = (\phi(v), \phi(w), l)$ if $e = (v, w, l) \in E$, $1 \leq i \leq 5$.
On the other hand, let $\phi' : V_1' \to V_2'$ be a strong isomorphism between $\sigma(G_1)$ and $\sigma(G_2)$.

By proof of Proposition 2.11, restriction $\phi = \phi'|_{V_1}$ is well-defined, bijective and implies an isomorphism between the underlying, edge-unlabelled directed graphs of G_1 and G_2 to respectively. Note that adjustment of ϕ' was not necessary due to labelling of new edge-vertices, i.e. $v_{e,1}, \dots, v_{e,5}$ are uniquely recoverable. We only have left to check that ϕ preserves edge-labels, but this immediately clear by inspecting edge-construction in equation (2.2). ■



(a) Visualisation of general structure of encoding σ (η with additional labelling)



(b) Visualisation of labelling in encoding σ (Labels lab$(v) = c_1$ and lab$(w) = c_2$)

Figure 2.9: Example of encoding σ in the proof of Proposition 2.17 (new vertices are shaded)

2.3.2 Pointer Techniques

For the last step of Theorem 2.10 we want to reduce isomorphism to strong isomorphism between labelled directed graphs, i.e. we need to find a way to encode vertex-labels via uniformly labelled vertices and edges. For that, we introduce so called *pointer techniques*. The core idea is pretty straightforward: Encode labels by isomorphically unique substructures. Since we can use labels, a single vertex with properly chosen label should be enough. Now every original vertex points towards the unique substructure representing its corresponding label.

Proposition 2.18 ($(vi) \implies (v)$). $\mathbf{LDG} \preceq^{\mathbf{P}} \mathbf{StrLDG}$

Proof. Let $G = (V, E, L, \mathrm{lab})$ be a labelled directed graph. We want to give a polynomial encoding $\tau(G) = (V', E', L', \mathrm{lab}')$ into an alternative labelled directed graph such that isomorphism of LDGs is equivalent to strong isomorphism between the encoded LDGs. This is accomplished by introducing vertices corresponding to labels and purging all existing labels in favour of uniform ones, indicating the role/purpose of a vertex in the construction. More precisely, we proceed as follows:

(1) Construct the labelled subdivision graph, i.e. replace every edge $(v_1, v_2, l) = e \in E$ by two edges $(v_e, v_1, 1)$ and $(v_e, v_2, 2)$, where v_e is a fresh vertex with label s, and replace original vertex-labels with fresh label o. Refer to encoding ρ in Proposition 2.16.

(2) Introduce fresh k-labelled vertex v_l for each label $l \in L$.

(3) Encode original vertex-labels with new edges (v, v_l, c) for each $v \in V$, where $l = \mathrm{lab}(v)$.

(4) Encode original edge-labels with new edges (v_e, v_l, w) for each edge $e = (v_1, v_2, l) \in E$.

That is, $L' = \{1, 2, o, s, k, w, c\}$, $V' = V \cup V^{(1)} \cup V^{(2)}$ and $E' = E^{(1)} \cup E^{(2)} \cup E^{(3)}$, where

$$\begin{aligned} V^{(1)} &= \{v_e : e \in E\}, \\ V^{(2)} &= \{v_l : l \in L\}, \\ E^{(1)} &= \{(v_e, v_1, 1), (v_e, v_2, 2) : e = (v_1, v_2, l) \in E\}, \\ E^{(2)} &= \{(v, v_l, c) : v \in V, \mathrm{lab}(v) = l\}, \\ E^{(3)} &= \{(v_e, v_l, w) : e = (v_1, v_2, l) \in E\}, \end{aligned} \tag{2.3}$$

and $\mathrm{lab}'|_V \equiv o$, $\mathrm{lab}'|_{V^{(1)}} \equiv s$, $\mathrm{lab}'|_{V^{(2)}} \equiv k$. An example of this encoding is shown in Figure 2.10. Step (1) is edge replacement, which was shown to be possible in polynomial time way back in the beginning of Subsection 2.3.1. Refer to Algorithm 2.1. In fact, it is basically the same construction ρ as in Proposition 2.16, with added relabelling of original vertices. Step (2) to (4) then can be done by traversing the graph from step (1) breadth-first and introducing new edges in at most $\mathcal{O}(|L|(|V| + |E|))$.

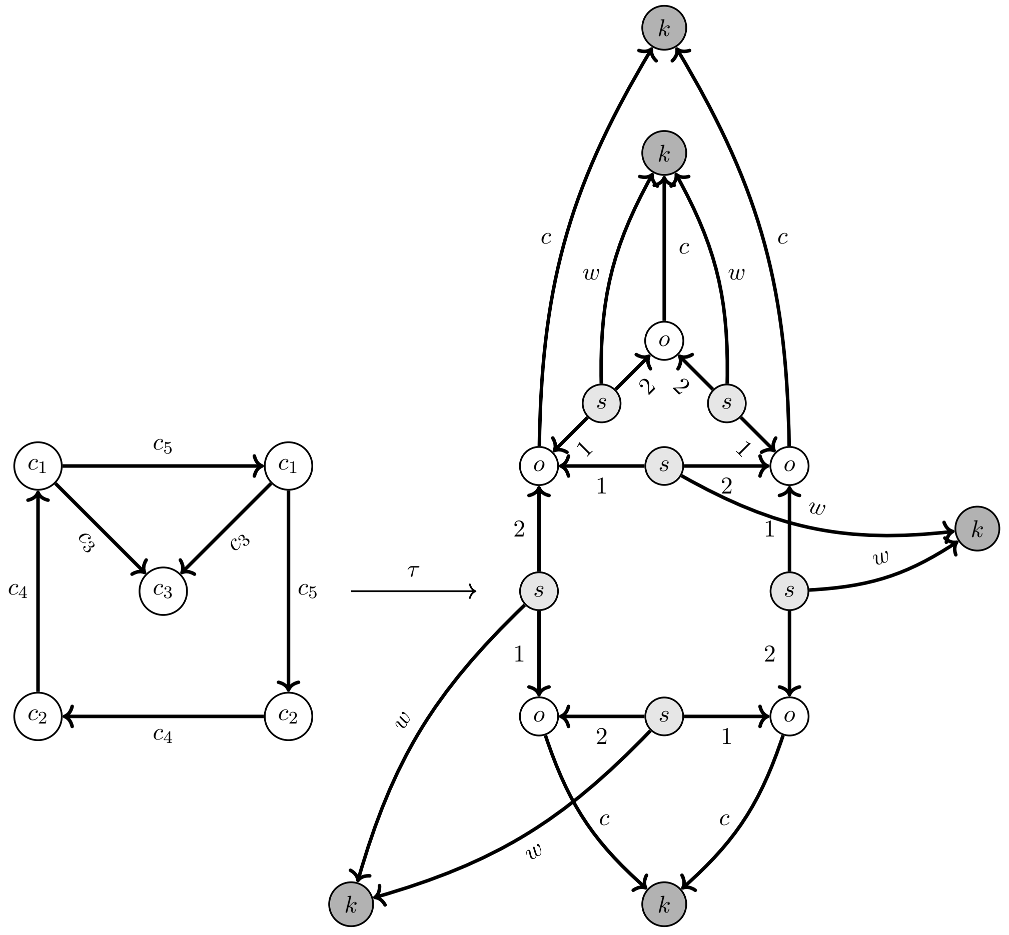


Figure 2.10: Example of encoding τ in the proof of Proposition 2.18 (new vertices are shaded)

We claim that two labelled directed graphs G_1 and G_2 are isomorphic if and only if $\tau(G_1)$ and $\tau(G_2)$ are strongly isomorphic. The first direction is rather trivial. One easily checks that $\phi' : V_1' \to V_2'$ with $\phi'|_{V_1} = \phi$, $\phi'(v_e) = v_{e'}$, $\phi'(v_l) = v_{\psi(l)}$ for each edge $e = (v_1, v_2, l) \in E_1$, $e' = (\phi(v_1), \phi(v_2), \psi(l)) \in E_2$ and label $l \in L_1$ is a strong isomorphism between LDGs $\tau(G_1)$ and $\tau(G_2)$, where $\phi : V_1 \to V_2$ is a given isomorphism between G_1 and G_2 w.r.t. $\psi : L_1 \to L_2$. This is evident by inspection of construction (2.3).
On the other hand, suppose $\phi' : V_1' \to V_2'$ is a given isomorphism between $\tau(G_1)$ and $\tau(G_2)$. Then $\phi'(\mathrm{lab}_1'^{-1}(\{d\})) = \mathrm{lab}_2'^{-1}(\{d\})$ for all $d \in \{o, s, k\}$ due to label-invariance of ϕ' and thus $\phi'(V_1) = V_2$, $\phi'(V_1^{(i)}) = V_2^{(i)}$, $i = 1, 2$, since each set uniquely corresponds to one of such vertex-labels. Hence $\phi = \phi'|_{V_1} : V_1 \to V_2$ and $\psi : L_1 \to L_2$, such that $\phi'(v_l) = v_{\psi(l)}$, are well-defined bijections. Recall proof of Proposition 2.16. The restriction $\phi'|_{V_1 \cup V_1^{(1)}} : V_1 \cup V_1^{(1)} \to V_2 \cup V_2^{(1)}$ declares a strong isomorphism between OOLDG-subgraphs $(V_i \cup V_i^{(1)}, E_i^{(1)}, \{1, 2, o, s\}, \mathrm{lab}_i'|_{V_i \cup V_i^{(1)}})$, $i = 1, 2$, i.e. the subdivision

graphs from construction step (1), which is equivalent to the fact that G_1 and G_2 are isomorphic as unlabelled directed graphs. Since we do not allow multi-edges, even if they are differently labelled, it is only left to check that $\psi\big(\,\mathrm{lab}_1(v)\big) = \mathrm{lab}_2\,\big(\phi(v)\big)$ for each vertex $v \in V_1$ and $\psi(l) = l'$, if $(v_1, v_2, l) \in E_1$ and $\big(\phi(v_1), \phi(v_2), l'\big) \in E_2$.
For fix $v \in V_1$, we find an unique c-labelled edge $(v, v_l, c) \in E_1'$ such that $\big(\phi'(v), v_{\psi(l)}, c\big) = \big(\phi'(v), \phi'(v_l), c\big) \in E_2'$ due to definition of ψ and construction (2.3), which is exactly equivalent to $\psi\big(\,\mathrm{lab}_1(v)\big) = \mathrm{lab}_2\,\big(\phi(v)\big)$. Similar, for edges $e = (v_1, v_2, l) \in E_1$ and $e' = \big(\phi(v_1), \phi(v_2), l'\big) \in E_2$, there exist unique, corresponding w-labelled edges $(v_e, v_l, w) \in E_1'$ and $(v_{e'}, v_{l'}, w) = (v_{e'}, \phi(v_l), w) \in E_2'$, equivalent to $l' = \psi(l)$. ■

3 Structural Isomorphism of Context-Free and Regular Grammars

3.1 Grammar Isomorphism and Isomorphic Strict Interpretations

An *alphabet* Σ is a finite set of symbols. A *word* ω over Σ is then a sequence $a_1 \dots a_n$ of finite length, consisting of symbols a_i in Σ, where the *empty word* (sequence of length zero) is denoted by ε. Moreover, denote with Σ^* the set of all words over Σ and let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For two alphabets Σ and Δ let $\Sigma\Delta = \{ab : a \in \Sigma, b \in \Delta\}$.

Definition 3.1 (Formal Grammar). A *formal grammar* is a four tuple $G = (N, \Sigma, P, S)$, where N and Σ are disjoint finite sets of *nonterminals* and *terminals* respectively, $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^+$ is a finite set of *productions* or *rules*, and $S \in N$ is the *start symbol*. We write $\gamma \to \omega$ instead of $(\gamma, \omega) \in P$. Note that we do not allow the right side of a production to be empty.

If not stated otherwise, lower-case Greek letters like γ, ω refer to words, upper-case Latin letters like A, B, S, T to nonterminals and lower-case Latin letters like a, b, c to terminals.

Definition 3.2 (Context-Free/Regular Grammar). A *context-free grammar* is a formal grammar $G = (N, \Sigma, P, S)$, where every production in P is of the form $A \to \gamma$ for some nonterminal $A \in N$ and $\gamma \in (N \cup \Sigma)^+$, or equivalently $P \subseteq N \times (N \cup \Sigma)^+$. If even $P \subseteq N \times (\Sigma \cup \Sigma N)$, we call G *regular*.

Example 3.3. Let $N = \{S, A, B\}$ and $\Sigma = \{a, b, c\}$. Then $G_1 = (N, \Sigma, P_1, S)$ and $G_2 = (N, \Sigma, P_2, S)$ with production sets

$$P_1 = \{S \to aA, A \to bB, B \to b, A \to c\} \quad \text{and} \quad P_2 = \{S \to aAB, A \to bB, A \to c\}$$

are both valid formal grammars. Moreover, both are context-free while G_1 is even regular (G_2 not because of production $S \to aAB$).

In this thesis we are only worried about structural isomorphisms of context-free formal grammars. That is to say, we only care about syntax. To motivate the following definitions however, we first analyse semantic properties. Without going into much detail, we can generate words using a formal grammar by gradually replacing nonterminals according to productions, starting with symbol S. Recall $G = G_1$ from the example above.

We start at S and use production $S \to aA$ to generate aA. Next, replace A with, for example, bB using $A \to bB$ to generate abB. Lastly, replace B by b using $B \to b$ and obtain $abb \in \Sigma^*$. Since they are no more nonterminals left, we are done. We call the subset $\mathcal{L}(G) \subseteq \Sigma^*$ of in this way obtained words the *(formal) language generated by* G. For a rigorous definition of formal languages refer to [Sch08].
Now consider $\mathcal{L} = ab^* \subseteq \{a,b\}^* = \Sigma^*$, i.e. the language consisting of words starting with a followed by arbitrary many b's. Language $\mathcal{L}$ can be generated by regular grammars $G = (\{S\}, \Sigma, \{S \to a, S \to Sb\}, S)$ or $G' = (\{T\}, \Sigma, \{T \to a, T \to Tb\}, T)$, i.e. $\mathcal{L}(H) = \mathcal{L} = \mathcal{L}(G)$. Note that G and G' do have the same formal structure in the sense that production sets are the same up to rewriting of nonterminals, induced by an (trivial) one-to-one correspondence between the nonterminal sets. We want to call G and G' *isomorphic grammars*. So what happens if we rewrite terminals as well? Consider formal language $\mathcal{L}' = 01^* \subseteq \{0,1\}^* = \Delta^*$. We say that $\mathcal{L}'$ is an *isomorphic strict interpretation* of $\mathcal{L}$ in the sense that the languages have the same basic structure, i.e. words start with one terminal symbol followed by arbitrary many times an other distinct terminal symbol. This can then be pulled down to the level of formal grammars, where we regard formal grammars to be isomorphic strict interpretations of each other if production sets are the same up to rewriting of nonterminals and terminals respectively. For example, $\mathcal{L}'$ can be generated by structurally similar, formal grammar $H = (\{T\}, \Delta, \{T \to 0, T \to T1\}, T)$. Here, just rewrite $S \leftrightarrow T$, $a \leftrightarrow 0$ and $b \leftrightarrow 1$.

Definition 3.4 (Grammar Isomorphism)**.** Let $G = (N, \Sigma, P, S)$ and $H = (M, \Sigma, Q, T)$ be context-free grammars. A *(grammar) isomorphism* is a bijection $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$ such that ϕ is the identity on Σ, $\phi(S) = T$ and

$$\phi(P) = \{\phi(A) \to \phi(\gamma) : A \to \gamma \in P\} = Q,$$

where we consider the natural extension of ϕ to all of $(N \cup \Sigma)^+$ via $\phi(a_1 \dots a_n) = \phi(a_1) \dots \phi(a_n)$. We say G and H are *isomorphic (context-free) grammars.*

Definition 3.5 (Isomorphic Strict Interpretation)**.** Let $G = (N, \Sigma, P, S)$ and $H = (M, \Delta, Q, T)$ be context-free grammars. An *isomorphic strict interpretation* is a bijection $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$ such that $\phi(\Sigma) = \Delta$, $\phi(S) = T$ and

$$\phi(P) = \{\phi(A) \to \phi(\gamma) : A \to \gamma \in P\} = Q.$$

We call G an *isomorphic strict interpretation* of H. Note that contrary to Definition 3.4, we do not require G and H to posses the same terminal set. Moreover, a grammar isomorphism is a special case of an isomorphic strict interpretation with invariant terminal set.

Property $\phi(S) = T$ is essential and non-trivial. Consider $N = \{S, A\}$, $M = \{T, A\}$, $\Sigma = \{a, b\}$ and production sets

$$P = \{S \to aS, S \to aA, A \to b\}, \quad Q = \{T \to b, A \to aA, A \to aT\},$$

i.e. regular grammars $G = (N, \Sigma, P, S)$ and $H = (M, \Sigma, Q, T)$. Then for bijection

$$\phi = \{S \mapsto A, A \mapsto T\} \sqcup \mathrm{id}_\Sigma,$$

we have $\phi_\Sigma = \mathrm{id}_\Sigma$ and $\phi(P) = \phi(Q)$, but $\mathcal{L}(G) = a^+b \neq \{b\} = \mathcal{L}(H)$.

Grammar Isomorphism (GIso) (N, Σ, P, S) isomorphic to (M, Σ, Q, T)	$\exists \phi : (N \cup \Sigma) \to (M \cup \Sigma)$ bijective such that (1) $\phi(N) = M$ (2) $\phi\|_\Sigma = \mathrm{id}_\Sigma$ (3) $\phi(S) = T$ (4) $\phi(P) = Q$
Isomorphic Strict Interpretation (ISI) (N, Σ, P, S) isomorphic strict interpretation of (M, Δ, Q, T)	$\exists \phi : (N \cup \Sigma) \to (M \cup \Delta)$ bijective such that (1) $\phi(N) = M$ (2) $\phi(\Sigma) = \Delta$ (3) $\phi(S) = T$ (4) $\phi(P) = Q$

Table 3.1: Comparison of grammar isomorphism and isomorphic strict interpretation

3.2 Template Construction

To proof **GI**-completeness of determining whether two given context-free/regular grammars are isomorphic or isomorphic strict interpretations of each other, we first need a way to analyse the general structure of production rules $A \to \gamma$, i.e. the relationship of nonterminals to terminals.

Definition 3.6 (Templates). Let $G = (N, \Sigma, P, S)$ be a context-free grammar. The *template* of a production $A \to \gamma$ in P is a word over $\Sigma \cup \{L_1, \ldots, L_{|N|}\}$, L_j's fresh symbols different from Σ, obtained from $A\gamma$ by replacing each occurrence of a nonterminal with an L_j, where L_j refers to the jth distinct nonterminal in $A\gamma$. Furthermore, denote with $\mathrm{Temp}(G)$ the lexicographically ordered set of all such templates of productions in P. We allow ourselves to call L_j's nonterminals when talking about templates (after all they can be easily differentiated from initial terminal set Σ).

Following Example 3.3, the template of production $S \to aAB$ is $L_1aL_2L_3$. Now we can use templates to reduce context-free grammars to regular grammars, exploiting order and frequency of nonterminals in relation to terminals.

Definition 3.7 (Regularisation of Context-Free Grammars). Let $G = (N, \Sigma, P, S)$ be a context-free grammar and $\mathrm{Temp}(G) = \{T_1, \dots, T_t\}$ its corresponding ordered set of templates defined as in Definition 3.6. Denote, for fix $1 \le i \le t$, with n_i the number of distinct nonterminals in T_i, that is to say the highest number k such that L_k occurs in T_i, and with p_i the number of distinct productions with template T_i. Furthermore, assume the productions themselves to be ordered in such a way that we can unambiguously refer to the jth production with template T_i. Now construct a regular grammar $\mathrm{Reg}(G) = (N', \Sigma', P', S)$ the following way: $N' = N \cup \{A_{ij} : 1 \le i \le t, 1 \le j \le p_i\}$, $\Sigma' = \{a\} \cup \{a_{ik} : 1 \le i \le t, 1 \le k \le n_i\}$ and $P' = P_1 \cup P_2 \cup P_3$, with

$$\begin{aligned} P_1 &= \{S \to aA_{ij} : 1 \le i \le t, 1 \le j \le p_i\}, \\ P_2 &= \{A_{ij} \to a_{ik}X_{ijk} : 1 \le i \le t, 1 \le j \le p_i, 1 \le k \le n_i\}, \\ P_3 &= \{X \to a : X \in N\}, \end{aligned} \tag{3.1}$$

where a, a_{ik}, A_{ij} are fresh symbols and X_{ijk} refers to the kth distinct nonterminal in the jth production with template T_i. We call $\mathrm{Reg}(G)$ the *regularisation* of G.

Since $|N'| = |N| + |P|$, $|\Sigma'| \le 1 + |P||N|$ and $|P'| = |P_1| + |P_2| + |P_3| \le |P| + |N| + |P||N|$, Reg is at worst of size $\mathcal{O}(|P| + |N| + |P||N|) = \mathcal{O}(|P||N|)$. Template set $\mathrm{Temp}(G)$ and X_{ijk}-references can be computed in $\mathcal{O}(s|P||N| \log(|N||\Sigma||P|))$, where s is an upper bound on the length of productions in P, i.e. $s = \max\{|A\gamma| : A \to \gamma \in P\}$. Refer to Algorithm 3.1. The production set P_1 is then computed in $\mathcal{O}(|P|)$, P_2 in $\mathcal{O}(|P||N|)$ and P_3 in $\mathcal{O}(|N|)$, accumulating to a total time complexity of at worst $\mathcal{O}(s|P||N| \log(|N||\Sigma||P|))$ for the computation of all $\mathrm{Reg}(G)$.

Example 3.8. Consider the context-free grammar $G = (N, \Sigma, P, S)$, where $N = \{S, A, B\}$, $\Sigma = \{b, c, d\}$ and $P = \{S \to dSAB, A \to bB, B \to bA, B \to c\}$. First, we calculate template set $\mathrm{Temp}(G) = \{T_1, T_2, T_3\}$:

i	**Production**	**Template** T_i	p_i	n_i
1	$S \to dSAB$	$L_1dL_1L_2L_3$	1	3
2	$A \to bB$ $B \to bA$	L_1bL_2	2	2
3	$B \to c$	L_1c	1	1

and substitution X_{ijk}:

$$\begin{array}{cccc} X_{1jk} & 1 & 2 & 3 \\ 1 & S & A & B \end{array} \qquad \begin{array}{ccc} X_{2jk} & 1 & 2 \\ 1 & A & B \\ 2 & B & A \end{array} \qquad \begin{array}{cc} X_{3jk} & 1 \\ 1 & B \end{array}$$

The regularisation $\mathrm{Reg}(G) = (N', \Sigma', P', S)$ of G is then given by

$$\begin{aligned}
&N' = N \cup \{A_{ij} : 1 \leq i \leq t, 1 \leq j \leq p_i\} = \{A, B, C, A_{11}, A_{21}, A_{22}, A_{31}\},\\
&\Sigma' = \{a\} \cup \{a_{ik} : 1 \leq i \leq t, 1 \leq k \leq n_i\} = \{a, b_{11}, b_{12}, b_{13}, b_{21}, b_{22}, b_{31}\},\\
&P' = P_1 \cup P_2 \cup P_3,\\
&P_1 = \{S \to aA_{11}, S \to aA_{21}, S \to aA_{22}, S \to aA_{31}\},\\
&P_2 = \{A_{11} \to b_{11}S, A_{11} \to b_{12}A, A_{11} \to b_{13}B, A_{21} \to b_{21}A,\\
&\qquad A_{21} \to b_{22}B, A_{22} \to b_{21}B, A_{22} \to b_{22}A, A_{31} \to b_{31}B\},\\
&P_3 = \{X \to a : X \in N\} = \{S \to a, A \to a, B \to a\}.
\end{aligned}$$

3.3 GI-Completeness of Structural Isomorphisms between Context-Free and Regular Grammars

The goal of this section is to prove the following theorem by Rosenkrantz & Hunt[RH85]:

Theorem 3.9 (Rosenkrantz & Hunt, 1984). *The following sets are polynomially equivalent:*

(i) $\mathbf{CFGI} = \{(G, H) : G, H \text{ isomorphic context-free grammars}\}$

(ii) $\mathbf{RGI} = \{(G, H) : G, H \text{ isomorphic regular grammars}\}$

(iii) $\mathbf{GI} = \{(G_1, G_2) : G_1, G_2 \text{ isomorphic graphs}\}$

(iv) $\mathbf{RGISI} = \{(G, H) : G, H \text{ regular grammars and } G \text{ strict interpretation of } H\}$

(v) $\mathbf{CFGISI} = \{(G, H) : G, H \text{ context-free grammars and } G \text{ iso. strict int. of } H\}$

Proof. We proceed with the same approach as in [RH85]. The claim then immediately follows by chain of arguments as depicted in Figure 3.1. ■

Proposition 3.10 ($(i) \Longrightarrow (ii)$). $\mathbf{CFGI} \preceq^{\mathbf{P}} \mathbf{RGI}$.

Proof. Let $G = (N, \Sigma, P, S)$ and $H = (M, \Delta, Q, T)$ be context-free grammars. We claim that G are H isomorphic context-free grammars if and only if their regularisations $\mathrm{Reg}(G)$ and $\mathrm{Reg}(H)$ are isomorphic, and they have the same set of templates, $\mathrm{Temp}(G) = \mathrm{Temp}(H)$. Let the regularisation $\mathrm{Reg}(G) = (N', \Sigma', P', S)$ of G be defined as in Definition 3.7. Moreover, if $\mathrm{Temp}(H) = \{T'_1, \ldots, T'_{t'}\}$, denote with m_i the number of distinct nonterminals in T'_i and with q_i the number of productions with template T'_i. Then the regularisation $\mathrm{Reg}(H) = (M', \Delta', Q', T)$ of H is given by $M' = M \cup \{A_{ij} : 1 \leq i \leq t', 1 \leq j \leq q_i\}$, $\Delta' = \{a\} \cup \{a_{ik} : 1 \leq i \leq t', 1 \leq k \leq m_i\}$ and $Q' = Q_1 \cup Q_2 \cup Q_3$, with

$$\begin{aligned}
&Q_1 = \{S \to aA_{ij} : 1 \leq i \leq t', 1 \leq j \leq q_i\},\\
&Q_2 = \{A_{ij} \to a_{ik}Y_{ijk} : 1 \leq i \leq t', 1 \leq j \leq q_i, 1 \leq k \leq m_i\}, \qquad (3.2)\\
&Q_3 = \{Y \to a : Y \in M\},
\end{aligned}$$

$$\begin{tikzcd}
(i)\ \textbf{CFGI} \arrow[rr, Rightarrow, "\text{Proposition 3.10}"] & & (ii)\ \textbf{RGI} \arrow[dl, Rightarrow, "\text{Proposition 3.12}"'] \\
& (iii)\ \textbf{GI} \arrow[r, Leftrightarrow, "\text{Theorem 2.10}"] & \textbf{DG} \arrow[d, Rightarrow, "\text{Proposition 3.13}"'] \\
(v)\ \textbf{CFGISI} \arrow[uu, Rightarrow, "\text{Proposition 3.14}"] & & (iv)\ \textbf{RGISI} \arrow[ll, Rightarrow, "\text{Trivial reduction by subset property}"]
\end{tikzcd}$$

Figure 3.1: Structure of the proof of Theorem 3.9

where a, a_{ik}, A_{ij} are fresh symbols and Y_{ijk} refers to the kth distinct nonterminal in the jth production with ith template in Q, recall Definition 3.7.

Suppose G and H are isomorphic. Then $\Sigma = \Delta$ and we find an isomorphism $\phi : (N \cup \Sigma) \to (M \cup \Delta)$. The template construction in Definition 3.6 only considers the order of nonterminals and their relation to terminals, which is invariant under isomorphisms, i.e. $A \to \gamma$ and $\phi(A) \to \phi(\gamma)$ have the exact same template, and thus $\mathrm{Temp}(G) = \mathrm{Temp}(H) = \{T_1, \dots, T_t\}$ in the same lexicographical order, since $\phi(P) = Q$. This in turn means $n_i = m_i$ for $1 \leq i \leq t$ and $\Sigma' = \Delta'$. Let $\mathrm{Temp}_i(G)$ and $\mathrm{Temp}_i(H)$ be the ordered set of productions with template T_i in P and Q respectively. Fix $1 \leq i \leq t$. Then $|\mathrm{Temp}_i(G)| = |\phi(\mathrm{Temp}_i(G))| = |\mathrm{Temp}_i(H)|$ and thus $p_i = q_i$. Moreover, let $\sigma_i \in \mathfrak{S}_{p_i}$ be a permutation of the set $\{1, \dots, p_i\}$ so that the image of the jth production in $\mathrm{Temp}_i(G)$ under ϕ corresponds to the $\sigma_i(j)$th production in $\mathrm{Temp}_i(H)$. More formally, if $\mathrm{Temp}_i(G) = \{B_{i1} \to \gamma_{i1} \to B_{ip_i} \to \gamma_{ip_i}\}$ and $\mathrm{Temp}_i(H) = \{C_{i1} \to \omega_{i1} \to C_{ip_i} \to \omega_{ip_i}\}$, then

$$\forall_{1 \leq j \leq p_i} \ \phi(B_{ij}) \to \phi(\gamma_{ij}) = C_{i\sigma_i(j)} \to \omega_{i\sigma_i(j)}. \tag{3.3}$$

Now let $\phi' : (N' \cup \Sigma') \to (M' \cup \Delta')$ be defined as follows:

$$\phi'(\alpha) = \begin{cases} \alpha & \text{for all } \alpha \in \Sigma', \\ \phi(\alpha) & \text{for all } \alpha \in N, \\ A_{i\sigma_i(j)} & \text{if } \alpha = A_{ij} \in N' \setminus N. \end{cases}$$

We claim that ϕ' declares an isomorphism between $\mathrm{Reg}(G)$ and $\mathrm{Reg}(H)$. Recall that $p_i = q_i$ and $n_i = m_i$ for all $1 \leq i \leq t$. Then by (3.3), ϕ' is well-defined, bijective, $\phi'|_{\Sigma'} = \mathrm{id}_{\Sigma'}$, $\phi'(N') = M'$ and $\phi'(S) = \phi(S) = T$. Moreover, $|P'| = |Q'|$ and thus it suffices to prove that $\phi'(P') \subseteq Q'$. Inspection of productions (3.1) and (3.2) immediately shows that $\phi'(P_1) \subseteq Q_1 \subseteq Q'$ and $\phi'(P_2) \subseteq Q_2 \subseteq Q'$. Now, consider $A_{ij} \to a_{ik}X_{ijk} \in P_3 \subseteq P'$ with $\phi'(A_{ij} \to a_{ik}X_{ijk}) = A_{i\sigma_i(j)} \to a_{ik}\phi(X_{ijk})$. The nonterminal $\phi(X_{ijk})$ is exactly the image of the kth distinct nonterminal in the jth production with template T_i under ϕ, which by (3.3) is $Y_{i\sigma_i(j)k}$, i.e.

$$\phi'(A_{ij} \to a_{ik}X_{ijk}) = A_{i\sigma_i(j)} \to a_{ik}\phi(X_{ijk}) = A_{i\sigma_i(j)} \to a_{ik}Y_{i\sigma_i(j)k} \in Q_3 \subseteq Q'.$$

For the reverse implication, suppose Reg(G) and Reg(H) are isomorphic via $\phi' : (N' \cup \Sigma') \to (M' \cup \Delta')$, and template sets Temp$(G)$ = Temp(H) = $\{T_1, \ldots, T_t\}$ are ordered the same way. Then $\Sigma = \Delta$ by simply grouping all terminals under the assumption that both terminal sets are minimal, or in other words, there are no terminals which do not appear in any production in P or Q respectively. Our first goal is to show that $p_i = q_i$ and $n_i = m_i$ for all $1 \leq i \leq t$. The latter is true by the initial assumption that G and H share the same, ordered template set, i.e. the number of distinct nonterminals in the ith template of P is the same as the number of distinct nonterminals in the ith template of Q. Note, that by constructions (3.1) and (3.2), $\phi'(P_l) = Q_l$ for $l = 1, 2, 3$, since $\phi'(S) = T$ and $\phi'|_{\Sigma'} = \mathrm{id}_{\Sigma'}$. Then, for fix $1 \leq i \leq t$ and $1 \leq j \leq p_i$, $A_{ij} \in N'$ and $\phi'(A_{ij}) = A_{i'j'} \in M'$ due to $T \to aA_{i'j'} = \phi'(S \to aA_{ij}) \in Q_1$, for some $1 \leq i' \leq t$ and $1 \leq j' \leq q_{i'}$. We claim that $i = i'$. Indeed, $\phi(A_{ij} \to a_{i1}X_{ij1}) = A_{i'j'} \to a_{i1}\phi(X_{ij1}) \in Q'$, so necessarily $i = i'$. Moreover, $\phi'(N) = M$ and $\phi'(N' \setminus N) = M' \setminus M$, and therefore $\phi'(\{A_{i,1}, \ldots, A_{i,p_i}\}) = \{A_{i,1}, \ldots, A_{i,q_i}\}$ describes an one-to-one correspondence between the two nonterminal subsets. Then not only $p_i = q_i$, but we also find a permutation $\sigma_i \in \mathfrak{S}_{p_i}$, such that $\sigma_i(j) = j'$, i.e. $\phi'(A_{ij}) = A_{ij'} = A_{i\sigma_i(j)}$, accounting for the not necessarily correspondingly ordered production sets P and Q. Let map $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$ be given by $\phi|_\Sigma = \mathrm{id}_\Sigma$ and $\phi|_N = \phi'|_N$, or equivalently $\phi = \mathrm{id}_\Sigma \sqcup \phi'|_N$,

$$\phi(\alpha) = \begin{cases} \alpha & \text{for all } \alpha \in \Sigma, \\ \phi'(\alpha) & \text{for all } \alpha \in N. \end{cases}$$

Then ϕ is by assumption clearly well-defined and bijective with $\phi(S) = \phi'(S) = T$. Since $p_i = q_i$ for all $1 \leq i \leq t$, $|P| = \sum_{i=1}^t p_i = \sum_{i=1}^t q_i = |Q|$ and again it suffices to show $\phi(P) \subseteq Q$. For that, fix arbitrary $A \to \gamma \in P$ and choose $1 \leq i \leq t$, $1 \leq j \leq p_i$ such that $A \to \gamma$ is exactly the jth production in P with template T_i. By construction (3.1), this production corresponds to the set $\{A_{ij} \to a_{i1}X_{ij1}, \ldots, A_{ij} \to a_{in_i}X_{ijn_i}\}$ of productions in P'. Then Q' has the productions

$$\begin{aligned} &\phi'\big(\{A_{ij} \to a_{i1}X_{ij1}, \ldots, A_{ij} \to a_{in_i}X_{ijn_i}\}\big) \\ =&\big\{\phi'(A_{ij}) \to a_{i1}\phi(X_{ij1}), \ldots, \phi'(A_{ij}) \to a_{in_i}\phi(X_{ijn_i})\big\} \\ =&\big\{A_{i\sigma_i(j)} \to a_{i1}\phi(X_{ij1}), \ldots, A_{i\sigma_i(j)} \to a_{in_i}\phi(X_{ijn_i})\big\}, \end{aligned}$$

which in turn unambiguously correspond to the $\sigma_i(j)$th production in Q with template T_i and thus $\phi(A \to \gamma) \in Q$, since $n_i = m_i$ and their being exactly n_i-many distinct productions of the form $A_{i\sigma_i(j)} \to \gamma \in Q'$ by construction (3.2). ∎

Next we discuss an encoding of a regular grammar into an unlabelled undirected graph.

Definition 3.11 (Graph Encoding of Regular Grammars). Let $G = (N, \Sigma, P, S)$ be a regular grammar with ordered terminal set $\Sigma = \{a_1, \dots, a_r\}$. Then G can be encoded as unlabelled undirected graph $\mathrm{Graph}(G) = (V, E)$, with vertices $V = V_1 \cup V_2 \cup V_3 \cup V_4$ and edges $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6 \cup E_7$, where

$$\begin{aligned}
V_1 &= \{[A, i] : A \in N, 1 \leq i \leq 4\},\\
V_2 &= \{[A \to a_k B, i] : A \to a_k B \in P, 1 \leq i \leq k+1\},\\
V_3 &= \{[A \to a_k, i] : A \to a_k \in P, 1 \leq i \leq k+1\},\\
V_4 &= \{[S]\},\\
E_1 &= \{\langle [A, i], [A, j] \rangle : A \in N, 1 \leq i, j \leq 4, i \neq j\},\\
E_2 &= \{\langle [A \to a_k B, i], [A \to a_k B, i+1] \rangle : A \to a_k B \in P, 1 \leq i \leq k\},\\
E_3 &= \{\langle [A \to a_k, i], [A \to a_k, i+1] \rangle : A \to a_k \in P, 1 \leq i \leq k\},\\
E_4 &= \{\langle [A, i], [A \to a_k B, 1] \rangle : A \to a_k B \in P, 1 \leq i \leq 2\},\\
E_5 &= \{\langle [A \to a_k B, k+1], [B, 4] \rangle : A \to a_k B \in P\},\\
E_6 &= \{\langle [A, i], [A \to a_k, 1] \rangle : A \to a_k \in P, 1 \leq i \leq 2\},\\
E_7 &= \{\langle [S], [S, 4] \rangle\}.
\end{aligned} \tag{3.4}$$

Note that $|V| \leq 4|N| + 2|P|(|\Sigma| + 1) + 1$ and $|E| \leq 6|N| + |P|(2|\Sigma| + 5) + 1$. Moreover, $|P| \leq |\Sigma||N|^2 + |\Sigma||N|$, since at worst $P = (N \times \Sigma N) \cup (N \times \Sigma)$. Therefore $\mathrm{Graph}(G)$ is of size and can be constructed in $\mathcal{O}(|N| + |\Sigma||P|) = \mathcal{O}(|\Sigma|^2|N|^2)$ if we assume terminal set and productions to be suitably ordered and accessible. Refer to Algorithm 3.2 for a more detailed description and Figure 3.2 for an exemplary encoding $\mathrm{Graph}(G)$ of regular grammar $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, $P = \{S \to aA, A \to bB, A \to c\}$.
The basic idea is to encode every nonterminal as unique 4-clique with V_1, E_1 and use V_4, E_7 to highlight the starting symbol 4-clique. Now productions of form $A \to a_k B$ are encoded as $k+2$-paths between corresponding 4-cliques by V_2, E_2, E_4, E_5, while productions $A \to a_k$ are encoded as $k+1$-paths starting in A's 4-clique and ending in an unique leaf by V_3, E_3, E_6.

Proposition 3.12 ((ii) $\Longrightarrow$ (iii)). **RGI $\preceq^{\mathbf{P}}$ GI**.

Proof. Let $G = (N, \Sigma, P, S)$ and $H = (M, \Sigma, Q, T)$ be regular grammars with ordered terminal set $\Sigma = \{a_1, \dots, a_r\}$. We claim that G and H are isomorphic if and only if $\mathrm{Graph}(G)$ and $\mathrm{Graph}(H)$ are isomorphic. To this end, consider $\mathrm{Graph}(H) = (V', E')$ to be defined analogously as in Definition 3.11.
Suppose G and H to be isomorphic w.r.t. $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$. Then $\phi' : V \to V'$, $\phi'([S]) = [T]$, $\phi'([A, i]) = [\phi(A), i]$, $\phi'([A \to a_k B, j]) = [\phi(A) \to a_k \phi(B), j]$ and $\phi'([A \to a_k, j]) = [\phi(A) \to a_k), j]$ for $1 \leq i \leq 4$, $1 \leq j \leq k$ and $A \to a_k B, A \to a_k \in P$, is a well-defined graph isomorphism. This is immediately clear by inspection of equations (3.4) in Definition 3.11, since one-to-one correspondences $\phi(P) = Q$ and $\phi(N) = M$ induce an one-to-one correspondence between edges in E and E'.

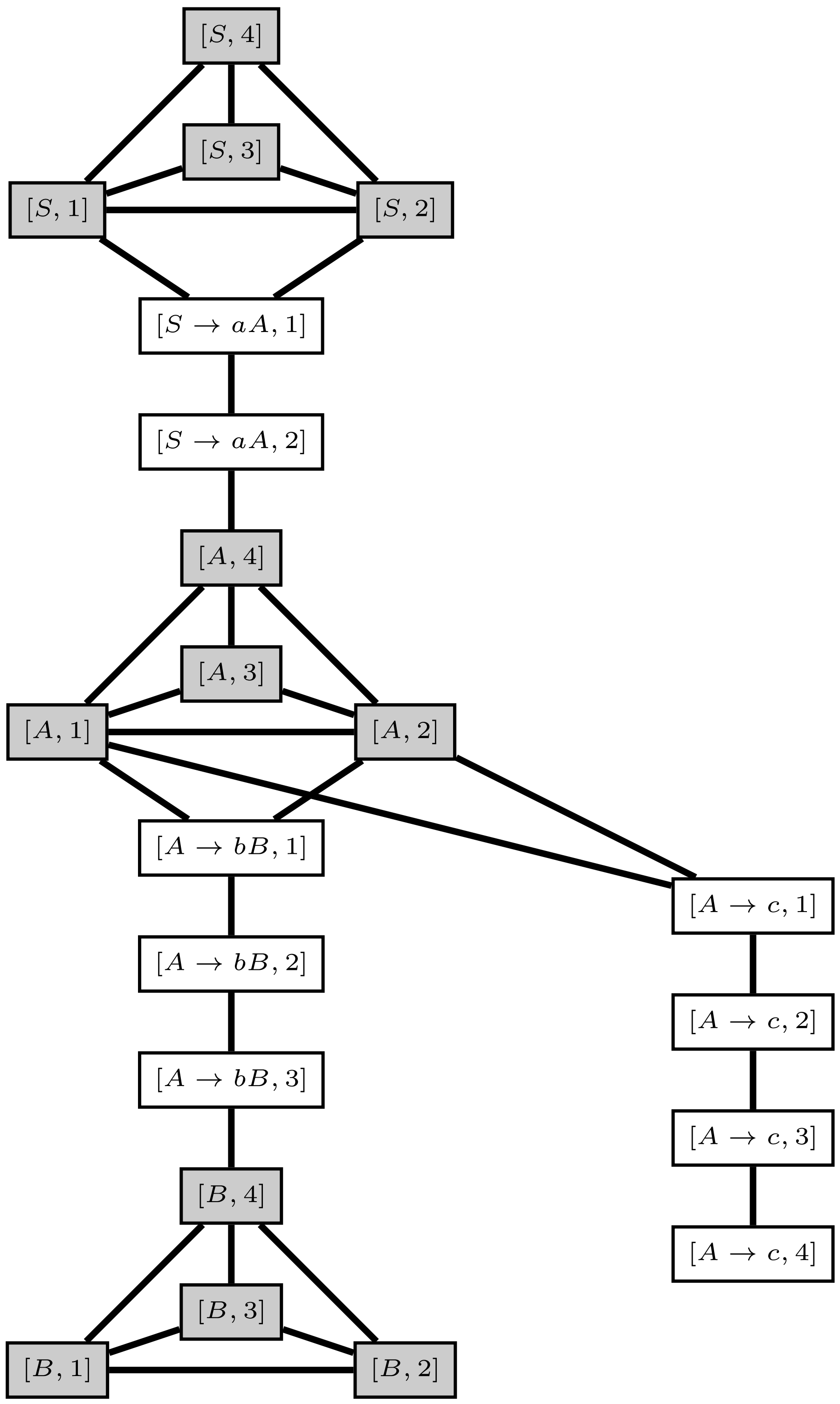
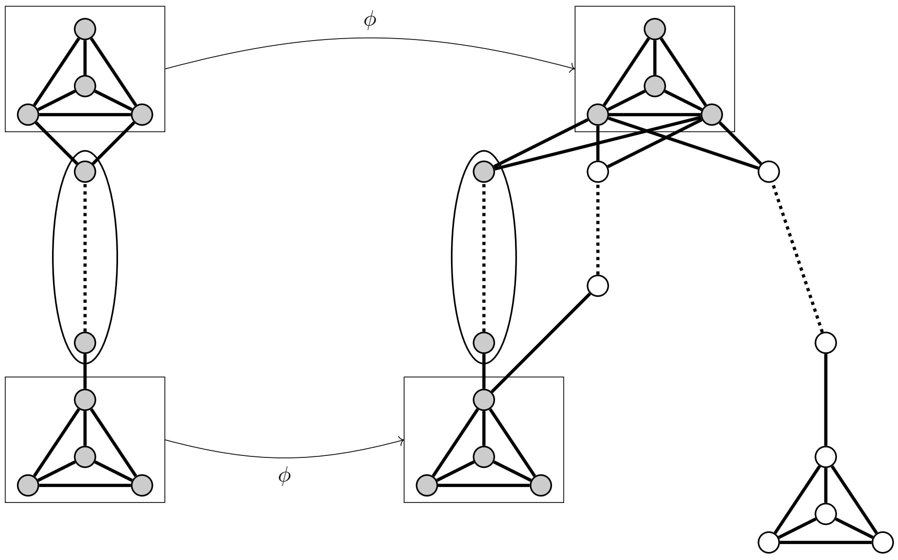
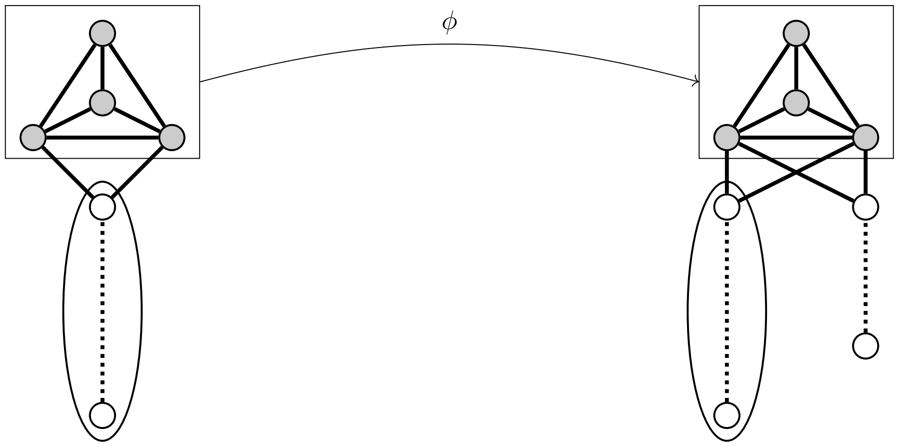


Figure 3.2: Example of encoding Graph in Definition 3.11

Regular grammar $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, $P = \{S \to aA, A \to bB, A \to c\}$ with ordered terminal set $\{a, b, c\} = \{a_1, a_2, a_3\}$ (4-cliques are shaded)



(a) Matching $k+2$-paths between 4-cliques



(b) Matching $k+1$-paths between 4-cliques and leafs

Figure 3.3: Matching isomorphically unique paths in proof of Proposition 3.12

On the other hand, suppose $\phi' : V \to V'$ is a graph isomorphism. We first make a couple of observations that will lead us to define an unique isomorphism between the underlying regular grammars. For $A \in N$, denote with Cl_A the 4-clique in $\mathrm{Graph}(G)$ consisting exactly of vertices $[A,1],[A,2],[A,3],[A,4]$, connected by edges E_1.

(a) $[S]$ and $[A \to a_k, k+1]$, for some production $A \to a_k \in P$, are the only leafs, i.e. $\{v \in V : \deg(v) = 1\} = \{[S]\} \cup \{[A \to a_k, k+1] : A \to a_k \in P\}$.

(b) Let $k \in \mathbb{N}$. Every path of length $k+2$ between two distinct 4-cliques $\mathrm{Cl}_A, \mathrm{Cl}_B$ is exactly of the form

$$\mathrm{Path}_i(A \to a_k B) = \Big([A,i],[A \to a_k B,1],\ldots,[A \to a_k B,k+1],[B,4]\Big) \tag{3.5}$$

for $i = 1, 2$ and $A \to a_k B \in P$. Every path of length $k + 1$ between a 4-clique Cl_A and a leaf has exactly the form

$$\mathrm{Path}_i(A \to a_k B) = \Big([A, i], [A \to a_k, 1], \dots, [A \to a_k, k], [A \to a_k, k+1]\Big) \tag{3.6}$$

for $i = 1, 2$ and $A \to a_k \in P$. In both cases we understand a path to be so that no edge is contained in Cl_A or Cl_B.

(c) Every 4-clique in $\mathrm{Graph}(G)$ is of the form Cl_A for some $A \in N$.

Claims (a) and (b) are immediately clear by equations (3.4). Moreover, (b) implies that $[A, i]$ and $[B, j]$ for distinct $A, B \in N$, $1 \le i, j \le 4$ are not adjacent, and thus can not be part of the same 4-clique. Consider $v \in V \setminus V_1$ of degree ≥ 3. Then $v \in \{[A \to a_k, 1], [A \to a_k B, 1]\}$ for some $A \to a_k, A \to a_k B \in P$. In the first case, $[A \to a_k, 2]$ adjacent with $\deg([A \to a_k, 2]) \in \{1, 2\}$, depending on if $k = 1$, and in the second, $[A \to a_k B, 2]$ adjacent with $\deg([A \to a_k, 2]) = 2$. This immediately implies (c). Now map $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$ with $\phi|_\Sigma = \mathrm{id}_\Sigma$ and $\phi'(\mathrm{Cl}_A) = \mathrm{Cl}_{\phi(A)}$ for all $A \in N$, is well-defined and bijective. Since $[S]$ is the only vertex in V of degree 1 directly connected to a 4-clique, $\phi'([S]) = [T]$ and thus $\phi'(\mathrm{Cl}_S) = \mathrm{Cl}_T$, i.e. $\phi(S) = T$. For $A \to a_k B \in P$ consider $\mathrm{Path}_1(A \to a_k B)$ defined by (3.5). Then $\phi'(\mathrm{Path}_1(A \to a_k B))$ is a path of length $k + 2$ between $\mathrm{Cl}_{\phi(A)}$ and $\mathrm{Cl}_{\phi(B)}$. By (b), $\phi'(\mathrm{Path}_1(A \to a_k B)) = \mathrm{Path}_i\big(\phi(A) \to a_k \phi(B)\big)$ for some $i \in \{1, 2\}$, and hence $\phi(A) \to a_k \phi(B) \in Q$. On the other hand, let $A \to a_k \in P$ and $\mathrm{Path}_1(A \to a_k)$ defined by (3.6). Then $\phi'(\mathrm{Path}_1(A \to a_k))$ is a path of length $k + 1$ between $\mathrm{Cl}_{\phi(A)}$ and a vertex of degree 1 and thus again by (b), $\phi'(\mathrm{Path}_1(A \to a_k)) = \mathrm{Path}_i\big(\phi(A) \to a_k\big)$, $i \in \{1, 2\}$, i.e. $\phi(A) \to a_k \in Q$.
Substituting $Q \leftrightarrow P$, $\phi' \leftrightarrow \phi'^{-1}$ and $\phi \leftrightarrow \phi^{-1}$ yields the reverse inclusion $Q \subseteq \phi(P)$. ■

Proposition 3.13 ($(iii) \implies (iv)$). **GI** $\preceq^{\mathbf{P}}$ **RGISI**.

Proof. Let $G = (V, E)$ be a directed graph. For each edge $(v, w) \in E$ and vertex $v \in V$ introduce productions $A_{vw} \to 0A_v$, $A_{vw} \to 1A_w$, $A_v \to 0S$ and $A_v \to 1S$. More formally, we consider the regular grammar $\mathrm{Gram}(G) = (N, \Sigma, P, S)$, $\Sigma = \{0, 1\}$, $N = N_1 \cup N_2 \cup \{S\}$ and $P = \{S \to 0\} \cup P_1 \cup P_2$, where

$$\begin{aligned} N_1 &= \{A_v : v \in V\}, \\ N_2 &= \{A_{vw} : (v, w) \in E\}, \\ P_1 &= \{A_v \to 0S, A_v \to 1S : v \in V\}, \\ P_2 &= \{A_{vw} \to 0A_v, A_{vw} \to 1A_w : (v, w) \in E\}. \end{aligned} \tag{3.7}$$

Moreover, $|N| = 1 + |N_1| + |N_2| = 1 + |V| + |E|$, $|P| = 1 + |P_1| + |P_2| = 1 + 2|V| + 2|E|$ and Gram can be computed in $\mathcal{O}(|V| + |E|)$ by traversing G breadth-first and gradually adding productions and nonterminals. We claim that two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if and only if $\mathrm{Gram}(G_1)$ is an isomorphic strict interpretation of $\mathrm{Gram}(G_2) = (M, \Sigma, Q, T)$.

First, let $\phi' : V_1 \to V_2$ be a graph isomorphism. Then

$$\begin{aligned} M \setminus \{T\} &= \big\{A_{v'} : v' \in V_2\big\} \cup \big\{A_{v'w'} : (v', w') \in E_2\big\} \\ &= \big\{A_{\phi'(v)} : v \in V_1\big\} \cup \big\{A_{\phi'(v)\phi'(w)} : (v, w) \in E_1\big\} \end{aligned} \tag{3.8}$$

disjoint unions, and thus not only $|N| = |M|$, but also $|V_1| = |V_2|$ and $|E_1| = |E_2|$ by unique structure of productions in equations (3.7) and decomposition in (3.8). Consequently, $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$, $\phi|_\Sigma = \mathrm{id}_\Sigma$, $\phi(S) = T$ and $\phi(A_v) = A_{\phi'(v)}$, or more formally

$$\phi(\alpha) = \begin{cases} \alpha & \text{for all } \alpha \in \Sigma, \\ T & \text{if } \alpha = S, \\ A_{\phi'(v)} & \text{if } \alpha = A_v \in N, \end{cases}$$

is well-defined and bijective. Then $\phi(P) = Q$, since $A_{\phi'(v)} \to 0T, A_{\phi'(v)} \to 1T \in Q$ and $A_{\phi'(v)\phi'(w)} \to 0A_{\phi'(v)}, A_{\phi'(v)\phi'(w)} \to 1A_{\phi'(w)} \in Q$ for every $v \in V_1$ and $(v, w) \in E_1$, which matches with definition of ϕ.
On the other hand, let $\phi : (N \cup \Sigma) \to (M \cup \Sigma)$ be an isomorphic strict interpretation. Productions $\phi(S) \to \phi(0) = T \to \phi(0)$ force ϕ to be the identity on Σ, i.e. $\phi(0) = 0$ and $\phi(1) = 1$. By inspecting equations (3.7), we immediately see that $\phi(N_i) = M_i$, $i = 1, 2$. Indeed, if $v \in V_1$, $\phi(A_v \to 0S) = \phi(A_v) \to 0T \in P_1$ and necessarily $A_v = A_{v'}$ for some $v' \in V_2$. Analogously for $v' \in V_2$ and we conclude $\phi(P_i) = Q_i$, $i = 1, 2$.
Then $\phi' : V_1 \to V_2$ such that $\phi(A_v) = A_{\phi'(v)}$, is well-defined and bijective. For $(v, w) \in E_1$ arbitrary, we have that $\phi(A_{vw} \to 0A_v) = A_{v'w'} \to 0A_{\phi'(v)} \in Q_2$ and $\phi(A_{vw} \to 1A_w) = A_{v'w'} \to 1A_{\phi'(w)} \in Q_2$ for some $(v', w') \in E_2$, which is equivalent to $A_{v'w'} = A_{\phi'(v)\phi'(w)}$ and thus $(\phi'(v), \phi'(w)) \in E_2$ by equation (3.7). This however, is already sufficient by $2|E_1| = |\phi(P_2)| = |Q_2| = 2|E_2|$ (again check equations (3.7) and analysis below). ■

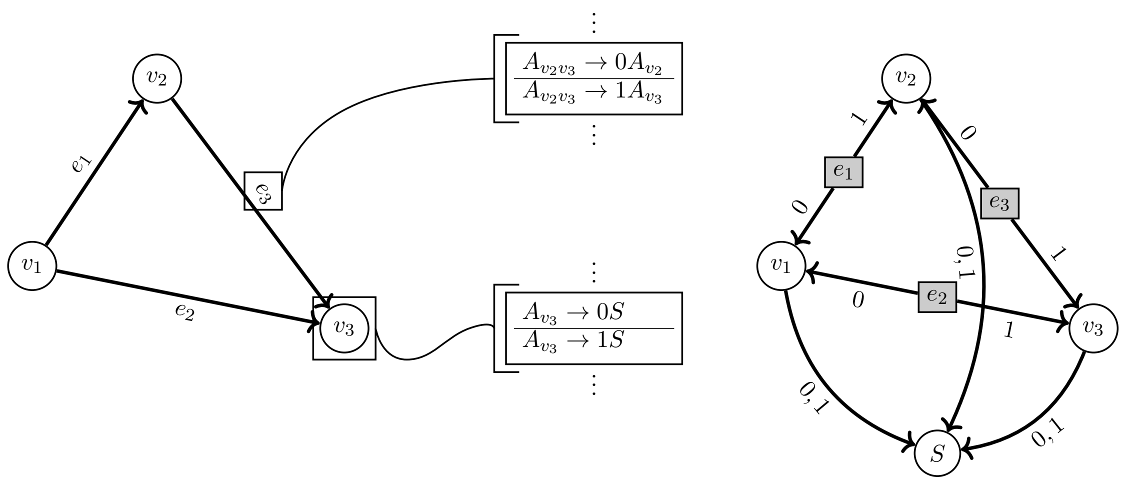


Figure 3.4: Computation of encoding Gram(G) in the proof of Proposition 3.13

Proposition 3.14 ($(v) \Longrightarrow (i)$). **CFGISI** $\preceq^{\mathbf{P}}$ **CFGI**.

Proof. Let $G = (N, \Sigma, P, S)$ and $H = (N, \Delta, Q, T)$ be context-free grammars and consider an encoding ρ into context-free grammars

$$\begin{aligned} \rho(G) &= (N \cup \Sigma, \{\#\}, P \cup \{a \to \# : a \in \Sigma\}, S), \\ \rho(H) &= (M \cup \Delta, \{\#\}, Q \cup \{b \to \# : b \in \Delta\}, T), \end{aligned} \tag{3.9}$$

where $\#$ is a fresh, not yet in $N \cup M \cup \Sigma \cup \Delta$ featured symbol. Note, ρ can, due to introducing $|\Sigma|$-many new productions and possibly updating the set of nonterminals (depending on concrete implementation), at worst be computed in $\mathcal{O}(|\Sigma| + |N| + |P|)$. For example, $G = (\{S, A, B\}, \{a, b, c\}, \{S \to aA, A \to bB, A \to c\}, S)$ is encoded as $\rho(G)$ with nonterminal set $\{S, A, B, a, b, c\}$ and production set extended by $\{a \to \#, b \to \#, c \to \#\}$.
We claim that G is an isomorphic strict interpretation of H if and only if $\rho(G)$ and $\rho(H)$ are isomorphic context-free grammars. Let first $\phi : (N \cup \Sigma) \to (M \cup \Delta)$ be a given isomorphic strict interpretation. Extend ϕ to bijective $\phi' : (N \cup \Sigma \cup \{\#\}) \to (M \cup \Delta \cup \{\#\})$ by assigning $\phi'(\#) = \#$. Then ϕ' describes an isomorphism between $\rho(G)$ and $\rho(H)$. Indeed, $\phi'(S) = \phi(S) = T$, $\phi'(P) = \phi(P) = Q$ and since $\phi(\Sigma) = \Delta$, there is an one-to-one correspondence between production sets $\{a \to \# : a \in \Sigma\}$ and $\{b \to \# : b \in \Delta\}$.
On the other hand, assume $\phi' : (N \cup \Sigma \cup \{\#\}) \to (M \cup \Delta \cup \{\#\})$ to be an isomorphism between $\rho(G)$ and $\rho(H)$, and consider the bijective restriction $\phi = \phi'|_{N \cup \Sigma}$. Then $\phi(S) = \phi'(S) = T$ and the unique structure of productions $a \to \#$ in (3.9) guarantees that $\phi(\Sigma) = \phi'(\Sigma) = \Delta$ and $\phi(P) = \phi'(P) = Q$ due to invariance of ϕ' on singleton terminal set $\{\#\}$. ■

Algorithm 3.1: Computation of template set Temp(G)
(Including corresponding substitution X_{ijk} and parameters n_i, p_i)

```
input   : Context-free grammar $G = (N, \Sigma, P, S)$
output  : Ordered sequence of templates Temp($G$), corresponding substitution
          $X_{ijk}$ and parameters $n_i, p_i$
runtime: $\mathcal{O}(s|P||N|\log(|N||\Sigma||P|))$
1  Set Temp($G$) $\leftarrow \emptyset$
2  Initialise partial map TempFunc : $P \to$ Temp($G$)
3  for $A \to \gamma \in P$ do                                   // $\mathcal{O}(s|P|\log(|N||\Sigma||P|))$
4      Let $\gamma = \gamma_1 \dots \gamma_n \in (N \cup \Sigma)^n$
5      Initialise partial map TempName[$A \to \gamma$] : $N \to \{L_1, \dots, L_{|N|}\}$ with
        TempName[$A \to \gamma$]($A$) $\leftarrow L_1$
6      Set $\omega \leftarrow L_1$
7      Set $k \leftarrow 2$
8      for $j \leftarrow 1$ to $n$ do                             // $\mathcal{O}(s\log(|N||\Sigma|))$
9          if $\gamma_j \in \Sigma$ then                           // $\mathcal{O}(\log|\Sigma|)$
10             $\omega \leftarrow \omega\gamma_j$
11         else                                               // $\mathcal{O}(\log|N|)$
12             if TempName[$A \to \gamma$]($\gamma_j$) undefined then
13                 Set TempName[$A \to \gamma$]($\gamma_j$) $\leftarrow L_k$
14                 $k \leftarrow k + 1$
15             end
16             $\omega \leftarrow \omega$TempName[$A \to \gamma$]($\gamma_j$)
17         end
18     end
19     Insert($\omega$, Temp($G$))                                 // $\mathcal{O}(s\log|P|)$
20     Set TempFunc($A \to \gamma$) $\leftarrow \omega$                       // $\mathcal{O}(s\log|P|)$
21 end
22 Sort( Temp($G$), $\leq_{\text{Lex}}$ )                              // $\mathcal{O}(s|P|\log|P|)$
23 RemoveDuplicates( Temp($G$))                              // $\mathcal{O}(s|P|\log|P|)$
24 Set $t \leftarrow$ |Temp($G$)|                            // Temp($G$) $= \{T_1, \dots, T_t\}$
   // $X_{ijk}$ is the $k$th distinct nonterminal in the $j$th production of
      the $i$th template
25 for $1 \leftarrow 1$ to $t$ do                                // $\mathcal{O}(s|P||N|\log(|N||P|))$
26     Set Temp$_i$ $\leftarrow$ TempFunc$^{-1}$($\{T_i\}$)                  // $\mathcal{O}(s|P|\log|P|)$
27     Sort( Temp$_i$, $\leq_{\text{Lex}}$ )                              // $\mathcal{O}(s|P|\log|P|)$
28     Set $p_i \leftarrow$ | Temp$_i$ |                       // Temp$_i$ $= \{K_{i1}, \dots, K_{ip_i}\}$
29     Set $n_i \leftarrow$ number of distinct nonterminals in $T_i$           // $\mathcal{O}(s)$
30     for $j \leftarrow 1$ to $p_i$ do                            // $\mathcal{O}(|P||N|\log|N|)$
31         for $k \leftarrow 1$ to $n_i$ do                         // $\mathcal{O}(|N|\log|N|)$
32             Set $X_{ijk} \leftarrow$ TempName[$K_{ij}$]$^{-1}$($L_k$)            // $\mathcal{O}(\log|N|)$
33         end
34     end
35 end
36 return Temp($G$), $X_{ijk}$, $n_i$, $p_i$
```

Algorithm 3.2: Construction of Graph(G)

```
input    : Regular grammar $G = (N, \Sigma, P, S)$
output   : Encoding as unlabelled undirected Graph($G$) $= (V, E)$
runtime: $\mathcal{O}(|N| + |\Sigma||P|) = \mathcal{O}(|\Sigma|^2|N|^2)$
1  Order and rename $\Sigma = \{a_1, \dots, a_r\}$
2  Initialise $E = \{[S]\}$
3  for $A \in N$ do                                         // $\mathcal{O}(|N|)$
4      for $i \leftarrow 1$ to $4$ do
5          $V \leftarrow E \cup \{[A, i]\}$
6          for $j \leftarrow 1$ to $i - 1$ do
7              $E \leftarrow E \cup \{\langle [A, j], [A, i] \rangle\}$
8          end
9      end
10 end
11 $E \leftarrow E \cup \{\langle [S], [S, 4] \rangle\}$
12 for $A \to a_k B \in P$ do                               // $\mathcal{O}(|P||\Sigma|)$
13     $V \leftarrow V \cup \{[A \to a_k B, 1]\}$
14     $E \leftarrow E \cup \{\langle [A, 1], [A \to a_k B, 1] \rangle\}$
15     $E \leftarrow E \cup \{\langle [A, 2], [A \to a_k B, 1] \rangle\}$
16     for $i \leftarrow 2$ to $k + 1$ do                        // $\mathcal{O}(|\Sigma|)$
17         $V \leftarrow V \cup \{[A \to a_k B, i]\}$
18         $E \leftarrow E \cup \{\langle [A \to a_k B, i - 1], [A \to a_k B, i] \rangle\}$
19     end
20     $E \leftarrow E \cup \{\langle [A \to a_k B, k + 1], [B, 4] \rangle\}$
21 end
22 for $A \to a_k \in P$ do                                 // $\mathcal{O}(|P||\Sigma|)$
23     $V \leftarrow V \cup \{[A \to a_k, 1]\}$
24     $E \leftarrow E \cup \{\langle [A, 1], [A \to a_k, 1] \rangle\}$
25     $E \leftarrow E \cup \{\langle [A, 2], [A \to a_k, 1] \rangle\}$
26     for $i \leftarrow 2$ to $k + 1$ do                        // $\mathcal{O}(|\Sigma|)$
27         $V \leftarrow V \cup \{[A \to a_k, i]\}$
28         $E \leftarrow E \cup \{\langle [A \to a_k, i - 1], [A \to a_k, i] \rangle\}$
29     end
30 end
31 return Graph($G$) $= (V, E)$
```

4 Structural Isomorphism of Term Rewriting Systems

4.1 Term Rewriting Systems and Structural Isomorphisms

We continue our ideas from the previous chapter. The left side of a production rule in context-free grammars was limited to one single nonterminal. Term rewriting systems partially lift this restriction by agglomerating rules of the form $f(x, g(y)) \to h(x)$, abstracting term manipulation by rewriting. We will see that term rewriting systems are much more specialised than general formal grammars, which justifies extensive study of computational properties of isomorphisms on these finite structures.

Definition 4.1 (Terms). Let $\mathcal{F}$ be a finite set of *function symbols*, $\mathcal{V}$ a finite, disjoint set of *variable symbols* and $\mathrm{ar} : \mathcal{F} \to \mathbb{N}_0$ a so called *arity* function. We say that $f \in \mathcal{F}$ is an *n-ary* function symbol, if $\mathrm{ar}(f) = n$. 0-ary function symbols are called *constants*. The set of *terms* $T(\mathcal{F}, \mathcal{V}, \mathrm{ar})$ is then inductively defined as follows:

(a) Every variable symbol is a term: $\mathcal{V} \subseteq T(\mathcal{F}, \mathcal{V}, \mathrm{ar})$.

(b) Every constant is a term: $\mathrm{ar}^{-1}(\{0\}) \subseteq T(\mathcal{F}, \mathcal{V}, \mathrm{ar})$.

(c) If $f \in \mathcal{F}$ is an n-ary function symbol, $n \in \mathbb{N}$, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is also a term.

For convenience, we omit ar and simply write $T(\mathcal{F}, \mathcal{V})$. If not stated otherwise, we reserve lower-case Latin letters at the beginning of the alphabet, like c, d, for constants, letters in the middle of the alphabet, like f, g, h, for arbitrary function symbols, and letters at the end of the alphabet, like x, y, z, for variable symbols.

Definition 4.2 (Free Variable/Function Symbols). The set of *free variable symbols* $\mathrm{Var}(t) \subseteq \mathcal{V}$ and set of *free function symbols* $\mathrm{Func}(t)$ of a term $t \in T(\mathcal{F}, \mathcal{V})$ is given as follows:

(a) $\mathrm{Var}(x) = \{x\}$, $\mathrm{Func}(x) = \emptyset$ for every $x \in \mathcal{V}$.

(b) $\mathrm{Var}(c) = \emptyset$, $\mathrm{Func}(c) = \{c\}$ for every constant $c \in \mathcal{F}$.

(c) If f is an n-ary function symbol and $t_1, \ldots, t_n$ terms, then $\mathrm{Var}\big(f(t_1, \ldots, t_n)\big) = \bigcup_{j=1}^{n} \mathrm{Var}(t_i)$ and $\mathrm{Func}\big(f(t_1, \ldots, t_n)\big) = \{f\} \cup \bigcup_{j=1}^{n} \mathrm{Func}(t_i)$.

Example 4.3. Let $\mathcal{F} = \{f, h, g, c\}$, $\mathcal{V} = \{x, y\}$ and $\mathrm{ar} = \{c \mapsto 0, g \mapsto 1, h \mapsto 1, f \mapsto 2\}$. Then x, c, $g(x)$, $f(g(x), y)$, $f(f(h(c), x), g(y))$ are all valid terms and $\mathrm{Var}\big(f(g(x), y)\big) = \{x, y\}$, $\mathrm{Func}\big(f(g(x), y)\big) = \{g, f\}$. Examples for invalid terms are f, $f(x)$ (f is neither a constant nor is it of arity 1) or $c(x)$ (c is a constant).

Definition 4.4 (Term Rewriting System). Recall Definition 4.1. A *term rewriting system (TRS)* is a tuple $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$, where $\mathcal{R} \subseteq (T(\mathcal{F}, \mathcal{V}) \setminus \mathcal{V}) \times T(\mathcal{F}, \mathcal{V})$ is a finite set of *rewriting rules* with additional condition $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$ for every $(\ell, r) \in \mathcal{R}$. That is, the right side of a rule does not introduce new variable symbols. To emphasise the rule character, we write $\ell \to r$ instead of $(\ell, r) \in \mathcal{R}$. A *string rewriting system* or *semi-Thue system (STS)* is a special case of term rewriting system, where all non-constant function symbols are of arity one, $|\mathcal{V}| = 1$ and there is a single unique constant not appearing on the left side of a rule, i.e. every rule is of the form $f_1(\ldots(f_n(x))) \to g_1(\ldots(g_m(x)))$ or $f_1(\ldots(f_n(x))) \to g_1(\ldots(g_m(c)))$, so that variable symbol $x \in \mathcal{V}$ and constant $c \in \mathcal{F}$ are unique.

Example 4.5 (Cont. of Example 4.3). Exemplary rewriting rules are $h(y) \to f(c, y)$, $f(g(x), y) \to g(x)$, $h(g(x)) \to g(x)$ or $h(y) \to c$, with the latter two forming a string rewriting system. Note that we have a total of three conditions: (1) Left and right side of a rule have to be valid terms, (2) the right side is not allowed to be a single variable symbol, (3) the right side does not introduce new variable symbols. Then for example, $f(x) \to c$ violates (1), $x \to h(x)$ violates (2) and $g(x) \to h(y)$ violates (3).

Similar to Chapter 3, we are interested in structural isomorphisms between term rewriting systems, i.e. we want to rewrite symbols to go from one system to an other. Before we can discuss such mappings however, we have to do a bit of preparation to account for the added layer of complexity.

Definition 4.6 (Term Homomorphism). A term homomorphism between two term sets $T(\mathcal{F}_1, \mathcal{V}_1, \mathrm{ar}_1)$ and $T(\mathcal{F}_2, \mathcal{V}_2, \mathrm{ar}_2)$ is a bijective map $\phi : (\mathcal{F}_1 \cup \mathcal{V}_1) \to (\mathcal{F}_2 \cup \mathcal{V}_2)$, such that $\phi(\mathcal{V}_1) = \mathcal{V}_2$ and $\mathrm{ar}_1(f) = \mathrm{ar}_2(\phi(f))$ for every $f \in \mathcal{F}_1$, canonically extended to all of $T(\mathcal{F}_1, \mathcal{V}_1)$ via $\phi(f(t_1, \ldots, t_n)) = \phi(f)(\phi(t_1), \ldots, \phi(t_n))$. Homomorphism ϕ is *$\mathcal{V}$-invariant* if $\mathcal{V}_1 = \mathcal{V}_2$ and $\phi|_{\mathcal{V}_1} = \mathrm{id}_{\mathcal{V}_1}$, and *$\mathcal{F}$-invariant* if $\mathcal{F}_1 = \mathcal{F}_2$ and $\phi|_{\mathcal{F}_1} = \mathrm{id}_{\mathcal{F}_1}$.

Definition above reads similar to Definition 3.2 if we interpret variable symbols are terminals and function symbols as nonterminals. Additionally, we demand term homomorphisms to be arity-preserving. Now we can introduce the notion of ($\mathcal{V}$-/$\mathcal{F}$-)equivalent rewriting rules and consequently ($\mathcal{V}$-/$\mathcal{F}$-)normal forms.

Definition 4.7 (Equivalence of Rewriting Rules, Normal Forms). Let $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ be a term rewriting system.

(a) R is in *$\mathcal{V}$-normal form*, if we cannot find distinct rewriting rules $\ell \to r, \ell' \to r' \in \mathcal{R}$ and $\mathcal{F}$-invariant term endomorphism $\phi : (\mathcal{F} \cup \mathcal{V}) \to (\mathcal{F} \cup \mathcal{V})$ with $\phi(\ell) \to \phi(r) = \ell' \to r'$. Otherwise we call $\ell \to r$ and $\ell' \to r'$ *$\mathcal{V}$-equivalent.*

(b) R is in *$\mathcal{F}$-normal form*, if we cannot find distinct rewriting rules $\ell \to r, \ell' \to r' \in \mathcal{R}$ and $\mathcal{V}$-invariant term endomorphism $\phi : (\mathcal{F} \cup \mathcal{V}) \to (\mathcal{F} \cup \mathcal{V})$ with $\phi(\ell) \to \phi(r) = \ell' \to r'$. Otherwise we call $\ell \to r$ and $\ell' \to r'$ *$\mathcal{F}$-equivalent.*

(c) R is in *normal form*, if f we cannot find distinct rewriting rules $\ell \to r, \ell' \to r' \in \mathcal{R}$ and term endomorphism $\phi : (\mathcal{F} \cup \mathcal{V}) \to (\mathcal{F} \cup \mathcal{V})$ with $\phi(\ell) \to \phi(r) = \ell' \to r'$. Otherwise we call $\ell \to r$ and $\ell' \to r'$ *equivalent.*

Clearly implications $(c) \implies (a)$ and $(c) \implies (b)$ hold. A term rewriting system in normal form is already in $\mathcal{V}$-/$\mathcal{F}$-normal form.

Example 4.8 (Cont. of Example 4.5). Both

$$\phi_1 = \{x \to y, y \mapsto x, c \mapsto c, g \mapsto g, h \mapsto h, f \mapsto f\}$$
$$\text{and } \phi_2 = \{x \to x, y \mapsto y, c \mapsto c, h \mapsto g, g \mapsto h, f \mapsto f\}$$

are valid term homomorphisms on $T(\mathcal{F}, \mathcal{V})$ itself, with the former being $\mathcal{F}$-invariant and the latter $\mathcal{V}$-invariant. Moreover, let

$$\mathcal{R} = \{f(g(x), y) \to h(x), f(h(x), y) \to g(x), f(g(y), x) \to h(y)\}.$$

Then $\mathcal{R}$ is neither in $\mathcal{V}$-normal form, $\phi_1(f(g(x), y)) \to \phi_1(h(x)) = f(g(y), x) \to h(y)$ ($f(g(x), y) \to h(x)$ and $f(g(y), x) \to h(y)$ are $\mathcal{V}$-equivalent), nor in $\mathcal{F}$-normal form, $\phi_2(f(g(x), y)) \to \phi_2(h(x)) = f(h(x), y) \to g(x)$ ($f(g(x), y) \to h(x)$ and $f(h(x), y) \to g(x)$ are $\mathcal{F}$-equivalent).

Isomorphism	$\mathcal{V}$-inv.	$\mathcal{V}$-global	$\mathcal{V}$-local	$\mathcal{F}$-inv.	$\mathcal{F}$-global	$\mathcal{F}$-local
GE		✓			✓	
GVE		✓		✓		
GFE	✓				✓	
LE			✓			✓
VSE		✓				✓
FSE			✓		✓	
LVE			✓	✓		
LFE	✓					✓

Table 4.1: Comparison of TRS isomorphisms

Definition 4.9 (TRS Isomorphisms). Let $R_1 = (\mathcal{F}_1, \mathcal{V}_1, \mathcal{R}_1)$ and $R_2 = (\mathcal{F}_2, \mathcal{V}_2, \mathcal{R}_2)$ ($\mathrm{ar}_1, \mathrm{ar}_2$ implicitly) be two given term rewriting systems.
R_1 and R_2 are *globally isomorphic*, in terms $R_1 \cong_{\mathbf{GE}} R_2$, if we find a term homomorphism $\phi : (\mathcal{F}_1 \cup \mathcal{V}_1) \to (\mathcal{F}_2 \cup \mathcal{V}_2)$ with $\phi(\mathcal{R}_1) = \mathcal{R}_2$. Map ϕ is then called a *global TRS isomorphism.*

(a) If ϕ is $\mathcal{F}$-invariant, we say that R_1 and R_2 are *$\mathcal{V}$-globally isomorphic*, $R_1 \cong_{\mathbf{GVE}} R_2$, and call ϕ a *$\mathcal{V}$-global TRS isomorphism.*

(b) If ϕ is $\mathcal{V}$-invariant, we say that R_1 and R_2 are *$\mathcal{F}$-globally isomorphic*, $R_1 \cong_{\mathbf{GFE}} R_2$, and call ϕ a *$\mathcal{F}$-global TRS isomorphism.*

R_1 and R_2 are *locally isomorphic*, in terms $R_1 \cong_{\mathbf{LE}} R_2$, if they are in normal form and we find term homomorphisms $\phi_i : (\mathcal{F}_1 \cup \mathcal{V}_1) \to (\mathcal{F}_2 \cup \mathcal{V}_2)$, $1 \leq i \leq n = |R_1|$, such that

$$\phi(\mathcal{R}_1) = \{\phi_1(\ell_1) \to \phi_1(r_1), \ldots, \phi_n(\ell_n) \to \phi_n(r_n)\} = \mathcal{R}_2,$$

where we denote with $\phi = (\phi_1, \ldots, \phi_n)$ the family of such term homomorphisms. Family ϕ is then called a *local TRS isomorphism.*

(c) If R_1 and R_2 are in $\mathcal{F}$-normal form and additionally $\phi_1|_{\mathcal{V}_1} = \ldots = \phi_n|_{\mathcal{V}_1}$, we say that R_1 and R_2 are *$\mathcal{V}$-standard isomorphic*, $R_1 \cong_{\mathbf{VSE}} R_2$, and call ϕ a *$\mathcal{V}$-standard isomorphism.*

(d) If R_1 and R_2 are in $\mathcal{V}$-normal form and additionally $\phi_1|_{\mathcal{F}_1} = \ldots = \phi_n|_{\mathcal{F}_1}$, we say that R_1 and R_2 are *$\mathcal{F}$-standard isomorphic*, $R_1 \cong_{\mathbf{FSE}} R_2$, and call ϕ a *$\mathcal{F}$-standard isomorphism.*

(e) R_1 and R_2 are *$\mathcal{V}$-locally isomorphic*, $R_1 \cong_{\mathbf{LVE}} R_2$, if they are $\mathcal{F}$-standard isomorphic and the ϕ_i's are additionally $\mathcal{F}$-invariant.

(f) R_1 and R_2 are *$\mathcal{F}$-locally isomorphic*, $R_1 \cong_{\mathbf{LFE}} R_2$, if they are $\mathcal{V}$-standard isomorphic and the ϕ_i's are additionally $\mathcal{V}$-invariant.

In the case of string rewriting systems, we can w.l.o.g. assume that respective variable symbol and constant symbol are the same, i.e. $\mathcal{V}_1 = \mathcal{V}_2$ and $\mathrm{ar}_1^{-1}(\{0\}) = \mathrm{ar}_2^{-1}(\{0\})$. Then two string rewriting systems are *(globally) isomorphic* if and only if they are globally isomorphic as term rewriting systems. A summary concerning all types of TRS isomorphism can be found in Table 4.1.

Now it is evident why normal forms are helpful and were introduced in the first place. They prevent "shrinking" of rewriting rule sets and ensure symmetry and transitivity of the isomorphism-relation, which will be shown and discussed in Lemma 4.11.

Lemma 4.10. *The following strict implications hold:*

$$\begin{tikzcd}
 & \textbf{GFE} \arrow[dl, Rightarrow, "(i)"'] \arrow[dr, Rightarrow, "(ii)"] & & \textbf{GVE} \arrow[dl, Rightarrow, "(iii)"'] \arrow[dr, Rightarrow, "(iv)"] & \\
\textbf{LFE} \arrow[dr, Rightarrow, "(v)"'] & & \textbf{GE} \arrow[dl, Rightarrow, "(vi)"] \arrow[dr, Rightarrow, "(vii)"'] & & \textbf{LVE} \arrow[dl, Rightarrow, "(viii)"] \\
 & \textbf{VSE} \arrow[dr, Rightarrow, "(ix)"'] & & \textbf{FSE} \arrow[dl, Rightarrow, "(x)"] & \\
 & & \textbf{LE} & &
\end{tikzcd} \tag{4.1}$$

Proof. Implications in (4.1) can be directly read from Table 4.1, where we use that $\mathcal{F}$-invariant $\Longrightarrow$ $\mathcal{F}$-global $\Longrightarrow$ $\mathcal{F}$-local and $\mathcal{V}$-invariant $\Longrightarrow$ $\mathcal{V}$-global $\Longrightarrow$ $\mathcal{V}$-local.
To proof that those implications are in fact strict, consider the following counter examples. We will just state rewriting rule sets $\mathcal{R}_1, \mathcal{R}_2$, variable/function symbol sets and arity functions are given implicitly.

(i) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(x,y) \to c, g(x) \to d\}$
are $\mathcal{F}$-locally isomorphic, but not $\mathcal{F}$-globally isomorphic.

(ii) $\mathcal{R}_1 = \{f(x,y) \to c\}$ and $\mathcal{R}_2 = \{f(y,x) \to c\}$
are globally isomorphic ,but not $\mathcal{F}$-globally isomorphic.

(iii) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(x,y) \to c, h(x) \to c\}$
are globally isomorphic, but not $\mathcal{V}$-globally isomorphic.

(iv) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(y,x) \to c, g(x) \to c\}$
are $\mathcal{V}$-locally isomorphic, but not $\mathcal{V}$-globally isomorphic.

(v) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(y,x) \to c, g(y) \to d\}$
are $\mathcal{V}$-standard isomorphic, but not $\mathcal{F}$-locally isomorphic.

(vi) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(y,x) \to c, g(y) \to d\}$
are $\mathcal{V}$-standard isomorphic, but not globally isomorphic.

(vii) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(x,y) \to c, g(y) \to c\}$
are $\mathcal{F}$-standard isomorphic, but not globally isomorphic.

(viii) $\mathcal{R}_1 = \{f(x) \to c\}$ and $\mathcal{R}_2 = \{g(x) \to c\}$
are $\mathcal{F}$-standard isomorphic, but not $\mathcal{V}$-locally isomorphic.

(ix) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(y,x) \to c, g(x) \to c\}$
are locally isomorphic, but not $\mathcal{V}$-standard isomorphic.

(x) $\mathcal{R}_1 = \{f(x,y) \to c, g(x) \to c\}$ and $\mathcal{R}_2 = \{f(y,x) \to c, g(x) \to d\}$
are locally isomorphic, but not $\mathcal{F}$-standard isomorphic.

■

From now on we assume, w.l.o.g., $\mathcal{F}, \mathcal{V}$ to be minimal in the sense that

$$\mathcal{F} = \bigcup_{\ell \to r \in \mathcal{R}} \big(\operatorname{Func}(\ell) \cup \operatorname{Func}(r)\big) \quad \wedge \quad \mathcal{V} = \bigcup_{\ell \to r \in \mathcal{R}} \operatorname{Var}(\ell).$$

In other words, every function/variable symbol is used in at least one rewriting rule $\ell \to r$. This prevents unintended non-isomorphism of term rewriting systems, due to ill-defined symbol sets. If not, we can compute minimal variable and function symbol sets in at worst $\mathcal{O}(s|\mathcal{R}|\log s)$, where $|\mathcal{R}|$ is the number of rewriting rules and s an fixed upper bound on the maximal length of rules $\ell \to r$ in $\mathcal{R}$. This can be accomplished by scanning every rule individually from left to right, collecting distinct symbols. Concrete implementations then are dependent on how rules, or more importantly terms, are represented and stored as data.

Lemma 4.11. *The binary relation $R_1 \sim R_2 \iff R_1$ and R_2 are locally isomorphic term rewriting systems, is an equivalence relation.*

Proof. Fix term rewriting systems R_1, R_2, R_3 such that $R_1 \cong_{\mathbf{LE}} R_2$ and $R_2 \cong_{\mathbf{LE}} R_3$ via local isomorphisms ϕ and ϕ' respectively, and denote with

$$\mathcal{R}_i = \left\{\ell_1^{(i)} \to r_1^{(i)}, \ldots, \ell_{n_i}^{(i)} \to r_{n_i}^{(i)}\right\},$$

the respective rule sets, where $n_i = |\mathcal{R}_i|$ for $i = 1, 2, 3$. Note that by Definition 4.6, inverses and compositions of term homomorphism are again term homomorphisms. We claim that R_1, R_2 and R_3 have the same number of rewriting rules, $n_1 = n_2 = n_3 = n$. Indeed, if $\ell_i^{(1)} \to r_i^{(1)}, \ell_j^{(1)} \to r_j^{(1)}$, $1 \leq i < j \leq n_1$ with

$$\phi_i(\ell_i^{(1)}) \to \phi_i(r_i^{(1)}) = \phi_j(\ell_j^{(1)}) \to \phi_j(r_j^{(1)}), \tag{4.2}$$

then for term homomorphism $\psi_i := \phi_j^{-1} \circ \phi_i$, $\psi_i(\ell_i^{(1)}) \to \psi_i(r_i^{(1)}) = \ell_j^{(1)} \to r_j^{(1)}$, due to equation (4.2), in contradiction to R_1 in normal form. The same holds for R_2. W.l.o.g. assume R_1, R_2, R_2 to be ordered in such a way that for all $1 \leq i \leq n$,

$$\phi_i(\ell_i^{(1)}) \to \phi_i(r_i^{(1)}) = \ell_i^{(2)} \to r_i^{(2)} \quad \wedge \quad \phi_i'(\ell_i^{(2)}) \to \phi_i'(r_i^{(2)}) = \ell_i^{(3)} \to r_i^{(3)}.$$

Then $R_1 \cong_{\mathbf{LVE}} R_1$ and $R_2 \cong_{\mathbf{LVE}} R_1$ by choosing $\mathrm{id}_{\mathcal{F}_1 \cup \mathcal{V}_1}$ and $\phi^{-1} = (\phi_1^{-1}, \ldots, \phi_n^{-1})$ respectively. Moreover, $R_1 \cong_{\mathbf{LVE}} R_3$ due to $\psi(\mathcal{R}_1) = \mathcal{R}_3$, where $\psi = (\phi_1' \circ \phi_1, \ldots, \phi_n' \circ \phi_n)$ is the composition of ϕ and ϕ'. ■

Normal form requirement in Lemma 4.11 was indispensable to assure the invariance of order of rewriting rule set under application of term homomorphisms and to ensure that inverses and compositions of term homomorphisms can be used to proof symmetry and transitivity of the binary relation. The same claim also holds for other types of isomorphism as defined in Definition 4.9 with respective normal forms. This concludes the following

Corollary 4.12. *The binary relation $R_1 \sim R_2 \iff R_1$ and R_2 are ($\mathcal{V}$-/$\mathcal{F}$-)locally/globally/standard isomorphic term rewriting systems, is an equivalence relation.*

4.2 Local TRS Isomorphisms are in P

We will state two explicit polynomial time algorithms which solve **LVE**, **LFE**, **LE**, and will later serve to justify polynomial reductions from **FSE** to **GFE** and **VSE** to **GVE**. Recall the previous chapter. To show structural isomorphism of context-free grammars, we gave a polynomial reduction to regular grammars using so called templates. We will pursue the same idea. Depending on the type of local isomorphism, we will either rewrite variable symbols or function symbols or both on a per rule basis.

Lemma 4.13. *Every term rewriting system $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ can be brought into maximal ($\mathcal{V}$-/$\mathcal{F}$-)normal form $R' = (\mathcal{F}, \mathcal{V}, \mathcal{R}')$. That is, $\mathcal{R}' \subseteq \mathcal{R}$ and for every $\mathcal{R}' \subsetneq \mathcal{R}'' \subseteq \mathcal{R}$, $(\mathcal{F}, \mathcal{V}, \mathcal{R}'')$ is not in ($\mathcal{V}$-/$\mathcal{F}$-)normal form. This ($\mathcal{V}$-/$\mathcal{F}$-)normal form is unique up to ($\mathcal{V}$-/$\mathcal{F}$-)equivalence of rewriting rules and can be constructed in polynomial time.*

Proof. Let $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ be a term rewriting system, $\mathcal{R} = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$, and fix $1 \leq j \leq n$. Set $\mathcal{V}_S = \{x_1, \ldots, x_K\}$, $K = |\mathcal{V}|$, and $\mathcal{F}_S = \{f_{k,l} : l \in L, 1 \leq k \leq p_l\}$, where $L = \{\text{ar}(f) : f \in \mathcal{F}\}$, $p_l = |\{f \in \mathcal{F} : \text{ar}(f) = l\}|$ and $|\mathcal{F}_S| = |\mathcal{F}|$. For $\mathcal{F}$ and $\mathcal{V}$ from Example 4.8 we get new variable symbol set $\mathcal{V}_S = \{x_1, x_2\}$ and function symbol set $\mathcal{F}_S = \{f_{1,0}, f_{1,1}, f_{2,1}, f_{1,3}\}$.
We define a local rewriting $\phi_j : (\mathcal{F} \cup \mathcal{V}) \to (\mathcal{F} \cup \mathcal{V}_S)$ of variable symbols the following way: Consider ℓ_j and replace each occurrence of a variable with x_k, where x_k refers to the kth distinct variable symbol in ℓ_j. Since $\text{Var}(r_j) \subseteq \text{Var}(\ell_j)$ this is already sufficient to state a $\mathcal{F}$-invariant term homomorphism. Analogously, rewrite function symbols locally via $\phi'_j : (\mathcal{F} \cup \mathcal{V}) \to (\mathcal{F}_S \cup \mathcal{V})$ by replacing each occurrence of a function symbol with $f_{k,l}$, where $f_{k,l}$ refers to the kth distinct function symbol in $\ell_j r_j$ of arity l (k depends on l), resulting in a $\mathcal{V}$-invariant term homomorphism. Moreover, assign $\phi_{\mathcal{V}} = (\phi_1, \ldots, \phi_n)$ and $\phi_{\mathcal{F}} = (\phi'_1, \ldots, \phi'_n)$. Refer to Algorithm 4.1 and Algorithm 4.2 for exemplary implementations. Both TRS isomorphisms $\phi_{\mathcal{V}}$ and $\phi_{\mathcal{F}}$ can be computed in at worst $\mathcal{O}(s|\mathcal{R}||\mathcal{V}||\mathcal{F}|\log(|\mathcal{V}||\mathcal{F}|))$. We call $\phi_i(\ell_i) \to \phi_i(r_i)$ the $\mathcal{V}$-local template and $\phi'_i(\ell_i) \to \phi'_i(r_i)$ the $\mathcal{F}$-local template of rewriting rule $\ell_i \to r_i$.
Before we proceed with the proof, a quick example: Recall Example 4.8. The $\mathcal{V}$-local template of $f(g(x), y) \to h(x)$ is $f(g(x_1), x_2) \to h(x_1)$ and the $\mathcal{F}$-local template is $f_{1,2}(f_{1,1}(x), y) \to f_{2,1}(x)$. Extend this to rule set $\mathcal{R} = \{f(g(x), y) \to h(x), g(f(x, y)) \to c\}$. Now we see why double indices in the functions symbol set are needed to ensure consistent arity. Rewriting function symbols in the same pattern as variable symbols would yield templates $\{f_1(f_2(x), y) \to f_3(x), f_1(f_2(x, y)) \to f_3\}$ which can not be constructed out of one singular term set due to both f_1 and f_2 appearing with arity 1 and 2. Moreover, it is not sufficient to just trivially increase the first index, i.e. rewriting $f(g(x), y) \to h(x)$ as $f_{1,2}(f_{2,2}, y) \to f_{3,0}(x)$. The resulting template set would be

$$\{f_{1,2}(f_{2,2}(x), y) \to f_{3,0}(x), f_{1,1}(f_{2,1}(x, y)) \to f_{3,0}\},$$

which needs underlying function symbol set of atleast order five, namely $\{f_{1,1}, f_{1,2}, f_{2,1}, f_{2,2}, f_{3,0}\}$. This prevents bijectivity of ϕ'_j's, and thus them being well-defined term homomorphisms.

We now only consider $\mathcal{V}$-normal forms and $\mathcal{V}$-local templates, the case of $\mathcal{F}$-normal forms and $\mathcal{F}$-local templates is shown analogously. $\mathcal{V}$-equivalence of distinct rewriting rules $\ell \to r, \ell' \to r'$ is equivalent to $\mathcal{V}$-local isomorphism of corresponding singleton rule sets $\{\ell \to r\}$ and $\{\ell' \to r'\}$. This justifies the use of the "equivalence", i.e. being $\mathcal{V}$-equivalent is indeed an equivalence relation, and we can work with induced equivalence classes $[\ell \to r] = \{\ell' \to r' \in \mathcal{R} : \mathcal{V}\text{-equivalent to } \ell \to r\}$. Moreover, these equivalence classes can easily be determined once we have $\phi_\mathcal{V}$, since by Corollary 4.12,

$$[\ell_i \to r_i] = \{\ell_j \to r_j \in \mathcal{R} : \phi_i(\ell_i) \to \phi_i(r_i) = \phi_j(\ell_j) \to \phi_j(r_j)\}.$$

In other words, $\phi_\mathcal{V}(\mathcal{R})$ directly implies a partitioning of $\mathcal{R}$ into $1 \leq m \leq n$ sets of rewriting rules of same $\mathcal{V}$-local template. Then every representation system $\mathcal{R}' = \{\ell_{i_1} \to r_{i_1}, \ldots, \ell_{i_m} \to r_{i_m}\}$ of this equivalence relation is already in maximal $\mathcal{V}$-normal form. Indeed, no rules are $\mathcal{V}$-equivalent due to assumption, and if we extend $\mathcal{R}'$, we necessarily have to choose a rule already in one of the represented equivalence classes, which would be a contradiction to $\mathcal{V}$-normality.
In the case of normal forms, consider $\psi = (\psi_1, \ldots, \psi_n)$, $\psi_i = (\phi'_i|_\mathcal{F} \sqcup \mathrm{id}_{\mathcal{V}_S}) \circ \phi_i$, well-defined due to $\mathcal{F}$-invariance of ϕ_i for each $1 \leq i \leq n$. Computationally speaking, we sequentially apply Algorithm 4.1 and then Algorithm 4.2, i.e. we take the maximal $\mathcal{F}$-normal form of the maximal $\mathcal{V}$-normal form of R. The claim then is shown with the same reasoning as above. ■

Definition 4.14 (Local Templates)**.** Recall notation from proof above and refer to Algorithm 4.1 and Algorithm 4.2. We call $\phi_i(\ell_i) \to \phi_i(r_i)$ the *$\mathcal{V}$-local template*, $\phi'_i(\ell_i) \to \phi'_i(r_i)$ the *$\mathcal{F}$-local template* and $\psi_i(\ell_i) \to \psi_i(r_i)$ simply the *template* of rewriting rule $\ell_i \to r_i \in \mathcal{R}$. Notion of ($\mathcal{V}$-/$\mathcal{F}$-local) template then is extended to all of term rewriting system R.

Corollary 4.15. *A term rewriting system R in ($\mathcal{V}$-/$\mathcal{F}$-)normal form is ($\mathcal{V}$-/$\mathcal{F}$-)locally isomorphic to its ($\mathcal{V}$-/$\mathcal{F}$-)local template. Moreover, as direct consequence of Corollary 4.12, two term rewriting systems R_1 and R_2 in ($\mathcal{V}$-/$\mathcal{F}$-)normal form are ($\mathcal{V}$-/$\mathcal{F}$-)locally isomorphic if and only if they have the same ($\mathcal{V}$-/$\mathcal{F}$-)local template.*

Proof. Follows immediately by proof of Lemma 4.13, inspection of Algorithm 4.1 and Algorithm 4.2, and the fact that $\phi_\mathcal{V}$ is $\mathcal{F}$-invariant and $\phi_\mathcal{F}$ is $\mathcal{V}$-invariant. ■

Now we can easily proof the initial statement from the start of the section.

Theorem 4.16 (Local TRS Isomorphisms are in **P**)**.** *Isomorphism* **LE**, **LVE** *and* **LFE** *can be decided in polynomial time.*

Proof. Recall $\phi_\mathcal{V}, \phi_\mathcal{F}, \psi$ from the proof of Lemma 4.13. Again, only consider the case of local isomorphism **LE**, the other cases are shown analogously. By Corollary 4.15, two term rewriting systems R_1 and R_2 in normal form are locally isomorphic if and only if their local templates R'_1 and R'_2 are the same. But this is already the case if and only if the sets of rewriting rules of R'_1 and R'_2 are the same. Both, construction of templates and verification of set equality can be done in polynomial time, with latter being possible in at most most $\mathcal{O}(s_1 n_1 \log n_1 + s_2 n_2 \log n_2)$, where $n_i = |\mathcal{R}_i|$ is the number of rewriting rules and $s_i = \max\{|\ell r| : \ell \to r \in \mathcal{R}_i\}$ is an upper bound on the length of rules in R_i, $i = 1, 2$. ■

4.3 Term Trees and TRS-Forests

In preparation for Theorem 4.20, we discuss an encoding from term rewriting systems into a forest of labelled directed trees. From now on until the end of the chapter, assume $\mathcal{F} \cup \mathcal{V}$ and $\mathbb{N}_0$ to be disjoint (symbol) sets.

Definition 4.17 (Rooted Graph, Tree). Let $G = (V, E, L, \mathrm{lab})$ be a possibly labelled, directed graph. Recall that vertex $v \in V$ is called a *root*, if it has no incoming edges, that is, the indegree is zero. Graph G is called *connected*, if the corresponding undirected graph is connected, i.e. if we can find a path between any two vertices in undirected G. Vertex $v \in V$ is called *initial*, if every other node $w \in V \setminus \{v\}$ is reachable from v, i.e. there exists a directed path from v to w. We then call G *rooted*, if G is connected and has an unique, initial root. If additionally there is at most one path between any two vertices in G, we say that G is a *tree*.

We can canonically encode a term into a tree by simply respecting the term structure. For example, $f(x, y)$ would be a tree with f-labelled root and two child-vertices, labelled with x and y respectively. Note that the order of arguments is of utmost importance. Terms $f(x, y)$ and $f(y, x)$ have the same basic structure, function symbol, followed by two variable symbols as arguments, and thus the same naive encoding. Once we consider rules, that is not sufficient any more. It can easily be checked that no type of isomorphism between term rewriting systems $\{f(x, y) \to g(x)\}$ and $\{f(y, x) \to g(x)\}$ can be found even though they posses the same naive encoding. The following **GI**-completeness proofs will thus make heavy use of OOLDG-trees. In essence, we will identify term rewriting systems by a forest of unconnected OOLDG-trees, where every connected component directly corresponds to a single rewriting rule. Dependent on type of TRS isomorphism, we will need to modify this forest by adding additional vertices and edges or replacing existing labels. Of particular interest are **GVE**, **GFE** and **GE**, where we will explicitly state such a polynomial reduction into OOLDGs. The remaining isomorphisms can then be reduced to those three cases, saving unnecessary constructions.

We will need the following notations: Suppose $T_1, \dots, T_n$ are OOLDG-trees with distinct (labelled) roots $v_1, \dots, v_n$. Graph $\mathrm{Join}(v, l, T_1, \dots, T_n)$ then is the OOLDG-tree with l-labelled root v and ordered subtrees $T_1, \dots, T_n$, i.e. extend the union of $T_1, \dots, T_n$ to a new OOLDG-tree by introducing new edges (v, v_i, i), $1 \le i \le n$. An example can be found in Figure 4.1.

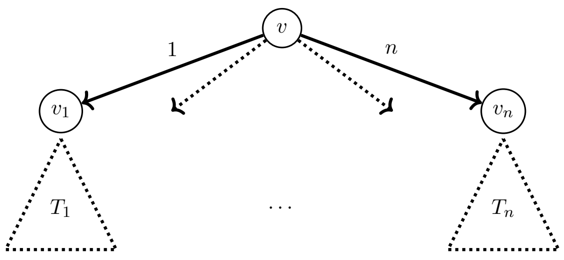


Figure 4.1: Visualisation of Join$(v, l, T_1, \dots, T_n)$

Definition 4.18 (Term Tree, TRS-Forest)**.** Let $t \in T(\mathcal{F}, \mathcal{V})$ be a term. The *term tree* of t is an OOLDG-tree Tree(t), inductively defined as follows:

(a) If $t = x$ or $t = c$ for some variable symbol $x \in \mathcal{V}$ or constant $c \in \mathcal{F}$, Tree(t) is a single vertex with respective label.

(b) If $t = f(t_1, \dots, t_n)$ for some n-ary function symbol $f \in \mathcal{F}$ and terms $t_1, \dots, t_n \in T(\mathcal{F}, \mathcal{V})$, then $\text{Tree}(t) = \text{Join}\big(v, f, \text{Tree}(t_1), \dots, \text{Tree}(t_n)\big)$, where v is a fresh vertex.

For a rewriting rule $\ell \to r$, we generalise the definition to $\text{Tree}(\ell \to r) = \text{Join}\big([\ell \to r], T, \text{Tree}(\ell), \text{Tree}(r)\big)$, where $[\ell \to r]$ is a fresh T-labelled root. We then call $G(R) = \bigcup_{\ell \to r \in \mathcal{R}} \text{Tree}(\ell \to r)$, $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ term rewriting system, i.e. the union of all rule trees of rewriting rules in $\mathcal{R}$, the *TRS-forest* of R (an agglomeration of trees is a forest). Note that $G(R)$ has exactly $|V| - |\mathcal{R}|$ many edges, or in other words, $|E|$ is in $\mathcal{O}(|V|)$.

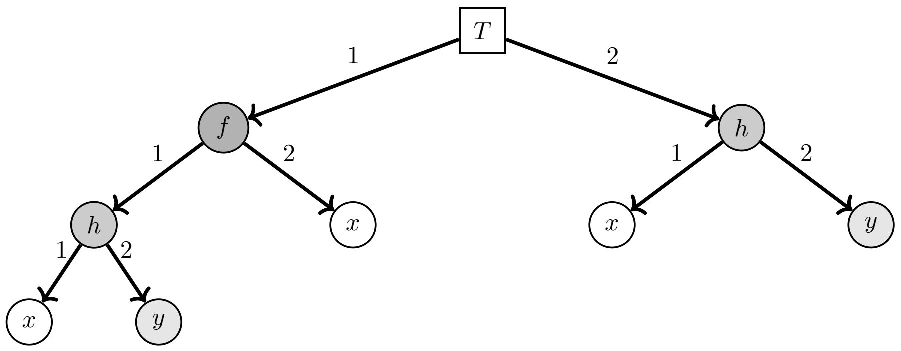


Figure 4.2: Term tree $\text{Tree}\big(f(h(x,y),x) \to h(x,y)\big)$

If not stated otherwise, let $G(R) = (V, E, L = \mathcal{F} \cup \mathcal{V} \cup \{T\} \cup \mathbb{N}, \text{lab})$, with optional indices when working with more than one rewriting system, and denote with $V(G)$ the not explicitly stated vertex set of some given graph G, used in particular when working with term trees. Formally, we should choose label set $\mathcal{F} \cup \mathcal{V} \cup \{T\} \cup \{1, \dots, m\}$, where m is atleast 2 and an upper bound for arity of function symbols in $\mathcal{F}$, i.e. $m = \max\{2, \mathrm{ar}(f) : f \in \mathcal{F}\} < \infty$ would be the minimal choice. Due to edge- and vertex-labels being distinct, edge-labels restricted to positive integers, we just assume the label set to be as stated initially to avoid unnecessary notation. The TRS-forest can be constructed in polynomial time, i.e. in $\mathcal{O}(s|\mathcal{R}|)$, where s is an upper bound on the length of rewriting rules, if we, for example, use doubly linked lists. In this model, every tree $\mathrm{Tree}(v, l, T_1, \ldots, T_n)$ belongs to a class `Tree` with attributes *next* of type `Tree[ ]` (array of `Tree`), *prev* of type `Tree` and *data* of some arbitrary type (here $\mathcal{F} \cup \mathcal{V} \cup \{T\}$), initialised by $\texttt{Tree}(l, n, S)$, where l is the data, n the number of child vertices, i.e. the length of *next*, and S is stored in *prev*. In the case of root vertices or $n = 0$, we initialise the respective attribute with `null`, i.e. $S = \texttt{null}$ or $\textit{next} = \texttt{null}$. Term tree $\mathrm{Tree}(\ell \to r)$ is then recursively constructed by linearly parsing each rewriting rule $\ell \to r$. The following Lemma will highlight the correlation between (global) isomorphism of term rewriting systems and isomorphism between corresponding TRS-forests. It will be the main ingredient for the proof of our core theorem.

Lemma 4.19. *Let R_1, R_2 be two term rewriting systems and consider the following two statements:*

- (i) *R_1 and R_2 are globally isomorphic.*
- (ii) *$G(R_1)$ and $G(R_2)$ are isomorphic.*

$(i) \implies (ii)$ always holds. The reverse implication is true if edge-labels are invariant and the symbol-relationship is preserved, i.e. if $G(R_1)$ and $G(R_2)$ are isomorphic w.r.t. $\psi : L_1 \to L_2$, then already $\psi(\mathcal{V}_1) = \mathcal{V}_2$ (and automatically $\psi(\mathcal{F}_1) = \mathcal{F}_2$).

Proof. $(i) \implies (ii)$: Fix a global TRS isomorphism $\phi : (\mathcal{F}_1 \cup \mathcal{V}_1) \to (\mathcal{F}_2 \cup \mathcal{V}_2)$. Moreover, let $\mathcal{R}_1 = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$ and $\mathcal{R}_2 = \{\ell'_1 \to r'_1, \ldots, \ell'_n \to r'_n\}$ be ordered with respect to ϕ, i.e. $\phi_i(\ell_i) \to \phi(r_i) = \ell'_i \to r'_i$ for each $1 \leq i \leq n$. Then $\psi = \phi \sqcup \mathrm{id}_{\mathbb{N} \cup \{T\}}$ is well-defined and bijective. Note that graph isomorphisms are uniquely determined on connected components. Let $T_i = \mathrm{Tree}(\ell_i \to r_i)$ and $T'_i = \mathrm{Tree}(\ell'_i \to r'_i)$. To define an OOLDG isomorphism $\phi : V_1 \to V_2$ it thus suffices to consider restrictions $\phi'_i : V(T_i) \to V(T'_i)$ on connected components $T_i = \mathrm{Tree}(\ell_i \to r_i)$ and $T'_i = \mathrm{Tree}(\ell'_i \to r'_i)$. Map ϕ' is then given by $\bigsqcup_{i=1}^{n} \phi'_i$. If v_i and v'_i are the roots of T_i and T'_i respectively, set $\phi'_i(v_i) = v'_i$. Consequently, ϕ_i can further be broken down into maps $\phi^l_i : V(\mathrm{Tree}(\ell_i)) \to V(\mathrm{Tree}(\ell'_i))$ and $\phi^r_i : V(\mathrm{Tree}(r_i)) \to V(\mathrm{Tree}(r'_i))$, such that $\phi'_i = \{v \mapsto v'\} \sqcup \phi^l_i \sqcup \phi^r_i$. It is left to show that if $\phi(t) = t'$ for terms $t \in T(\mathcal{F}_1, \mathcal{V}_1)$ and $t' \in T(\mathcal{F}_2, \mathcal{V}_2)$, $\mathrm{Tree}(t)$ and $\mathrm{Tree}(t')$ are indeed isomorphic. This can be done by structural induction. Denote with ϕ_t an isomorphism between $\mathrm{Tree}(t)$ and $\mathrm{Tree}(\phi(t))$.

In the base case, either $t = x$ or $t = c$ for variable symbol $x \in \mathcal{V}_1$ or constant $c \in \mathcal{F}_1$. Thus $\phi(t) = \phi(x)$ or $\phi(t) = \phi(c)$, and $\mathrm{Tree}(t)$, $\mathrm{Tree}(\phi(t))$ only consist of one vertex v, v' respectively, with corresponding label. Then $\phi_t = \{v \mapsto v'\}$ is an isomorphism. Now let $t = f(t_1, \ldots, t_n)$ for some function symbol $f \in \mathcal{F}_1$ and terms $t_1, \ldots, t_n$. Suppose $\mathrm{Tree}(t_i) \cong \mathrm{Tree}(\phi(t_i))$ given by isomorphisms ϕ_{t_i}, $1 \leq i \leq n$, according to induction hypothesis. We have $\text{Tree}(t) = \text{Join}(v, f, \text{Tree}(t_1), \ldots, \text{Tree}(t_n))$ and

$$\text{Tree}(\phi(t)) = \text{Join}(v', \phi(f), \text{Tree}(\phi(t_1)), \ldots, \text{Tree}(\phi(t_n)))$$

due to $\phi(t) = \phi(f)\big(\phi(t_1), \ldots, \phi(t_n)\big)$. Then $\phi_t = \{v \mapsto v'\} \sqcup \bigsqcup_{i=1}^{n} \phi_{t_i}$ is a well-defined isomorphism between the term trees. Note that the gradually constructed isomorphism is indeed edge-label-invariant and preserves symbol-relationship, i.e. $(i) \iff (ii)$ under tightened label-assumptions.

$(ii) \implies (i)$: Both, edge-label invariance and symbol-relationship preservation are necessary and non-trivial requirements to ensure TRS isomorphism. For example, consider singleton term rewriting systems $R_1 = (\{f, c\}, \{x\}, \{f(x, c) \to f(x, c)\})$ and $R_2 = (\{f, c\}, \{x\}, \{f(c, x) \to f(c, x)\})$. The respective TRS-forests/term trees $G(R_1)$ and $G(R_2)$ are depicted in Figure 4.4. Clearly R_1 and R_2 cannot be isomorphic in any way due

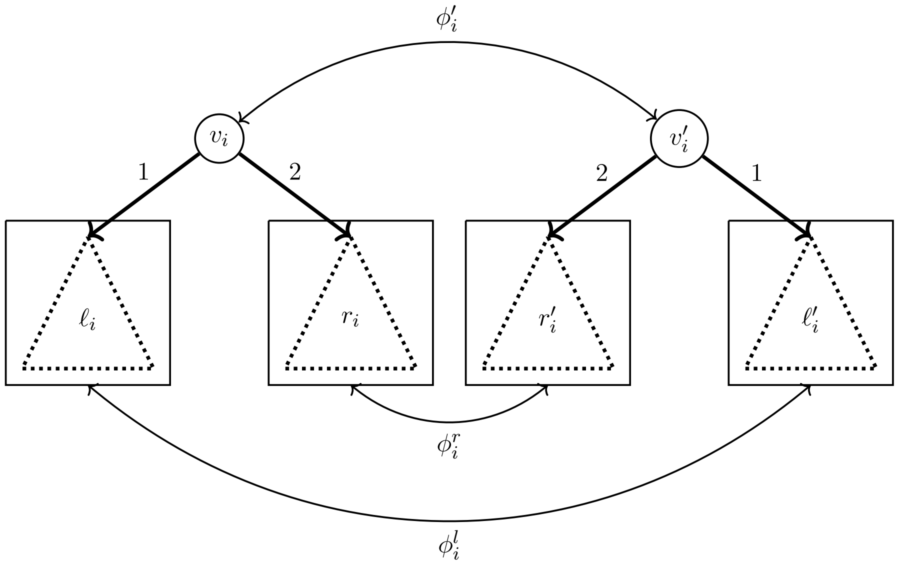


Figure 4.3: Reduction of isomorphisms on term trees to isomorphisms on term trees in the first part of the proof of Lemma 4.19

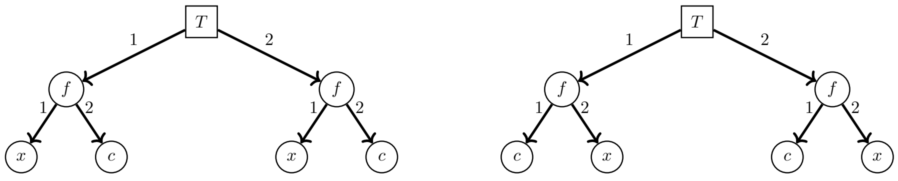


Figure 4.4: Isomorphic term trees of non-equivalent rewriting rules $f(x, c) \to f(x, c)$ and $f(c, x) \to f(c, x)$

to different ordering of x, c. In fact, any TRS isomorphism between R_1 and R_2 is equivalent to $f(x, c) \to f(x, c)$ and $f(c, x) \to f(c, x)$ being equivalent. On the contrary, $G(R_1) \cong G(R_2)$ w.r.t. one-to-one correspondences $\psi = \{x \mapsto c, c \mapsto x\} \sqcup \text{id}_{\{1,2,f,T\}}$ or

$\psi' = \{1 \mapsto 2, 2 \mapsto 1\} \sqcup \mathrm{id}_{\{x,c,f,T\}}$. The former isomorphism is edge-invariant but not symbol-relationship preserving, the latter exactly the opposite.
Now suppose $\phi' : V_1 \to V_2$ is an OOLDG isomorphism between $G(R_1)$ and $G(R_2)$ w.r.t. $\psi : L_1 \to L_2$, such that $\psi|_{\mathbb{N}} = \mathrm{id}_{\mathbb{N}}$ and $\psi(\mathcal{V}_1) = \mathcal{V}_2$. Then $\psi(T) = T$ due to ϕ' mapping root vertices onto root vertices. Moreover, $|\mathcal{R}_1| = |\operatorname{lab}_1^{-1}(\{T\})| = |\operatorname{lab}_2^{-1}(\{T\})| = |\mathcal{R}_2|$ and thus there is an one-to-one correspondence between term trees in $G(R_1)$ and in $G(R_2)$. Set $\phi = \psi|_{\mathcal{F}_1 \cup \mathcal{V}_1}$, well-defined and bijective. Note $\mathrm{ar}_1(f) = \mathrm{outdeg}(v)$ for each vertex $v \in V_1$ if $\mathrm{lab}_1(v) = f \in \mathcal{F}_1$, and therefore, since ϕ' preserves outward degrees, $\mathrm{ar}_1(f) = \mathrm{ar}_2(\phi(f))$ for any function symbol $f \in \mathcal{F}_1$. Map ϕ is indeed a term homomorphism due to symbol-relationship preserving property of label set function ψ. We will show that if $\phi'(\mathrm{Tree}(\ell \to r)) = \mathrm{Tree}(\ell' \to r')$ for term trees $\mathrm{Tree}(\ell \to r)$, $\ell \to r \in \mathcal{R}_1$ and $\mathrm{Tree}(\ell' \to r')$, $\ell' \to r' \in \mathcal{R}_2$, then already $\phi(\ell) \to \phi(r) = \ell' \to r'$. Isomorphism ϕ' necessarily maps $[\ell \to r]$ onto $[\ell' \to r']$ and thus, due to invariance of edge-labels, it suffices to check that $\phi(t) = t'$ if $\mathrm{Tree}(t)$ and $\mathrm{Tree}(t')$ are isomorphic under ϕ', for any terms $t \in T(\mathcal{F}_1, \mathcal{V}_1)$ and $t' \in T(\mathcal{F}_2, \mathcal{V}_2)$. This can be shown by structural induction over t. In the base case, $t = x$ for some variable symbol $x \in \mathcal{V}_1$, i.e. $\mathrm{Tree}(t)$ only consists of one vertex v. Then $\mathrm{Tree}(t')$ is necessarily a single vertex v' and due to $\psi(\mathcal{V}_1) = \mathcal{V}_2$, $\mathrm{lab}_2(v') \in \mathcal{V}_2$, i.e. $\phi(t) = \phi(x) = \psi(\mathrm{lab}_1(v)) = \mathrm{lab}_2(v') = t'$. Analogously, if $t = c$ for some constant $c \in \mathcal{F}_1$. Now suppose t is of the form $t = f(t_1, \dots, t_n)$ for some n-ary function symbol $f \in \mathcal{F}_1$ and terms $t_1, \dots, t_n$, i.e. $\mathrm{Tree}(t) = \mathrm{Join}(v, f, T_1, \dots, T_n)$, where $T_i = \mathrm{Tree}(t_i)$, $1 \le i \le n$. Then $\mathrm{Tree}(t') = \mathrm{Join}(v', f', T_1', \dots, T_n')$, $t' = f'(t_1', \dots, t_n')$, where $f' \in \mathcal{F}_2$ is an n-ary function symbol and $t_1', \dots, t_n'$ are terms such that $T_i' = \mathrm{Tree}(t_i')$, $1 \le i \le n$. Consequently, $f' = \psi(f) = \phi(f)$ and T_i isomorphic to T_i' for any $i \le i \le n$ due to assumption on ϕ and ψ. By induction hypothesis, $\phi(t_i) = t_i'$ and thus $\phi(t) = \phi(f(t_1, \dots, t_n)) = \phi(f)(t_1', \dots, t_n') = f'(t_1', \dots, t_n') = t'$. ■

4.4 GI-Completeness of Global TRS Isomorphisms

The goal of this section is to prove the following theorem:

Theorem 4.20 (**GI**-Completeness of Global TRS Isomorphisms). *The following sets are polynomially equivalent:*

(*i*) $\mathbf{GI} = \{(G_1, G_2) : G_1, G_2$ *isomorphic graphs*$\}$

(*ii*) $\mathbf{GVE} = \{(R_1, R_2) : R_1, R_2$ $\mathcal{V}$*-globally isomorphic TRS*$\}$

(*iii*) $\mathbf{GFE} = \{(R_1, R_2) : R_1, R_2$ $\mathcal{F}$*-globally isomorphic TRS*$\}$

(*iv*) $\mathbf{GE} = \{(R_1, R_2) : R_1, R_2$ *globally isomorphic TRS*$\}$

(*v*) $\mathbf{FSE} = \{(R_1, R_2) : R_1, R_2$ $\mathcal{F}$*-standard isomorphic TRS*$\}$

(*vi*) $\mathbf{VSE} = \{(R_1, R_2) : R_1, R_2$ $\mathcal{V}$*-standard isomorphic TRS*$\}$

(*vii*) $\mathbf{STS} = \{(R_1, R_2) : R_1, R_2$ *globally isomorphic STS*$\}$

Proof. Follows immediately by chain of arguments as depicted in Figure 4.5. ■

$$\begin{tikzcd}
(vi)\ \mathbf{VSE} \arrow[d, Rightarrow, "\text{Proposition 4.22}"] & (v)\ \mathbf{FSE} \arrow[d, Rightarrow, "\text{Proposition 4.21}"] & (vii)\ \mathbf{STS} \arrow[l, Rightarrow, "\text{Definition 4.9}"'] \arrow[d, Rightarrow, "\text{Definition 4.9}"] \\
(ii)\ \mathbf{GVE} \arrow[dr, Rightarrow, "\text{Proposition 4.22}"] & (iii)\ \mathbf{GFE} \arrow[d, Rightarrow, "\text{Proposition 4.21}"] & (iv)\ \mathbf{GE} \arrow[dl, Rightarrow, "\text{Proposition 4.23}"'] \\
(i)\ \mathbf{GI} \arrow[r, Leftrightarrow, "\text{Theorem 2.10}"] \arrow[dr, Leftrightarrow, "\text{Theorem 2.10}"] & \mathbf{StrOOLDG} & \\
 & \mathbf{DG} \arrow[uuul, Rightarrow, bend left=60, "\text{Proposition 4.24}"] \arrow[uuur, Rightarrow, bend right=60, "\text{Proposition 4.24}"'] &
\end{tikzcd}$$

Figure 4.5: Proof of Theorem 4.20

Proofs of implications $(ii) \implies (i)$, $(iii) \implies (i)$ and $(iv) \implies (i)$ will follow the same basic pattern. For fix term rewriting system R, we start by constructing TRS-forest $G(R)$. We then apply pointer techniques, similar to Proposition 2.18, where we reduced LDG isomorphism to strong LDG isomorphism, by introducing new vertices encoding only a subset of vertex-labels and unambiguously separating $\mathcal{V}$-/$\mathcal{F}$-labels. This forces a strong isomorphism between modified TRS-forests to be equivalent to an isomorphism between original TRS-forests, where edge-labels and some vertex-labels are invariant. The different claims then follow by Lemma 4.19.

Proposition 4.21 ($(v) \implies (iii) \implies (i)$). **FSE $\preceq^{\mathbf{P}}$ GFE $\preceq^{\mathbf{P}}$ GI**

Proof. We start by giving a polynomial reduction from **FSE** to **GFE**. Fix term rewriting systems $R_i = (\mathcal{F}_i, \mathcal{V}_i, \mathcal{R}_i)$ in $\mathcal{V}$-normal form and consider $\mathcal{V}$-local templates $R'_i = (\mathcal{F}_i, \mathcal{V}_S, \phi_{\mathcal{V}_i}(\mathcal{R}_i))$ as given by proof of Lemma 4.13 and Algorithm 4.1, $i = 1, 2$. This templates can be computed in polynomial time. We claim that R_1 and R_2 are $\mathcal{F}$-standard isomorphic if and only if R'_1 and R'_2 are $\mathcal{F}$-globally isomorphic. Recall that by

Corollary 4.15, R_i and R'_i are $\mathcal{V}$-locally isomorphic. First, note that due to $\mathcal{V}$-normal form, $|\mathcal{R}_i| = |\phi_{\mathcal{V}_i}(\mathcal{R}_i)| = n_i \in \mathbb{N}$ for $i = 1, 2$, and in every case $n = n_1 = n_2$. Suppose $\phi = (\phi_1, \ldots, \phi_n)$ is a $\mathcal{F}$-standard isomorphism between R_1 and R_2. Initial definition $R_i = (\mathcal{F}_i, \mathcal{V}_i, \mathcal{R}_i)$ is justified due to $|\mathcal{V}_1| = |\mathcal{V}_2|$, and thus both R'_1 and R'_2 possessing the same variable symbol set. Then $\phi' = \phi_{\mathcal{V}_2} \circ \phi \circ \phi_{\mathcal{V}_1}^{-1}$ is a $\mathcal{F}$-global isomorphism between R'_1 and R'_2. Indeed, for fix $i = 1, \ldots, n$, $\phi'_i = (\phi_{\mathcal{V}_2})_i \circ \phi_i \circ (\phi_{\mathcal{V}_1})_i^{-1}$, bijective,

$$\phi'_i(\mathcal{V}_S) = \big((\phi_{\mathcal{V}_2})_i \circ \phi_i \circ (\phi_{\mathcal{V}_1})_i^{-1}\big)(\mathcal{V}_S) = \big((\phi_{\mathcal{V}_2})_i \circ \phi_i\big)(\mathcal{V}_1) = (\phi_{\mathcal{V}_2})_i(\mathcal{V}_2) = \mathcal{V}_S$$

and $\phi'_i|_{\mathcal{F}_1} = \phi_i|_{\mathcal{F}_1}$, due to $(\phi_{\mathcal{V}_1})_i^{-1}|_{\mathcal{F}_1} = \mathrm{id}_{\mathcal{F}_1}$, $(\phi_{\mathcal{V}_2})_i|_{\mathcal{F}_2} = \mathrm{id}_{\mathcal{F}_2}$, as long as we assume $\mathcal{R}_1$, $\phi_{\mathcal{V}_1}(\mathcal{R}_1)$ and $\mathcal{R}_2$ to be ordered in such a way that we can compose those functions in the desired way. Moreover, ϕ'_i is $\mathcal{V}$-invariant due to construction in Algorithm 4.1 and $\mathcal{V}$-equivalence of images of ith rewriting rule $\ell_i \to r_i \in \mathcal{R}_1$ under $\phi_i|_{\mathcal{F}_1} \sqcup \mathrm{id}_{\mathcal{V}_1}$ and ϕ_i. Then $\phi'(\phi_{\mathcal{V}_1}(\mathcal{R}_1)) = \big(\phi_{\mathcal{V}_2} \circ \phi \circ \phi_{\mathcal{V}_1}^{-1}\big)(\phi_{\mathcal{V}_1}(\mathcal{R}_1)) = \phi_{\mathcal{V}_2}(\mathcal{R}_2)$ and we are done. For given $\mathcal{F}$-global isomorphism ϕ' between R'_1 and R'_2 one shows the same way that $\phi := \phi_{\mathcal{V}_2}^{-1} \circ \phi' \circ \phi_{\mathcal{V}_1}$ defines a $\mathcal{F}$-standard isomorphism between R_1 and R_2. That is to say, diagram (4.3) is commutative. Indeed, $\phi_i|_{\mathcal{F}_1} = \phi'_i|_{\mathcal{F}_1}$ due to $\mathcal{F}$-invariance of $\phi_{\mathcal{V}_1}^{-1}$ and $\phi_{\mathcal{V}_2}$, and therefore $\phi_1|_{\mathcal{F}_1} = \ldots = \phi_n|_{\mathcal{F}_1}$.

$$\begin{tikzcd}
(\mathcal{F}_1, \mathcal{V}_1, \mathcal{R}_1) \arrow[rr, "\phi"] \arrow[d, shift left, "\phi_{\mathcal{V}_1}"] & & (\mathcal{F}_2, \mathcal{V}_2, \mathcal{R}_2) \arrow[d, shift left, "\phi_{\mathcal{V}_2}"] \\
\big(\mathcal{F}_1, \mathcal{V}_S, \phi_{\mathcal{V}_1}(\mathcal{R}_1)\big) \arrow[rr, "\phi'"] \arrow[u, shift left, "\phi_{\mathcal{V}_1}^{-1}"] & & \big(\mathcal{F}_2, \mathcal{V}_S, \phi_{\mathcal{V}_2}(\mathcal{R}_2)\big) \arrow[u, shift left, "\phi_{\mathcal{V}_2}^{-1}"]
\end{tikzcd} \tag{4.3}$$

Now reduce **GFE** to **GI**. Recall Subsection 2.3.2 about pointer techniques. We will encode $\mathcal{F}$-labelled vertices via pointer and then use strong isomorphism on resulting OOLDGs. Let $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ be a term rewriting system. An OOLDG $\mathrm{Graph}_{\mathcal{F}}(R) = (V', E', L' = \mathcal{V} \cup \{F, T\} \cup \mathbb{N}_0, \mathrm{lab}')$ then can be constructed as stated below.

(1) Construct $G(R) = (V, E, L = \mathcal{F} \cup \mathcal{V} \cup \{T\} \cup \mathbb{N}, \mathrm{lab})$.

(2) For each function symbol $f \in \mathcal{F}$, introduce fresh F-labelled vertex v_f.

(3) Encode $\mathcal{F}$-labels with new edges $(v, v_f, 0)$, where v is a vertex with label $f \in \mathcal{F}$.

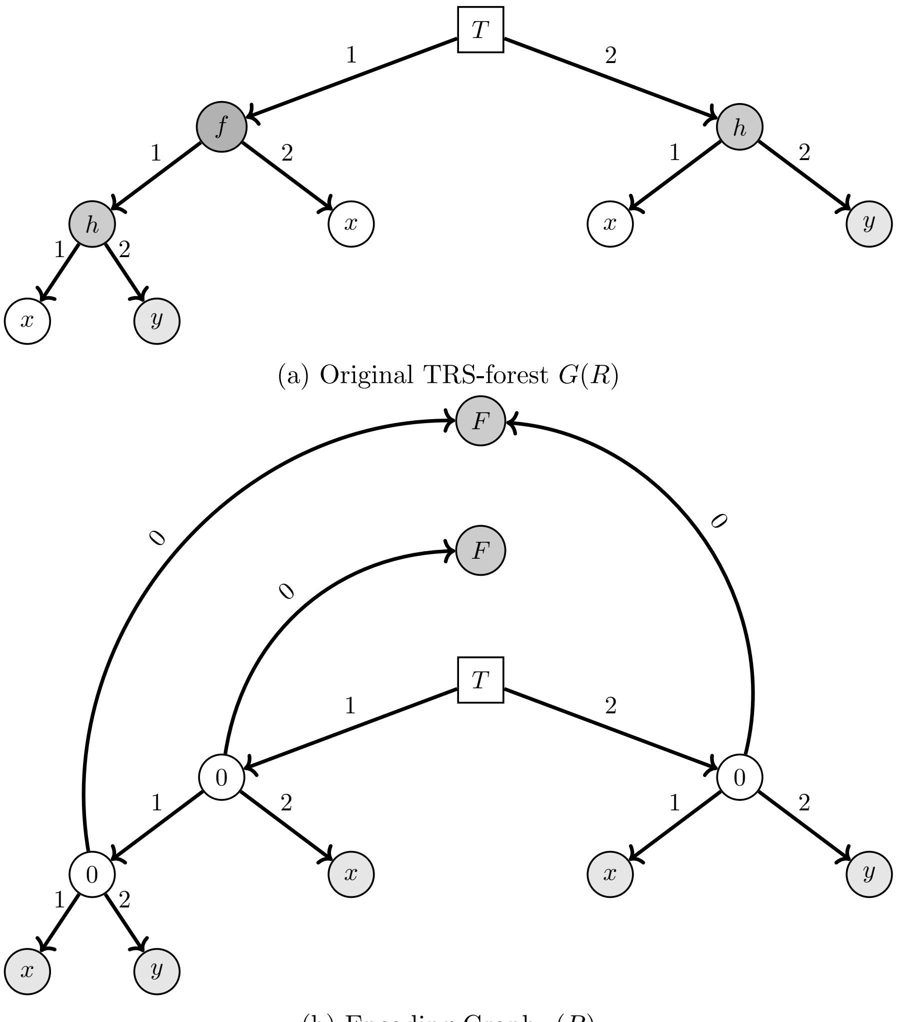
(4) Replace all $\mathcal{F}$-labels with uniform label 0, i.e. relabel every $\mathcal{F}$-labelled vertex.

That is, $V' = V \cup \{v_f : f \in \mathcal{F}\}$, $E' = E \cup \{(v, v_f, 0) : v \in V, \mathrm{lab}(v) = f \in \mathcal{F}\}$ and labelling function $\mathrm{lab}' : V' \to \mathcal{V} \cup \{F, T\} \cup \mathbb{N}_0$,

$$\mathrm{lab}'(v) = \begin{cases} F & \text{if } v = v_f \text{ for some } f \in \mathcal{F} \\ 0 & \text{if } \mathrm{lab}(v) \in \mathcal{V} \\ \mathrm{lab}(v) & \text{otherwise} \end{cases}$$

Step (1) is done in polynomial time as mentioned at the beginning of the section. Step

(2) is in $\mathcal{O}(|\mathcal{F}|)$ and steps (3)/(4) are done simultaneously in at most $\mathcal{O}(s|\mathcal{R}|\log|\mathcal{F}|)$ by traversing each tree breadth-first and checking whether it is $\mathcal{F}$-labelled or not. Now, two term rewriting systems R_1, R_2 are $\mathcal{F}$-globally isomorphic if and only if OOLDGs $G_1 = \mathrm{Graph}_{\mathcal{F}}(R_1)$ and $G_2 = \mathrm{Graph}_{\mathcal{F}}(R_2)$ are strongly isomorphic. We show that strong isomorphism between G_1 and G_2 is equivalent to an isomorphism between $G(R_1)$ and $G(R_2)$, where edge- and $\mathcal{V}$-labels are invariant. By Lemma 4.19 this is equivalent to global-isomorphism between R_1 and R_2, where variable symbols are invariant, but this is exactly $\mathcal{F}$-global isomorphism.



(a) Original TRS-forest $G(R)$

(b) Encoding $\mathrm{Graph}_{\mathcal{F}}(R)$

Figure 4.6: Example of encoding $\mathrm{Graph}_{\mathcal{F}}$ in the proof of Proposition 4.21 for singleton TRS $\mathcal{R} = \{f(h(x,y),x) \to h(x,y)\}$

Let $\phi' : V_1' \to V_2'$ be a strong isomorphism between G_1 and G_2. Then $|\mathcal{F}_1| = |\,\mathrm{lab}_1'^{-1}(\{F\})| = |\,\mathrm{lab}_2'^{-1}(\{F\})| = |\mathcal{F}_2|$, $|V_1| = |V_1'| - |\mathcal{F}_1| = |V_2'| - |\mathcal{F}_2| = |V_2|$, and thus $\psi : L_1 \to L_2$,

$\psi = \{f \mapsto f' : f \in \mathcal{F}, v_{f'} = \phi'(v_f)\} \sqcup \mathrm{id}_{\{T\} \cup \mathcal{V}_1 \cup \mathbb{N}}$ and $\phi = \phi'|_{V_1} : V_1 \to V_2$, are well-defined and bijective maps. Therefore $G(R_1)$ and $G(R_2)$ are isomorphic w.r.t ϕ and ψ if we consider $\mathcal{F}$-labelled vertices to be unlabelled, and we just need to check label property on this particular vertex subset. Fix such a $\mathcal{F}$-labelled vertex $v \in V_1$, $\mathrm{lab}_1(v) = f \in \mathcal{F}_1$ with unique 0-labelled outgoing edge $(v, v_f, 0) \in E_1'$. For $\phi(v)$, this corresponds to $(\phi(v), \phi'(v_f), 0) \in E_2'$, but $\phi'(v_f) = v_{\psi(f)}$, which is already equivalent to $\psi(\mathrm{lab}_1(v)) = \mathrm{lab}_2(\phi(v))$.
On the other hand, consider an isomorphism $\phi : V_1 \to V_2$ w.r.t. $\psi : L_1 \to L_2$, where $\psi|_{\mathbb{N} \cup \mathcal{V}_1} = \mathrm{id}_{\mathbb{N} \cup \mathcal{V}_1}$ and in particular, $\psi(T) = T$ due to ϕ necessarily mapping roots onto roots. Then $\psi|_{\mathcal{F}_1} : \mathcal{F}_1 \to \mathcal{F}_2$ is bijective and can be used to extend ϕ to all of V_1' via $\phi' = \phi \sqcup \{v_f \mapsto v_{\psi(f)} : f \in \mathcal{F}_1\}$. Map ϕ' is clearly well-defined and bijective, and due to construction induces strong an isomorphism between $G(R_1)$ and $G(R_2)$, considered as appropriately labelled subgraphs of G_1 and G_2 respectively. Indeed, non-invariant $\mathcal{F}$-labels were purged in favour of uniform 0-labels. All v_f-vertices are F-labelled, so we just have to check isomorphism property on newly introduced 0-labelled edges. For $(v, v_f, 0) \in E_1$, $\mathrm{lab}_1(v) = f$, and thus $(\phi(v), \phi'(v_f), 0) \in E_2$, due to $\phi'(v_f) = v_{\psi(f)}$ and $\mathrm{lab}_2(\phi(v)) = \psi(\mathrm{lab}_1(v))$. We are done, since by construction

$$|E_1' \setminus E_1| = |\,\mathrm{lab}_1'^{-1}(\{0\})| = |\,\mathrm{lab}_1^{-1}(\mathcal{F}_1)| = |\,\mathrm{lab}_2^{-1}(\mathcal{F}_2)| = |\,\mathrm{lab}_2'^{-1}(\{0\})| = |E_2' \setminus E_2|.$$

■

Proposition 4.22 ($(vi) \implies (ii) \implies (i)$). **VSE** $\preceq^{\mathbf{P}}$ **GVE** $\preceq^{\mathbf{P}}$ **GI**.

Proof. A polynomial reduction from **VSE** to **GVE** is given the following way: Two term rewriting systems $R_i = (\mathcal{F}_i, \mathcal{V}_i, \mathcal{R}_i)$ in $\mathcal{F}$-normal form, $i = 1, 2$, are $\mathcal{V}$-standard isomorphic if and only if their $\mathcal{F}$-local templates $R_i' = (\mathcal{F}_S, \mathcal{V}_i, \phi_{\mathcal{F}_i}(\mathcal{R}_i))$ as given by proof of Lemma 4.13 and Algorithm 4.2, $i = 1, 2$, are $\mathcal{V}$-globally isomorphic. This is proven analogously to Proposition 4.21 and equivalent to the fact that diagram (4.4) is commutative.

$$\begin{tikzcd}
(\mathcal{F}_1, \mathcal{V}_1, \mathcal{R}_1) \arrow[r, "\phi"] \arrow[d, shift left, "\phi_{\mathcal{F}_1}"] & (\mathcal{F}_2, \mathcal{V}_2, \mathcal{R}_2) \arrow[d, shift left, "\phi_{\mathcal{F}_2}"] \\
(\mathcal{F}_S, \mathcal{V}_1, \phi_{\mathcal{F}_1}(\mathcal{R}_1)) \arrow[r, "\phi'"] \arrow[u, shift left, "\phi_{\mathcal{F}_1}^{-1}"] & (\mathcal{F}_S, \mathcal{V}_2, \phi_{\mathcal{F}_2}(\mathcal{R}_2)) \arrow[u, shift left, "\phi_{\mathcal{F}_2}^{-1}"]
\end{tikzcd} \tag{4.4}$$

We only have to argue that both $\mathcal{F}$-local templates posses the same function symbol set if initial term rewriting systems R_i are $\mathcal{V}$-standard isomorphic, but this immediately clear due to existence of term homomorphism between $T(\mathcal{F}_1, \mathcal{V}_1)$ and $T(\mathcal{F}_2, \mathcal{V}_2)$. Indeed, in this case $\{\mathrm{ar}_1(f) : f \in \mathcal{F}_1\} = \{\mathrm{ar}_2(f) : f \in \mathcal{F}_2\}$ and $|\,\mathrm{ar}_1^{-1}(l)| = |\,\mathrm{ar}_1^{-1}(l)|$ for any $l \in \mathbb{N}_0$ (refer to proof of Lemma 4.13).
Next, let $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ be a term rewriting system. Again we will use pointers, this

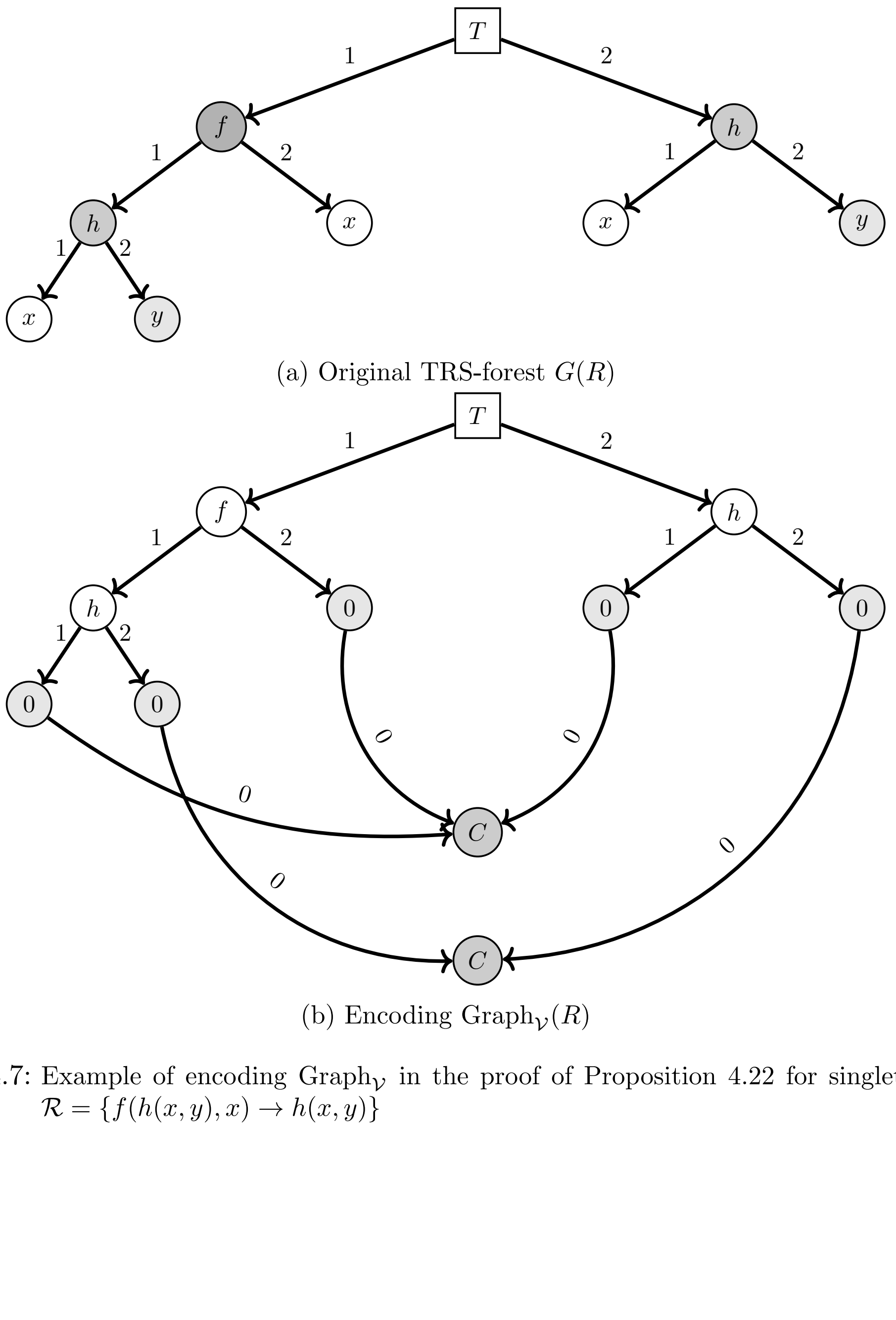
time to encode $\mathcal{V}$-labelled vertices. An OOLDG $\text{Graph}_{\mathcal{V}}(R) = (V', E', L' = \mathcal{V} \cup \{C, T\} \cup \mathbb{N}_0, \text{lab}')$ then can be constructed as stated below.

(1) Construct $G(R) = (V, E, L = \mathcal{F} \cup \mathcal{V} \cup \{T\} \cup \mathbb{N}, \text{lab})$.

(2) For each variable symbol $x \in \mathcal{V}$, introduce fresh C-labelled vertex v_x.

(3) Encode $\mathcal{V}$-labels with new edges $(v, v_x, 0)$, where v is a vertex with label $x \in \mathcal{V}$.

(4) Replace all $\mathcal{V}$-labels with uniform label 0, i.e. relabel every $\mathcal{V}$-labelled vertex.



(a) Original TRS-forest $G(R)$

(b) Encoding $\text{Graph}_{\mathcal{V}}(R)$

Figure 4.7: Example of encoding $\text{Graph}_{\mathcal{V}}$ in the proof of Proposition 4.22 for singleton TRS $\mathcal{R} = \{f(h(x, y), x) \to h(x, y)\}$

That is, $V' = V \cup \{v_x : x \in \mathcal{V}\}$, $E' = E \cup \{(v, v_x, 0) : v \in V, \mathrm{lab}(v) = x \in \mathcal{V}\}$ and labelling function $\mathrm{lab}' : V' \to \mathcal{V} \cup \{C, T\} \cup \mathbb{N}_0$,

$$\mathrm{lab}'(v) = \begin{cases} C & \text{if } v = v_x \text{ for some } x \in \mathcal{V} \\ 0 & \text{if } \mathrm{lab}(v) \in \mathcal{V} \\ \mathrm{lab}(v) & \text{otherwise} \end{cases}$$

Again, step (1) is done in polynomial time as mentioned at the beginning of the section. Step (2) is in $\mathcal{O}(|\mathcal{V}|)$ and steps (3)/(4) are done simultaneously in at most $\mathcal{O}(s|\mathcal{R}| \log |\mathcal{V}|)$ by traversing each tree breadth-first and checking whether it is $\mathcal{V}$-labelled or not. Strong isomorphism between $\mathrm{Graph}_{\mathcal{V}}(R_1)$ and $\mathrm{Graph}_{\mathcal{V}}(R_2)$ then is equivalent to an isomorphism between $G(R_1)$ and $G(R_2)$, where edge- and $\mathcal{F}$-labels are invariant. This is shown the same way as in Proposition 4.21, just switch the notions of $\mathcal{F}$ and $\mathcal{V}$ and labels F and C. By Lemma 4.19 this is then equivalent to a global isomorphism between R_1 and R_2 with invariant function symbols, but this is exactly $\mathcal{V}$-global isomorphism. ■

Proposition 4.23 ($(iv) \implies (i)$). $\mathbf{GE} \preceq^{\mathbf{P}} \mathbf{GI}$.

Proof. Let $R = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ be a term rewriting system. Contrary to the $\mathcal{F}/\mathcal{V}$-global case, we are not required to ensure invariance of one of the symbol sets. According to Lemma 4.19, we cannot outright use $G(R)$ and then apply OOLDG isomorphism, even though this appears to be the easiest solution on paper. A more nuanced encoding Graph of a R into an OOLDG $\mathrm{Graph}(R) = (V', E', L' = \mathbb{N}_0 \cup \{z, g, F, C\})$ can thus be accomplished by combining both encodings from Proposition 4.21 and Proposition 4.22 into one:

(1) Construct $G(R) = (V, E, L = \mathcal{F} \cup \mathcal{V} \cup \{T\} \cup \mathbb{N}, \mathrm{lab})$.

(2) For each variable symbol $x \in \mathcal{V}$, introduce fresh C-labelled vertex v_x.

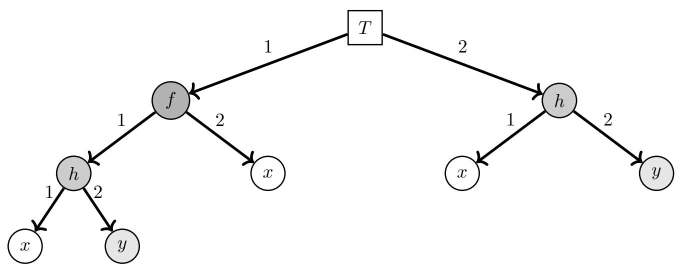
(3) Encode $\mathcal{V}$-labels with new edges $(v, v_x, 0)$, where v is a vertex with label $x \in \mathcal{V}$.

(4) Replace all $\mathcal{V}$-labels with fresh, uniform label z, i.e. relabel every $\mathcal{V}$-labelled vertex.

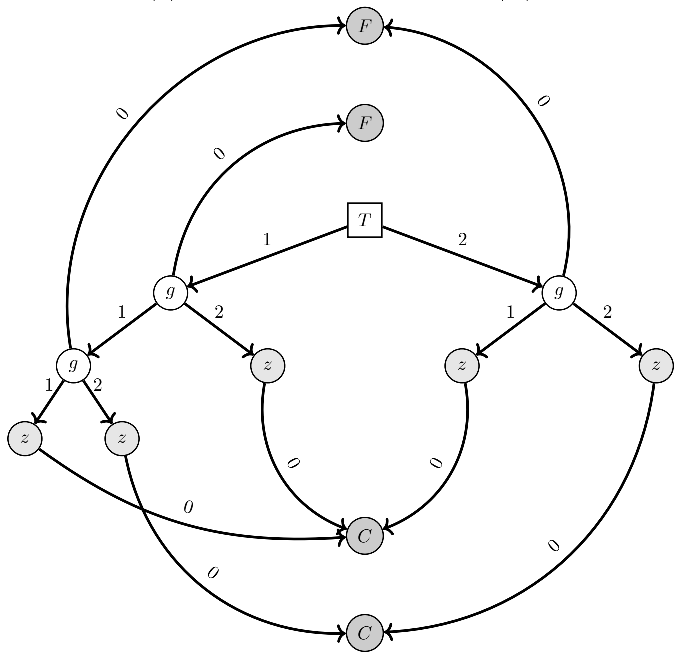
(5) For each function symbol $f \in \mathcal{F}$, introduce fresh F-labelled vertex v_f.

(6) Encode $\mathcal{F}$-labels with new edges $(v, v_f, 0)$, where v is a vertex with label $f \in \mathcal{F}$.

(7) Replace all $\mathcal{F}$-labels with fresh, uniform label g, i.e. relabel every $\mathcal{F}$-labelled vertex.



(a) Original OOLDG-forest $G(R)$



(b) Encoding Graph(R)

Figure 4.8: Example of encoding Graph in the proof of Proposition 4.23 for singleton TRS $\mathcal{R} = \{f(h(x,y),x) \to h(x,y)\}$

That is, $V' = V \cup \{v_x : x \in \mathcal{V}\} \cup \{v_f : f \in \mathcal{F}\}$, $E' = E \cup \{(v, v_x, 0) : v \in V, \mathrm{lab}(v) = x \in \mathcal{V}\} \cup \{(v, v_f, 0) : v \in V, \mathrm{lab}(v) = f \in \mathcal{F}\}$ and labelling function $\mathrm{lab}' : V' \to \mathbb{N}_0 \cup \{z, g, F, C\}$,

$$\mathrm{lab}'(v) = \begin{cases} C & \text{if } v = v_x \text{ for some } x \in \mathcal{V} \\ F & \text{if } v = v_f \text{ for some } f \in \mathcal{F} \\ z & \text{if } \mathrm{lab}(v) \in \mathcal{V} \\ g & \text{if } \mathrm{lab}(v) \in \mathcal{F} \\ \mathrm{lab}(v) & \text{otherwise} \end{cases}$$

Step (1) is done in polynomial time as mentioned at the beginning of the section. Steps (2) and (5) are in $\mathcal{O}(|\mathcal{F}| + |\mathcal{V}|)$ and Steps (3)/(4) and (6)/(7) are done simultaneously in at most $\mathcal{O}(s|\mathcal{R}|\log(|\mathcal{F}||\mathcal{V}|))$ by traversing each tree breadth-first and checking

whether it is $\mathcal{F}$-labelled or not. Then strong isomorphism between $G_1 = \mathrm{Graph}(R_1)$ and $G_2 = \mathrm{Graph}(R_2)$ is equivalent to edge-label invariant and symbol-relationship preserving isomorphism between $G(R_1)$ and $G(R_2)$, and the initial claim is a direct consequence of Lemma 4.19. Note that this was almost shown in Proposition 2.18 if we disregard edge-labels in the construction, i.e. without introducing vertex set $V^{(1)}$ and edge sets $E^{(1)}, E^{(3)}$ (recall notation). We accomplish label-type differentiation by introducing two labels to encode additionally introduced vertices. Relabelling of original vertices, also with two different labels, simplifies the proof.
First, fix a strong isomorphism $\phi' : V_1' \to V_2'$. Then $\psi : L_1 \to L_2$,

$$\psi = \mathrm{id}_{\mathbb{N} \cup \{T\}} \sqcup \{f \mapsto f' : f \in \mathcal{F}_1, \phi'(v_f) = v_{f'}\} \sqcup \{x \mapsto x' : x \in \mathcal{V}_1, \phi'(v_x) = v_{x'}\}$$

is well-defined and bijective due to label-preserving property, i.e. $|\{v_f : f \in \mathcal{F}_1\}| = |\operatorname{lab}_1'^{-1}(\{F\})| = |\operatorname{lab}_2'^{-1}(\{F\})| = |\{v_f : f \in \mathcal{F}_2\}|$, analogously for vertices v_x, where x is an arbitrary variable symbol in either $\mathcal{V}_1$ or $\mathcal{V}_2$. Moreover, $V_i = \operatorname{lab}_i'^{-1}(\{z, g\})$, $i = 1, 2$, and we check that $\phi = \phi'|_{V_1} : V_1 \to V_2$ is indeed an isomorphism w.r.t. ψ. Note that in this case, ϕ already satisfies all assumptions of reverse implication in Lemma 4.19, since $\mathcal{V}_2 = \psi(\mathcal{V}_1)$ and $\operatorname{lab}_1(v) \in \mathcal{V}_1$ if and only if $\operatorname{lab}_1'(v) = z$.
By vertex set restriction, G_1 and G_2 are strongly isomorphic if we consider vertices to be unlabelled, with notable exception of T-labelled roots. Fix $\mathcal{F}$-labelled vertex $v \in V_1$, $\operatorname{lab}_1(v) = f$ with unique 0-labelled outgoing edge $(v, v_f, 0) \in E_2'$. Just as in Proposition 4.21, $\psi(\operatorname{lab}_1(v)) = \operatorname{lab}_2(\phi(v))$ by inspecting 0-labelled outgoing edge of $\phi(v)$. The case of $\mathcal{V}$-labelled vertices is shown analogously, i.e. as in Proposition 4.22, where we can distinguish the two cases by different labels z and g in $G(R_i)$. Consequently, G_1 and G_2 are isomorphic w.r.t. ψ.
On the other hand, let an isomorphism $\phi : V_1 \to V_2$ w.r.t. $\psi : L_1 \to L_2$ be given, where $\psi|_{\mathbb{N}} = \mathrm{id}_{\mathbb{N}}$ and $\psi(\mathcal{V}_1) = \mathcal{V}_2$. In particular $\psi(T) = T$, since ϕ maps roots onto roots. Extend ϕ to all of V_1' via

$$\phi' = \phi \sqcup \{v_f \mapsto v_{\psi(f)} : f \in \mathcal{F}_1\} \sqcup \{v_x \mapsto v_{\psi(x)} : x \in \mathcal{V}_1\},$$

well-defined and bijective due to $|\mathcal{F}_1| = |\mathcal{F}_2|$ and $|\mathcal{V}_1| = |\mathcal{V}_2|$. This again induces a strong isomorphism between $G(R_1)$ and $G(R_2)$, understood as appropriately relabelled subgraphs of G_1 and G_2. Indeed, if $\operatorname{lab}_1(v) \in \mathcal{F}_1$, $\operatorname{lab}_2(\phi(v)) = \psi(\operatorname{lab}_1(v)) \in \mathcal{F}_2$ and hence $\operatorname{lab}_1'(v) = g = \operatorname{lab}_2'(\phi(v))$, analogously for $\operatorname{lab}_1(v) \in \mathcal{V}_1$. It is left to show isomorphism property for newly introduced 0-labelled vertices. For fix non-root vertex $v \in V_1$ this is proven the same way as in Proposition 4.21 and thus in Proposition 4.22, where again we can distinguish the two cases $\mathcal{F}/\mathcal{V}$-labelled by different labels g and z in $G(R_i)$. ■

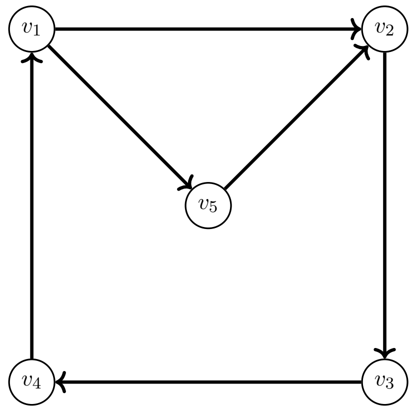
Proposition 4.24 ((i) $\implies$ (vi) and (i) $\implies$ (vii) $\implies$ (v)). **GI** $\preceq^{\mathbf{P}}$ **VSE** *and* **GI** $\preceq^{\mathbf{P}}$ **STS** $\preceq^{\mathbf{P}}$ **FSE**.

Proof. Let $G = (V, E)$ be an unlabelled directed graph. We define the encodings $R(G) = (\mathcal{F}, \mathcal{V}, \mathcal{R})$ and $R'(G) = (\mathcal{F}', \mathcal{V}', \mathcal{R}')$ into string rewriting systems and term rewriting systems respectively, the following way:

(1) $\mathcal{F} = \{f_v : v \in V\} \cup \{c\}$, $\mathcal{V} = \{x\}$, $\mathcal{R} = \{f_v(f_w(x)) \to c : (v,w) \in E\}$ and

(2) $\mathcal{F}' = \{f, c\}$, $\mathcal{V}' = \{x_v : v \in V\}$, $\mathcal{R}' = \{f(x_v, x_w) \to c : (v,w) \in E\}$.

Note that f, c, x are fixed for respective encoding, while function symbols f_v and variable symbols x_v depend on vertices $v \in V$. The concept is the same for both. Vertices are represented by either unique function or variable symbols, while edges are encoded by rewriting rules. See Figure 4.9 for an example. Now two unlabelled directed graphs G_1 and G_2 are isomorphic if and only if string rewriting systems $R(G_i) = R_i$, $i = 1,2$, are isomorphic or if and only if term rewriting systems $R'(G_i) = R'_i$,$i = 1,2$, are $\mathcal{V}$-standard isomorphic.
If $\phi : V_1 \to V_2$ is a graph isomorphism between G_1, G_2, one easily checks that $\phi' : \mathcal{F}_1 \to \mathcal{F}_2$, $\phi'(x) = x$, $\phi'(c) = c$ and $\phi'(f_v) = f_{\phi(v)}$ is a well-defined STS isomorphism, and $\psi' : (\mathcal{F}' \cup \mathcal{V}'_1) \to (\mathcal{F}' \cup \mathcal{V}'_2)$, $\psi'(x_v) = x_{\phi(v)}$ a well-defined $\mathcal{V}$-standard isomorphism.
On the other hand, first assume $\phi' : \mathcal{F}_1 \to \mathcal{F}_2$ is a STS isomorphism between R_1 and R_2. Since ϕ' preserves arity of function symbols, map $\phi : V_1 \to V_2$, so that $\phi'(f_v) = f_{\phi(v)}$, is well-defined and bijective due to $|V_1| = |\mathcal{F}_1 \setminus \{c\}| = |\mathcal{F}_2 \setminus \{c\}| = |V_2|$ and ϕ' invariant on $\{x, c\}$. Fix $(v,w) \in E_1$. Then $f_{\phi(v)}(f_{\phi(w)}(x)) \to c = \phi'(f_v)(\phi'(f_w)(x)) \to c \in \mathcal{R}_2$, which is equivalent to $(\phi(v), \phi(w)) \in E_2$. Since $|E_1| = |\mathcal{R}_1| = |\mathcal{R}_2| = |E_2|$, we are done.
Now assume $\psi' : (\mathcal{F}' \cup \mathcal{V}'_1) \to (\mathcal{F}' \cup \mathcal{V}'_2)$ is a $\mathcal{V}$-standard isomorphism, or equivalently, a $\mathcal{V}$-global isomorphism, between R'_1 and R'_2. Again, due to arity-preserving property of ψ', we have $\psi'(f) = f$ and $\psi'(c) = c$. Thus map $\phi : V_1 \to V_2$, so that $\psi'(x_v) = x_{\phi(v)}$ is well-defined and bijective due to $|V_1| = |\mathcal{V}'_1| = |\mathcal{V}'_2| = |V_2|$. Fix $(v,w) \in E$. Then analogously to the case before, $(\phi(v), \phi(w)) \in E_2$ and since $|E_1| = |\mathcal{R}'_1| = |\mathcal{R}'_2| = |E_2|$ we are done. ■



(a) Unlabelled directed graph G of order 5 and size 6

$$
\begin{aligned}
\mathcal{F} &= \{f_1, f_2, f_3, f_4, f_5, c\} \\
\mathcal{R} &= \{f_1(f_2(x)) \to c, f_1(f_5(x)) \to c, \\
&\quad f_2(f_3(x)) \to c, f_3(f_4(x)) \to c, \\
&\quad f_4(f_1(x)) \to c, f_5(f_2(x)) \to c\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}' &= \{x_1, x_2.x_3, x_4, x_5\} \\
\mathcal{R}' &= \{f(x_1, x_2) \to c, f(x_1, x_5) \to c, \\
&\quad f(x_2, x_3) \to c, f(x_3, x_4) \to c, \\
&\quad f(x_4, x_1) \to c, f(x_5, x_2) \to c\}
\end{aligned}
$$

(b) Encodings $R(G)$ and $R'(G)$

Figure 4.9: Example of encodings $R(G)$ and $R'(G)$ in the proof of Proposition 4.24
Function symbol f_i and variable symbol x_i respectively refer to vertex v_i

Algorithm 4.1: Computation of $\mathcal{V}$-local template

```
   input    : TRS $R = (\mathcal{F}, \mathcal{V}, \mathrm{ar}, \mathcal{R})$
   output   : $\phi_\mathcal{V} = (\phi_1, \ldots, \phi_n)$ family of $\mathcal{F}$-invariant term homomorphisms,
              canonical rewriting of variable symbols on a per rule basis
   runtime: $\mathcal{O}(s|\mathcal{R}||\mathcal{V}||\mathcal{F}| \log(|\mathcal{V}||\mathcal{F}|))$
   // $\mathcal{R} = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$
1  Let $K \leftarrow |\mathcal{V}|$
2  Initialise new variable symbol set $\mathcal{V}_S = \{x_1, \ldots, x_K\}$
3  for $j \leftarrow 1$ to $n$ do                                   // $\mathcal{O}(s|\mathcal{R}| \log |\mathcal{V}|)$
4      Initialise partial map $\phi_j : \mathcal{F} \cup \mathcal{V} \to \mathcal{F} \cup \mathcal{V}_S$
5      Let $\ell_j = \gamma_1 \ldots \gamma_m \in (\mathcal{F} \cup \mathcal{V})^*$
       // We store terms without symbols (, ), ,
6      Set $k \leftarrow 1$
       // Rename variable symbols from left to right
7      for $i \leftarrow 1$ to $m$ do                                  // $\mathcal{O}(s \log |\mathcal{V}|)$
8          if $\gamma_i \in \mathcal{V}$ and $\phi_j(\gamma_i)$ undefined then        // $\mathcal{O}(\log |\mathcal{V}|)$
9              Set $\phi_j(\gamma_i) \leftarrow x_k$
10             $k \leftarrow k + 1$
11         end
12     end
13 end
   // Extend $\phi_j$ to all of $\mathcal{F} \cup \mathcal{V}$
14 for $x \in \mathcal{V}$ do                                         // $\mathcal{O}(|\mathcal{V}| \log |\mathcal{V}|)$
15     if $\phi_j(x)$ undefined then                                  // $\mathcal{O}(\log |\mathcal{V}|)$
16         Set $\phi_j(x) \leftarrow x_k$
17         $k \leftarrow k + 1$
18     end
19 end
20 for $f \in \mathcal{F}$ do                                         // $\mathcal{O}(|\mathcal{F}| \log |\mathcal{F}|)$
21     Set $\phi_j(f) \leftarrow f$
22 end
23 return $\phi_\mathcal{V} = (\phi_1, \ldots, \phi_n)$
```

Algorithm 4.2: Computation of $\mathcal{F}$-local template

```
   input   : TRS $R = (\mathcal{F}, \mathcal{V}, \mathrm{ar}, \mathcal{R})$
   output  : $\phi_{\mathcal{F}} = (\phi'_1, \ldots, \phi'_n)$ family of $\mathcal{V}$-invariant term homomorphisms,
             canonical rewriting of function symbols on a per rule basis
   runtime: $\mathcal{O}(s|\mathcal{R}||\mathcal{V}||\mathcal{F}|\log(|\mathcal{V}||\mathcal{F}|))$
   // $\mathcal{R} = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$
1  Let $L \leftarrow \{\mathrm{ar}(f) : f \in \mathcal{F}\}$                    // $\mathcal{O}(|\mathcal{F}|\log|\mathcal{F}|)$
2  for $l \in L$ do                                                              // $\mathcal{O}(|\mathcal{F}|)$
3      Let $p_l \leftarrow 0$
4  end
5  Initialise new function symbol set $\mathcal{F}_S \leftarrow \emptyset$
6  for $j \leftarrow 1$ to $n$ do                                                // $\mathcal{O}(s|\mathcal{R}|\log|\mathcal{F}|)$
7      Initialise partial map $\phi'_j : \mathcal{F} \cup \mathcal{V} \to \mathcal{F}_S \cup \mathcal{V}$
8      Let $\ell_j r_j = \gamma_1 \ldots \gamma_m \in (\mathcal{F} \cup \mathcal{V})^*$
       // We store terms without symbols (, ), ,
       // Rename function symbols from left to right
9      for $i \leftarrow 1$ to $m$ do                                            // $\mathcal{O}(s\log|\mathcal{F}|)$
10         if $\gamma_i \in \mathcal{F}$ and $\phi_j(\gamma_i)$ undefined then
11             Let $l \leftarrow \mathrm{ar}(\gamma_i)$                          // $\mathcal{O}(\log|\mathcal{F}|)$
12             Set $p_l \leftarrow p_l + 1$
13             Set $\mathcal{F}_S \leftarrow \mathcal{F}_S \cup \{f_{p_l,l}\}$
14             Set $\phi'_j(\gamma_i) \leftarrow f_{p_l,l}$                      // $\mathcal{O}(\log|\mathcal{F}|)$
15         end
16     end
17 end
   // Extend $\phi'_j$ to all of $\mathcal{F} \cup \mathcal{V}$
18 for $f \in \mathcal{F}$ do                                                    // $\mathcal{O}(|\mathcal{F}|\log|\mathcal{F}|)$
19     if $\phi'_j(f)$ undefined then
20         Let $l \leftarrow \mathrm{ar}(f)$                                     // $\mathcal{O}(\log|\mathcal{F}|)$
21         Set $p_l \leftarrow p_l + 1$
22         Set $\mathcal{F}_S \leftarrow \mathcal{F}_S \cup \{f_{p_l,l}\}$
23         Set $\phi'_j(\gamma_i) \leftarrow f_{p_l,l}$                          // $\mathcal{O}(\log|\mathcal{F}|)$
24     end
25 end
26 for $x \in \mathcal{V}$ do                                                    // $\mathcal{O}(|\mathcal{V}|\log|\mathcal{V}|)$
27     Set $\phi'_j(x) \leftarrow x$
28 end
29 return $\phi_{\mathcal{F}} = (\phi'_1, \ldots, \phi'_n)$
```

5 Summary and Conclusion

Understandably, we did not introduce new results to characterise categorisation of class **GI** more precisely. Instead we gave a compact, historical overview, highlighting the most important papers and contributions leading up to the current state of research in the field of graph isomorphism, with particular focus László Babai. After gradually refining methods based on results by himself and Eugene Luks, dating back as early as 1982, he finally provided a quasipolynomial time algorithm for **GI** in 2015, partially corrected and improved by Harald Helfgott in 2017. This led into three main arguments against **NP**-completeness of **GI**, based on heuristic evidence, involving Babais' quasipolynomial claim, Uwe Schöning's proof of correlation between **GI** and the polynomial hierarchy, possibly leading to its collapse, and the fact that **GI** is in **coAM**. The pending per-review may hopefully open the door for new theoretical and practical achievements in the future, and spark new interest in this field of research.
In preparation for the core **GI**-completeness-theorems, we subsequently studied (strong) isomorphism restricted to certain subclasses of graphs, like directed graphs, pseudographs or labelled directed graphs. By using edge replacement and pointer techniques to encode one class into an other, we convinced ourselves that these specialised isomorphism problems are polynomially equivalent to the standard problem of determining isomorphisms between unlabelled undirected graphs. Similar constructions can be used to prove **GI**-completeness of other classes, as long as they are of a certain "universality", in contrast to highly specialised graph classes like trees, which are, as we noted, computationally much easier.
This led into the conclusion that determining grammar isomorphisms and isomorphic strict interpretations on context-free/regular grammars, and global isomorphisms on term rewriting systems are also **GI**-complete. Both problems could be handled with similar methods, only slightly differing in their application. Instead of explicitly stating graph encodings for every special case, we took advantage of the general structure of respective objects in form of templates, in particular the relationship between the different symbol sets, i.e. nonterminals vs. terminals or function symbols vs. variable symbols. Algorithm 3.1, Algorithm 4.1 and Algorithm 4.2 can easily be generalised to handle more than two disjoint symbol sets and can then be applied for reductions or polynomial solvability proofs if isomorphisms demand symbol-relationship to be preserved. In fact, we saw that, under canonical local rewriting of symbols, proof of isomorphism reduces to simple set comparison, which is always possible in polynomial time if binary comparison of structural components is reasonably defined. The runtime of presented algorithms can be greatly improved by choosing more appropriate data structures, better encapsulating defining properties.

List of Algorithms

List of Tables

List of Figures

Bibliography

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Co., Boston, Massachusetts, USA, first edition, 1974.

[Ave95] J. Avenhaus. *Reduktionssysteme – Rechnen und Schliessen in gleichungsdefinierten Strukturen.* Springer-Lehrbuch. Springer, Berlin, Heidelberg, D, first edition, 1995.

[Bab85] L. Babai. Trading Group Theory for Randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, New York, NY, USA, 1985. Association for Computing Machinery.

[Bab16] L. Babai. Graph Isomorphism in Quasipolynomial Time. `https://arxiv.org/pdf/1512.03547.pdf` [2016-01-19], 2016.

[Bab17] L. Babai. Fixing the UPCC case of Split-or-Johnson. `https://people.cs.uchicago.edu/~laci/upcc-fix.pdf` [2017-01-14], 2017.

[Bab19] L. Babai. *GROUP, GRAPHS, ALGORITHMS: THE GRAPH ISOMORPHISM PROBLEM*, pages 3319–3336. World Scientific Publishing Co Pte Ltd, 2019.

[BC79] K. S. Booth and C. J. Colbourn. *Problems polynomially equivalent to graph isomorphism.* Technical Report CS-77-04, University of Waterloo, 1979.

[BC08] L. Babai and P. Codenotti. Isomorhism of Hypergraphs of Low Rank in Moderately Exponential Time. In *2008 Fourty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 667–676, Washington, DC, USA, 2008. IEEE Computer Society.

[BL83] L. Babai and E. M. Luks. Canonical Labeling of Graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. Association for Computing Machinery.

[Bod90] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

[CFI92] JY. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, third edition, 2009.

[Col81] C. J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21, 1981.

[Coo71] S. A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[Coo72] S. A. Cook. A Hierarchy for Nondeterministic Time Complexity. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 187–192, New York, NY, USA, 1972. Association for Computing Machinery.

[Fru39] R. Frucht. Herstellung von Graphen mit vorgegebener abstrakter Gruppe. *Compositio Mathematica*, 6(1):239–250, 1939.

[Gen15] J. Gentzen. A canonical labeling technique by Brendan McKay and isomorphism testing of deterministic finite automata. Post on Blog gentzen.wordpress.com, 2015. Accessed: 2021-03-10.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman and Company, New York, NY, USA, first edition, 1979.

[HBD17] H. A. Helfgott, J. Bajpai, and D. Dona. Graph isomorphisms in quasi-polynomial time. `https://arxiv.org/pdf/1710.04574.pdf` [2017-10-12], 2017.

[HMS11] T. Hertli, R. A. Moser, and D. Scheder. Improving PPSZ for 3-SAT using Critical Variables. `https://arxiv.org/pdf/1009.4830.pdf` [2018-10-31], 2011.

[HS65] J. Hartmanis and R. E. Stearns. On the Computational Complexity of Algorithms. *Transactions of the American Mathematical Society*, 117(1):285–306, 1965.

[HW74] J. E. Hopcroft and J. K. Wong. Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 172–184, New York, NY, USA, 1974. Association for Computing Machinery.

[IP99] R. Impagliazzo and R. Paturi. Complexity of k-SAT. In *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference) (Cat.No.99CB36317)*, pages 237–240, Washington, DC, USA, 1999. IEEE Computer Society.

[Kar72] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, 1972.

[Kla15] E. Klarreich. Landmark Algorithm Breaks 30-Year Impasse. Online Article published by Quanta Magazine, 2015. Accessed: 2021-03-12.

[KS10] K. Kutzkov and D. Scheder. Using CSP To Improve Deterministic 3-SAT. `https://arxiv.org/pdf/1007.1166.pdf` [2018-10-26], 2010.

[Lad75] R. E. Ladner. On the Structure of Polynomial Time Reducibility. *J. ACM*, 22(1):155–171, 1975.

[Luk82] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

[RH85] D. J. Rosenkrantz and H. B. Hunt. Testing for Grammatical Coverings. *Theoretical Computer Science*, 38(1):323–341, 1985.

[Sch88] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.

[Sch08] U. Schöning. *Theoretische Informatik – kurz gefasst*. Spektrum Akademischer Verlag, Heidelberg, D, fifth edition, 2008.

[Sch09] P. Schweitzer. *Problems of Unknown Complexity: Graph isomorphism and Ramsey theoretic numbers*. PhD thesis, Naturwissenschaftlich-Technische Fakultäten der Universität des Saarlandes, 2009.

[Spi96] D. A. Spielman. Faster Isomorphism Testing of Strongly Regular Graphs. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 576–584, New York, NY, USA, 1996. Association for Computing Machinery.

[SSRS13] M. Schmidt-Schauß, C. Rau, and D. Sabel. Algorithms for Extended Alpha-Equivalence and Complexity. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 255–270, Dagstuhl, D, 2013. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[vM50] D. van Melkebeek. *Randomness and Completeness in Computational Complexity*. Springer-Verlag, Berlin, Heidelberg, D, first edition, 1950.

[ZKT85] V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.