DECIDABILITY OF FINITE SATISFIABILITY OF TWO-VARIABLE FIRST-ORDER LOGIC WITH COUNTING AND LOCAL NAVIGATION IN UNORDERED UNRANKED TREES

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2018

By
Yegor Guskov
School of Computer Science

Contents

Abstract **9**

Declaration **10**

Copyright **11**

Acknowledgements **12**

Notation **13**

1 Introduction **17**

2 Background **22**

2.1 Two-variable fragment 22

2.2 Two-variable fragment with transitive relations 23

2.3 Two-variable fragment with total preorder relations 27

2.4 Two-variable fragment with equivalence relations 32

2.5 Two-variable fragment with linear order relations 34

2.6 Two-variable fragment with tree navigation relations 38

2.6.1 Complexity results 41

2.6.2 $\mathcal{FO}^2[]$ with navigation in trees and XPath 43

2.7 Two-variable fragment with counting 45

2.7.1 Expressiveness of $\mathcal{C}^2()$ 46

2.7.2 Complexity results 47

2.7.3 Two-variable first-order logic with counting and tree navigation predicates 49

3 Terminology **51**

4 Finite satisfiability of $\mathcal{C}^2(\downarrow^+)$ **58**
4.1 Preliminaries 58
4.2 Auxiliary sentences and translators 61
4.2.1 The sentence ω_{arb} and the arboreal property 61
4.2.2 The sentence ω_{alt} and the alternating property 62
4.2.3 The translator Ω_{chr} and the chromatic property 62
4.2.4 The translator Ω_{diff} and the differentiated property 63
4.2.5 The translator Ω_{frst} and labelling forest elements 64
4.2.6 The translator Ω_{pop} and the populated property 66
4.2.7 The translator Ω_{rew} and the rewirable property 67
4.2.8 The translator Ω 68
4.3 Existence of silent pairs 70
4.4 Model rewiring 77
4.5 Summary 88

5 Conclusions **90**

Bibliography **93**

A Rewiring cases **98**

List of Figures

1.1 Relationship between predicates 20

2.1 An infinite model 24
2.2 Satisfiability of two-variable and monadic two-variable first-order logic with transitive relations 27
2.3 A sketch of a model of a total preorder 29
2.4 Finite satisfiability of monadic two-variable first-order logic with total preorder relations 31
2.5 A sketch of a model of an equivalence relation 32
2.6 Finite satisfiability of two-variable and monadic two-variable first-order logic with equivalence relations 34
2.7 Finite satisfiability of monadic two-variable first-order logic with linear order relations 37
2.8 Finite satisfiability of monadic two-variable first-order logic with total preorder and linear order relations 39
2.9 An example of relation $\downarrow^*$ interpreted as an unordered tree 40
2.10 An example of relations $\downarrow^*$ and $\rightarrow^*$ interpreted as an ordered tree 40
2.11 Finite satisfiability of two-variable first-order logic with tree navigation relations 44
2.12 Finite satisfiability of two-variable first-order logic with counting . 48

3.1 An example of a tree-like cycle 56

4.1 Before changing 2-types 60
4.2 After changing 2-types 60
4.3 Case 1 73
4.4 Before rewiring (Case 2) 73
4.5 After rewiring (Case 2) 73
4.6 Before rewiring (Case 3) 74

4.7 After rewiring (Case 3) 74
4.8 Case 4 74
4.9 Before rewiring (Case 6) 75
4.10 After rewiring (Case 6) 75
4.11 Before rewiring (Case 5) 75
4.12 After rewiring (Case 5) 75
4.13 Before changing 2-types 78
4.14 After changing 2-types 78
4.15 Case 1 before rewiring 79
4.16 Case 1 after rewiring 79
4.17 Case 2 before rewiring 80
4.18 Case 2 after rewiring 80
4.19 Case 3 before rewiring 80
4.20 Case 3 after rewiring 80
4.21 Case 4 before rewiring 81
4.22 Case 4 after rewiring 81
4.23 Case 5 before rewiring 81
4.24 Case 5 after rewiring 81
4.25 Case 7 before rewiring 82
4.26 Case 7 after rewiring 82
4.27 Case 8 before rewiring 82
4.28 Case 8 after rewiring 82
4.29 Case 10(a) before rewiring 83
4.30 Case 10(a) after rewiring 83
4.31 Case 10(b) before rewiring 83
4.32 Case 10(b) after rewiring 83
4.33 Case 10(c) before rewiring 84
4.34 Case 10(c) after rewiring 84
4.35 Case 11(a) before rewiring 84
4.36 Case 11(a) after rewiring 84
4.37 Case 11(b) before rewiring 85
4.38 Case 11(b) after rewiring 85
4.39 Case 11(c) before rewiring 85
4.40 Case 11(c) after rewiring 85
4.41 Case 12(a) before rewiring 86

4.42 Case 12(a) after rewiring 86
4.43 Case 12(b) before rewiring 86
4.44 Case 12(b) after rewiring 86
4.45 Case 12(c) before rewiring 87
4.46 Case 12(c) after rewiring 87
4.47 Case 17 before rewiring 87
4.48 Case 17 after rewiring 87

A.1 Case 1 before rewiring 98
A.2 Case 1 after rewiring 98
A.3 Case 2 before rewiring 99
A.4 Case 2 after rewiring 99
A.5 Case 3 before rewiring 99
A.6 Case 3 after rewiring 99
A.7 Case 4 before rewiring 100
A.8 Case 4 after rewiring 100
A.9 Case 5 before rewiring 100
A.10 Case 5 after rewiring 100
A.11 Case 6 before rewiring 101
A.12 Case 6 after rewiring 101
A.13 Case 7 before rewiring 101
A.14 Case 7 after rewiring 101
A.15 Case 8 before rewiring 102
A.16 Case 8 after rewiring 102
A.17 Case 9 before rewiring 103
A.18 Case 9 after rewiring 103
A.19 Case 10 (a) before rewiring 103
A.20 Case 10 (a) after rewiring 103
A.21 Case 10 (b) before rewiring 104
A.22 Case 10 (b) after rewiring 104
A.23 Case 10 (c) before rewiring 104
A.24 Case 10 (c) after rewiring 104
A.25 Case 11 (a) before rewiring 105
A.26 Case 11 (a) after rewiring 105
A.27 Case 11 (b) before rewiring 105
A.28 Case 11 (b) after rewiring 105

A.29 Case 11 (c) before rewiring 106
A.30 Case 11 (c) after rewiring 106
A.31 Case 12 (a) before rewiring 106
A.32 Case 12 (a) after rewiring 106
A.33 Case 12 (b) before rewiring 107
A.34 Case 12 (b) after rewiring 107
A.35 Case 12 (c) before rewiring 107
A.36 Case 12 (c) after rewiring 107
A.37 Case 13 (a) before rewiring 108
A.38 Case 13 (a) after rewiring 108
A.39 Case 13 (b) before rewiring 108
A.40 Case 13 (b) after rewiring 108
A.41 Case 13 (c) before rewiring 109
A.42 Case 13 (c) after rewiring 109
A.43 Case 14 (a) before rewiring 109
A.44 Case 14 (a) after rewiring 109
A.45 Case 14 (b) before rewiring 110
A.46 Case 14 (b) after rewiring 110
A.47 Case 14 (c) before rewiring 110
A.48 Case 14 (c) after rewiring 110
A.49 Case 15 (a) before rewiring 111
A.50 Case 15 (a) after rewiring 111
A.51 Case 15 (b) rewiring 111
A.52 Case 15 (b) after rewiring 111
A.53 Case 15 (c) before rewiring 112
A.54 Case 15 (c) after rewiring 112
A.55 Case 16 before rewiring 112
A.56 Case 16 after rewiring 112
A.57 Case 17 before rewiring 113
A.58 Case 17 after rewiring 113
A.59 Case 18 before rewiring 113
A.60 Case 18 after rewiring 113
A.61 Case 19 before rewiring 114
A.62 Case 19 after rewiring 114
A.63 Case 20 before rewiring 114

A.64 Case 20 after rewiring 114
A.65 Case 21 before rewiring 115
A.66 Case 21 after rewiring 115
A.67 Case 22 before rewiring 115
A.68 Case 22 after rewiring 115
A.69 Case 23 before rewiring 116
A.70 Case 23 after rewiring 116
A.71 Case 24 before rewiring 116
A.72 Case 24 after rewiring 116
A.73 Case 25 before rewiring 117
A.74 Case 25 after rewiring 117

Abstract

First-order logic is a widely used relational language of reasoning. Due to the fact that it is algorithmically impossible to determine whether a given sentence of first-order logic is valid or not, various restrictions of first-order logic have been devised to obtain languages that are less expressive but can be reasoned about by computers more easily.

One of such prominent fragments of first-order logic is the two-variable fragment. Not only is it possible to check the validity of the sentences of this logic in a bounded amount of time, it is also the case that adding various useful reasoning capabilities to two-variable first-order logic often preserves the time-bounded validity check.

This thesis addresses the problem of the complexity of decidability of various extensions of two-variable first-order logic on a broad scale by offering a systematic overview and comparison of the available results in the field.

In addition to that the thesis gives the proof of decidability of the extension of two-variable first-order logic with counting quantifiers and the relation that models parent/child relations in a tree graph. The NEXPTIME upper complexity bound is established for the problem of finite satisfiability of this logic.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank my supervisor Dr. Ian Pratt-Hartmann for his utmost professionalism and all the energy he put into supporting my progress.

I would also like to thank George Kourtis, Francis Southern and Joseph Razavi for creating an environment of learning at the School of Computer Science of the University of Manchester.

Notation

Fragments of first-order logic

$\mathcal{FO}$ – First-order logic.

$\mathcal{FO}^2()$ – Two-variable first-order logic. The set of sentences of $\mathcal{FO}$ in which only the variables x and y appear.

$\mathcal{FO}^2[]$ – Monadic two-variable first-order logic. The set of sentences of $\mathcal{FO}^2()$ in which only unary predicates appear.

$\mathcal{FO}^2_S[]$ – Singular monadic two-variable first-order logic. The set of sentences of $\mathcal{FO}^2[]$ that can only be interpreted in such a way that for every element of a model of a sentence exactly one unary predicate from the signature of the sentence holds true (unary predicates form a partition of the universe of every model of the sentences of the language).

$\mathcal{GF}$ – Guarded fragment of first-order logic.

$\mathcal{GF}^2()$ – Two-variable guarded fragment of first-order logic. The set of sentences of $\mathcal{GF}$ in which only the variables x and y appear.

$\mathcal{GF}^2[]$ – Monadic two-variable guarded fragment of first-order logic. The set of sentences of $\mathcal{GF}^2()$ in which only unary predicates appear.

$\mathcal{GF}^2_S[]$ – Singular monadic two-variable guarded fragment of first-order logic. The set of sentences of $\mathcal{GF}^2[]$ that can only be interpreted in such a way that for every element of a model of a sentence exactly one unary predicate from the signature of the sentence holds true (unary predicates form a partition of the universe of every model of the sentences of the language).

$\mathcal{C}^2()$ – Two-variable first-order logic with counting quantifiers $\exists_{\geq n}$, $\exists_{\leq n}$, $\exists_{=n}$.

$\mathcal{C}^2[]$ – Monadic two-variable first-order logic with counting quantifiers. The set of sentences of $\mathcal{C}()$ in which only unary predicates appear.

Special predicates

$\downarrow^+$ – Binary predicate that is interpreted as a tree graph.

$\downarrow_R^+$ – Binary predicate that is interpreted as a ranked tree graph. A ranked tree graph is a tree graph in which every node has n or fewer children, for some given number n.

$\downarrow^*$ – Binary predicate that is interpreted as the transitive closure of a tree graph.

$\rightarrow^+$ – Binary predicate that is interpreted as the "next sibling" relation of an ordered tree graph.

$\rightarrow^*$ – Binary predicate that is interpreted as the transitive closure of the "next sibling" relation of an ordered tree graph.

$<$ – Binary predicate that is interpreted as a transitive relation.

$\lesssim^*$ – Binary predicate that is interpreted as a total preorder relation. Total preorder is a relation that is reflexive, transitive and total.

$\lesssim^+$ – Binary predicate that is interpreted as the successor of a total preorder relation.

$\sim$ – Binary predicate that is interpreted as an equivalence relation.

$\widehat{\sim}$ – Binary predicate that is interpreted as an equivalence closure of a given binary relation.

$\precsim$ – Binary predicate that is interpreted as a strict partial order relation. Strict partial order is an irreflexive transitive binary relation. It is also antisymmetric, which is implied by the first two properties.

$\leq^*$ – Binary predicate that is interpreted as a linear order.

$\leq^+$ – Binary predicate that is interpreted as the successor of a linear order.

Terminology

$\phi \in \mathcal{X}$ – A sentence from the set of sentences of some logic fragment $\mathcal{X}$.

$\mathfrak{A} \models \phi$ – A model of some given sentence ϕ in which the sentence is true.

$e \in \mathfrak{A}$ – An element e of the universe of a given model $\mathfrak{A}$.

$\Sigma(\phi)$ – The signature of a given sentence ϕ.

$\mathfrak{A}|\Sigma(\phi)$ – The model generated by taking the substructure over a given model $\mathfrak{A}$ and restricting it to the signature of a given sentence ϕ. Alternatively written as $\mathfrak{A}|\phi$.

$\tau(\phi)$ – The set of all 1-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\mathrm{tp}^{\mathfrak{A}}[e]$ – The 1-type of a given element e in a given model $\mathfrak{A}$.

$\tau_i \in \tau(\phi)$ – A 1-type τ_i from the set of 1-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\tau_i(x)$ – The conjunctive formula of $\mathcal{FO}^2()$ over the free variable x that expresses a given 1-type τ_i.

$\pi(\phi)$ – The set of all 2-types over the signature $\Sigma(\phi)$ for a given sentence ϕ.

$\mathrm{tp}^{\mathfrak{A}}[e, e']$ – The 2-type of some given elements e and e' in a given model $\mathfrak{A}$.

$\pi_i \in \pi(\phi)$ – A 2-type π_i from the set of 2-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\pi_i(x, y)$ – The conjunctive formula of $\mathcal{FO}^2()$ over the free variables x and y that expresses a given 2-type π_i.

$s(\phi)$ – The set of all silent 2-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\mu(\phi)$ – The set of all message-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\mu^{\rightarrow}(\phi)$ – The set of all forward message-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\mu^{\leftarrow}(\phi)$ – The set of all reverse message-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\mu^{\leftrightarrow}(\phi)$ – The set of all invertible message-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\varepsilon(\phi)$ – The set of all edge-types over the signature $\Sigma(\phi)$ of a given sentence ϕ.

$\|\mathfrak{A}\|$ – The size of a given model $\mathfrak{A}$.

$\omega = \Omega(\phi)$ – The sentence ω generated by a nondeterministic run of a given translator Ω on a given sentence ϕ.

$\Omega(\phi) := \omega_1 \wedge \omega_2 \wedge ... \wedge \omega_i$ – The definition of the translator Ω that takes as input ϕ and produces as output the sentence $\omega_1 \wedge \omega_2 \wedge ... \wedge \omega_i$ for some set of sentences $\omega_1, \omega_2, ..., \omega_i$.

$\mathrm{st}^{\mathfrak{A}}[e]$ – The star-type of some given element e in a given model $\mathfrak{A}$.

Abbreviations

RE – The complexity class of recursively-enumerable problems

Co-RE – The complexity class of co-recursively-enumerable problems

VAS – The complexity class of problems that are equivalent in complexity to the problem of reachability in vector addition systems.

Chapter 1

Introduction

First-order logic, denoted $\mathcal{FO}$, is a relational language of reasoning that was first formulated by Hilbert in 1917 [Moore, 1988] and has since been very well studied and found a wide range of applications in theoretical and practical fields of science and technology.

One of the key questions concerning first-order logic is the problem of establishing whether there exists a reliable way of finding which sentences of first-order logic are theorems and which are not. This problem is called the problem of validity of sentences of first-order logic and in [Turing, 1936] it was shown to be algorithmically undecidable.

Related to the problem of validity of sentences of $\mathcal{FO}$ is the problem of the satisfiability of the sentences of the language. A sentence of first-order logic is called satisfiable whenever there exists an interpretation of that sentence symbols in which the sentence is true. A sentence that is satisfiable in all interpretations is a theorem and is called valid. If the logical negation of a given sentence is unsatisfiable, then the sentence holds true in all interpretations and is a theorem as well. Therefore, the problem of satisfiability of sentences of $\mathcal{FO}$ is as hard as the problem of validity of sentences of $\mathcal{FO}$ and is also undecidable.

Since it is practically desirable to operate with sentences of logic the satisfiability status of which can be resolved in a finite amount of time, restrictions on the language of $\mathcal{FO}$ have been devised to overcome the problem of undecidability of satisfiability. Such restrictions of $\mathcal{FO}$ are called fragments of first-order logic, since they posses only part of the expressive capabilities of $\mathcal{FO}$. The idea behind artificially limiting the expressive power of first-order logic lies in the fact that in many cases a new language can be obtained in such a way that, while

retaining useful capabilities for reasoning, it has a decidable satisfiability problem associated with it.

One way to obtain a decidable fragment of $\mathcal{FO}$ is to limit the number of variable names that can appear in the sentences of first-order logic. For instance, the logic that is formed from the set of all sentences of $\mathcal{FO}$ that contain no function symbols and no constants and in which only two variable names x and y appear, is called two-variable first-order logic and it is denoted $\mathcal{FO}^2()$. It was shown half a century ago that the satisfiability problem of $\mathcal{FO}^2()$ is decidable[1].

By the time the decidability of the satisfiability problem of two-variable first-order was established, it was already known that the three-variable logic is not algorithmically solvable, since it contains the undecidable $\forall\exists\forall$ prefix class of first-order logic [Kahr et al., 1962]. This lead researchers to enquire how two-variable first-order logic can be extended with additional capabilities of reasoning, while preserving the decidability of the resulting language.

One way of increasing the expressive power of two-variable first-order logic is by adding to the language one or more special binary predicates that are known to have certain properties, which are inexpressible in the language of two-variable first-order logic itself. In other words, the special binary predicates added to the language of two-variable first-order logic are allowed to receive their interpretation only from a certain class of structures, such as equivalence relations, linear orders and others. The new languages obtained in such a way can describe more specific and more complicated structures that cannot be distinguished from each other by the sentences of two-variable first-order logic alone.

The inspiration for extending $\mathcal{FO}^2()$ with binary predicates that have a fixed interpretation can be traced to research in modal logics. According to [Szwast and Tendera, 2013], placing special restrictions on the reachability relation in the frame-structures of the sentences of modal logics makes it possible to model such concepts as equivalence and transitivity in these propositional languages. Additionally, it is of historical interest to mention that in an early paper in the field of decidability of two-variable fragments of first order logic Erich Grädel, Martin Otto and Eric Rosen claim [Grädel et al., 1998]: "We would like to acknowledge that the programme to investigate the borderline of decidability in the neighbourhood of $\mathcal{FO}^2()$ has been strongly advocated by Moshe Vardi, whose questions and

[1]The history of establishing the satisfiability of two-variable first-order logic and the references to the relevant literature can be found in Section 2.1.

conjectures have been an important incentive for our study."

Two-variable first-order logic was found to be useful in its own right, as it turned out that this language is expressive enough for various practical applications. The importance of the two-variable fragment of first-order logic and its extensions is well highlighted by the quote from [Kieronski and Tendera, 2009]: "One of the main motivations to study $\mathcal{FO}^2()$ comes from the fact that it embeds propositional modal logic which in the last decades has been used in many areas of computer science like e.g. artificial intelligence, program verification, database theory and distributed computing... $\mathcal{FO}^2()$ is also used as a representative language for a number of knowledge representation logics (description logics)... Many powerful variants of description logics can be embedded in the extension of $\mathcal{FO}^2()$ with counting quantifiers."

Because of the fact that other less expressive logics are in some sense contained within two-variable first-order logic, studying the satisfiability problem of two-variable first-order logic has theoretical benefits as well. If a new formal language is devised, its expressive power can often be compared to a relevant extension of two-variable first-order logic in a straightforward way. Depending on whether such a formal language is more expressive or less expressive than a particular extension of two-variable logic, a lower or upper complexity bound can be immediately derived for the formal language from the complexity bound of the two-variable fragment (if such a complexity bound is already known for that particular two-variable fragment).

Overview of following chapters

This thesis deals with the problem of decidability of satisfiability of two-variable first-order logic and its extensions both in broad scale and in specific terms.

Chapter 2 presents the currently known complexity results in the field of satisfiability of two-variable first-order logics. Because the results in this area are numerous and interrelated, they are given a systematic treatment, using diagrams that show the relative complexity and expressivity of various two-variable logics extended with a particular type of relations.

The information about the decidability of satisfiability of extensions of two-variable logic is grouped in this chapter by the kind of special binary predicates that two-variable first-order logic is extended with. The chapter starts with the review of the presently available results concerning the most general type

of predicate first, which is transitivity. Then, sections about more specific types of predicates with more restricted properties, such as total preorders, follow.

Figure 1.1 demonstrates how different types of binary relations are nested in terms of their generality within each other. The transitive relation is the most general relation among the relations that have been studied in conjunction with two-variable first-order logic. The rest of the classes of relations are derived from transitivity by applying more and more specific restrictions, such as reflexivity and antisymmetry, to them.

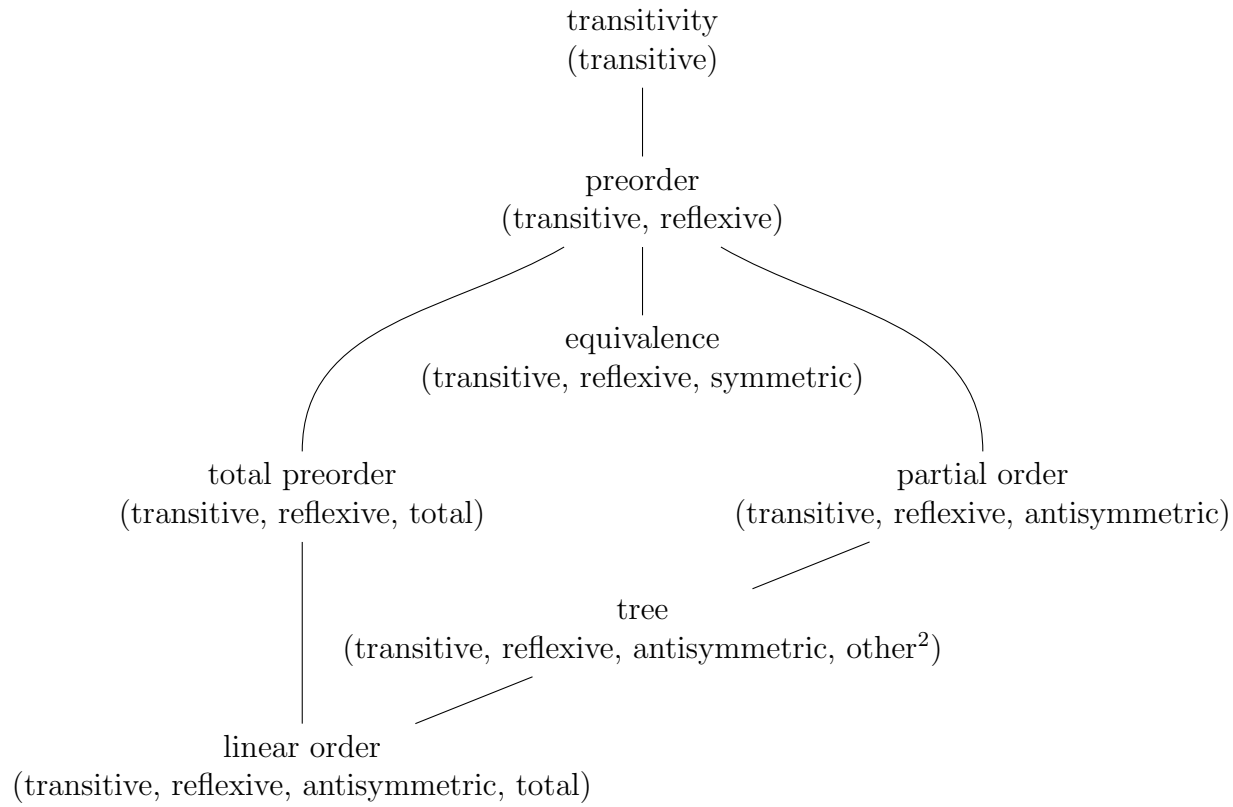


Figure 1.1: Relationship between predicates

The chapter is concluded by a section that presents an overview of the complexity results about the extensions of two-variable first-order logic with special binary predicates and counting quantifiers.

Chapter 3 lays out the terminological apparatus that is going to be necessary in the next part of the thesis for the rigorous demonstration of a new technical finding in this area of research – the proof that the finite satisfiability of two-variable logic with a local tree navigation predicate and counting quantifiers is

decidable in NEXPTIME.[2]

Chapter 4 presents the main novel contribution of this thesis: the proof that the problem of finite satisfiability of a specific extension of two-variable first-order logic, namely the two-variable first-order logic with counting quantifiers and the successor of a vertical tree navigation relation, is decidable in NEXPTIME. Thus, the state of knowledge in the subject of decidability of satisfiability of extensions of two-variable first-order logic is advanced forward. The chapter starts with a preliminaries Section 4.1 that provides a layout of the proof structure and describes informally the key techniques employed in obtaining the upper complexity bound. Precise definition of the syntax and semantics of two-variable first-order logic with the successor of a vertical tree navigation relation is given at the beginning of Chapter 3.

Chapter 5 summarises what was achieved in Chapter 2 and Chapter 4 and explains what future desirable results can be obtained both in the broad field of satisfiability of the extensions of two-variable logics and in the narrow field of satisfiability of two-variable first-order logic with counting and tree navigation predicates.

[2]A tree is a partial order that has one minimum element and for any node of the tree, the set of the nodes that the given node relates to is well-ordered

Chapter 2

Background

2.1 Two-variable fragment

In 1966 Dana Scott showed how the sentences of first-order logic with two variables, denoted $\mathcal{FO}^2()$, can be translated into the sentences of the Gödel prefix fragment $\exists\exists\forall^*$ without equality, which was already known to be decidable at the time [Scott, 1962], and thus established decidability of the satisfiability of $\mathcal{FO}^2()$.

Later in [Mortimer, 1975] the finite model property was established for the two-variable first-order logic with equality. The finite model property was proved by showing that no sentence in the logic is satisfied only by infinite models through the application of Ehrenfeucht–Fraïssé games (Mortimer citing [Ehrenfeucht, 1961]). The author said that "this answers a question raised by W. Hodges" and that the finite model property confirms the result of Scott that the two-variable first-order logic is decidable. In fact, the author showed that if a sentence has a model, then the sentence has a model of at most doubly-exponential size with respect to the length of the sentence.

The result about the doubly-exponential small model property of $\mathcal{FO}^2()$ was improved to the exponential small model property in [Grädel et al., 1997a]. At the time of the publication of [Grädel et al., 1997a] it was known from the results of [Lewis, 1980] and [Fürer, 1984] that the satisfiability of $\mathcal{FO}^2()$ was at least NEXPTIME-hard. As a consequence of this, the tight NEXPTIME complexity bound was established for the satisfiability of $\mathcal{FO}^2()$.

The same NEXPTIME complexity upper bound and lower bound apply to the finite satisfiability problem of $\mathcal{FO}^2()$, because of the finite model property of the logic: for every infinite model of a given sentence, there exists a finite model

of the sentence.

Later in [Etessami et al., 2002] NEXPTIME-hardness (lower complexity bound) result was obtained for the restriction of the language of $\mathcal{FO}^2()$ to the set of two-variable sentences containing only unary predicates, denoted $\mathcal{FO}^2[]$. This restriction of $\mathcal{FO}^2()$ is going to be called *monadic two-variable first-order logic*. The word 'monadic' is employed here only to designate the fact that the language of $\mathcal{FO}^2[]$ does not have any binary relations, and only unary relations are allowed to appear in the language sentences.

From now on, the special symbols are going to appear in the round and square brackets of the symbol $\mathcal{FO}^2()$ and $\mathcal{FO}^2[]$ to designate extensions of two-variable first-order logic and monadic two-variable first-order logic with special binary predicates. In the case of the former logic, this means that certain binary predicates are assigned a specific interpretation, but there are no changes to the syntax of $\mathcal{FO}^2()$. In the case of the latter logic, this means that the sentences of the language can contain not only the unary predicates, but also the specific binary predicates that are mentioned in the square brackets (the interpretations of these binary predicates are fixed as well).

Denoting two-variable first-order logic as $\mathcal{FO}^2()$ and monadic two-variable first-order logic as $\mathcal{FO}^2[]$ is introduced in this thesis for the purpose of differentiating between the extensions of these logics that will be discussed in the next sections.

2.2 Two-variable fragment with transitive relations

One of the limitations of $\mathcal{FO}^2()$ is that certain types of structures cannot be expressed with just two variables. For example, it is possible to encode the transitivity property for an arbitrary binary predicate R using three variables in the following way.

$$\forall x \forall y \forall z((R(x,y) \wedge R(y,z)) \rightarrow R(x,z))$$

However, the language of $\mathcal{FO}^2()$ cannot force all interpretations of a given binary relation to have the transitivity property. It is noted in [Grädel et al., 1998] that the fact that transitivity is not expressible in two-variable infinitary

logic $\mathcal{FO}^2()$ follows from standard pebble game arguments.

Another way to arrive at the same conclusion is to observe that if the transitivity property was expressible in $\mathcal{FO}^2()$ for a given binary relation R, then the following sentence would be satisfiable only in infinite models.

$$\exists x \forall y \neg R(y, x) \wedge \forall x \neg R(x, x) \wedge \forall x \exists y R(x, y) \tag{2.1}$$

This first conjunct of the sentence says that there exists an element in the model that no other element is related to by R. The second conjunct of the sentence says that no element of the model is related to itself by R. The third conjunct of the sentence says that for every element there exists an element such that the former element is related to the latter element by R. Since the relation R is assumed to be transitive, the model of this sentence must necessarily contain the infinite chain of elements in itself (as depicted in Figure 2.1). This contradicts the fact that $\mathcal{FO}^2()$ has the finite model property and every sentence of $\mathcal{FO}^2()$ that has a model, has a finite model.

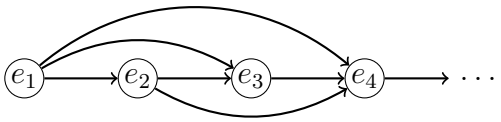


Figure 2.1: An infinite model

The ability to reason about binary relations with the transitivity property is useful in many scenarios. This is because it is possible to use the transitivity relation for expressing in $\mathcal{FO}^2()$ other properties of binary relations such as a partial order, a preorder, a linear order, an equivalence relation and others that are not expressible in $\mathcal{FO}^2()$ otherwise.

This opens the question of the decidability of satisfiability of the extension of $\mathcal{FO}^2()$ with a specific binary predicate $<$ that is only allowed to be interpreted as a transitive relation. More precisely, the set of sentences of this language, denoted $\mathcal{FO}^2(<)$, coincides with the set of sentences of $\mathcal{FO}^2()$ and a given sentence $\phi \in \mathcal{FO}^2(<)$ is satisfied by a given model $\mathfrak{A}$ in the logic $\mathcal{FO}^2(<)$ if and only if $\mathfrak{A} \models \phi$ and the interpretation of $<$ in $\mathfrak{A}$ has the transitivity property.

As it was demonstrated above by the sentence 2.1, given a transitive binary relation, it is possible to express an infinity axiom in $\mathcal{FO}^2()$. Therefore, there exist sentences of $\mathcal{FO}^2(<)$ that are satisfied only in infinite models and not in

finite models. Hence, unlike for $\mathcal{FO}^2()$, the satisfiability problem of $\mathcal{FO}^2(<)$ does not coincide with the finite satisfiability problem of this logic.

In [Szwast and Tendera, 2013] it was shown that the satisfiability of $\mathcal{FO}^2(<)$ is decidable in 2-NEXPTIME. The finite satisfiability of $\mathcal{FO}^2(<)$ was proved to be decidable in triply-exponential nondeterministic time in [Pratt-Hartmann, 2017].

At the same time in [Kieroński, 2003] the satisfiability of monadic two-variable first-order logic $\mathcal{FO}^2[]$ with one extra transitive binary relation $<$, denoted $\mathcal{FO}^2[<]$, was determined to be 2-EXPTIME-hard. The logic $\mathcal{FO}^2[<]$ is strictly less expressive than $\mathcal{FO}^2(<)$, as the latter is equivalent to the former, except that it also allows for the appearance of any number of other binary predicates (apart from $<$) in its sentences. Because of that, the lower complexity bound for the satisfiability of $\mathcal{FO}^2[<]$ also applies to $\mathcal{FO}^2(<)$.

Similarly, the upper complexity bound of $\mathcal{FO}^2(<)$ applies to $\mathcal{FO}^2[<]$, since an algorithm that decides the satisfiability status of the sentences of $\mathcal{FO}^2(<)$ also does so for all the sentences of $\mathcal{FO}^2[<]$. As mentioned above, the complexity upper bound of $\mathcal{FO}^2(<)$ was established to be 2-NEXPTIME in [Szwast and Tendera, 2013]. Consequently, the satisfiability of both $\mathcal{FO}^2[<]$ and $\mathcal{FO}^2(<)$ is 2-EXPTIME-hard and in 2-NEXPTIME. The exact complexity classes for these two decision problems are unknown.

In [Kieronski, 2005] it was demonstrated that both the satisfiability and the finite satisfiability of monadic two-variable first-order logic with two independent transitive relations $<_1$ and $<_2$, denoted $\mathcal{FO}^2[<_1, <_2]$, are undecidable. These relations can be mentioned in the same sentence and their simultaneous use can define more complex statements than what the use of only one transitive relation can allow. The logic $\mathcal{FO}^2[<_1, <_2]$ is strictly more expressive than the logic $\mathcal{FO}^2[<]$, as every sentence of $\mathcal{FO}^2[<]$ can be expressed in $\mathcal{FO}^2[<_1, <_2]$ by simply substituting the appearance of the relation symbol $<$ with the relation symbol $<_1$. The converse is not true, since by definition the interpretations of the two transitive relations $<_1$ and $<_2$ are not connected to each other in any way and one relation $<$ cannot alone simulate all simultaneous uses of the pair of transitive relations.

Complexity of logics in a form of a diagram

The relationships between the complexity classes of the satisfiability problem of the two-variable logics with transitivity are shown in Figure 2.2.

The figure consists of four horizontal segments, each dedicated to one of the four complexity classes: NEXPTIME, 2-EXPTIME, 2-NEXPTIME and RE.

The segment on the bottom of the picture corresponds to the NEXPTIME complexity class, and is labelled as such. It contains the fragments $\mathcal{FO}^2[]$ and $\mathcal{FO}^2()$, indicating that the complexity of the satisfiability of these two logics is NEXPTIME-complete.

The fact that $\mathcal{FO}^2[]$ is situated below $\mathcal{FO}^2()$ in the diagram and is connected with a solid line ——— to it shows that the logic $\mathcal{FO}^2()$ is strictly more expressive than $\mathcal{FO}^2[]$.

The horizontal segment dedicated to the 2-EXPTIME complexity class is situated on top of the segment dedicated to the NEXPTIME complexity class, suggesting that the former class of decision problems contains the latter.

Similarly, the horizontal segment dedicated to the 2-NEXPTIME complexity class is situated on top of the 2-EXPTIME horizontal segment.

The symbol $\underline{\mathcal{FO}^2[<]}$ is situated in the 2-EXPTIME box and the symbol $\overline{\mathcal{FO}^2[<]}$ is situated in the 2-NEXPTIME box. A dotted line ····· is connecting these two symbols across the two horizontal segments. This shows that the logic $\mathcal{FO}^2[<]$ is both 2-EXPTIME-hard and in 2-NEXPTIME. The same applies to the symbols $\underline{\mathcal{FO}^2(<)}$ and $\overline{\mathcal{FO}^2(<)}$ and the logic $\mathcal{FO}^2(<)$.

When a tight complexity bound has been established for a given logic, the logic is mentioned in the diagrams only once, without employing the underline and overline notation to designate its upper and lower complexity bounds.

The symbol $\underline{\mathcal{FO}^2[<]}$ is located underneath the symbol $\underline{\mathcal{FO}^2(<)}$ and the solid line connects the two symbols, because the 2-EXPTIME-hardness of $\mathcal{FO}^2(<)$ follows from the same complexity result about the logic $\mathcal{FO}^2[<]$. Accordingly, 2-NEXPTIME upper bound complexity result of $\mathcal{FO}^2[<]$ follows from the same complexity result concerning $\mathcal{FO}^2(<)$ and the symbol of the upper bound of the latter logic is situated above the symbol of the upper bound of the former logic on the diagram.

Finally, the horizontal segment of the diagram dedicated to the RE complexity class (the class of decision problems that are recursively enumerable) contains the logic $\mathcal{FO}^2[<_1, <_2]$, indicating that the satisfiability problem of this logic is undecidable.

The diagram with the complexities of the finite satisfiability of two-variable

logics with transitive relations is omitted, because there are not enough complexity results available in this area for a meaningful graphical comparison.

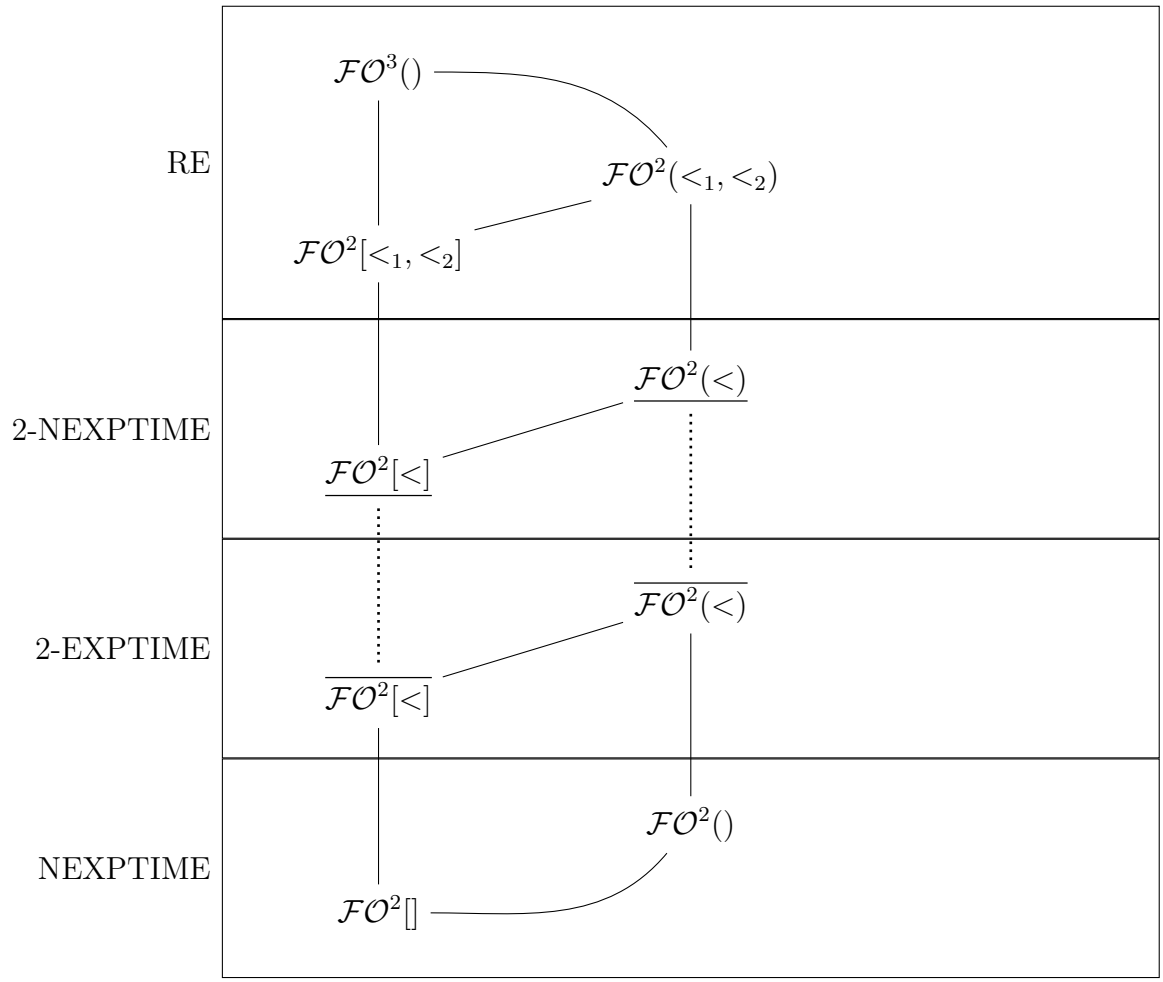


Figure 2.2: Satisfiability of two-variable and monadic two-variable first-order logic with transitive relations

2.3 Two-variable fragment with total preorder relations

Since the satisfiability of monadic two-variable first-order logic with two transitive relations $\mathcal{FO}^2[<_1, <_2]$ is undecidable [Kieronski, 2005], there is an incentive to study the complexity of satisfiability of more restricted languages.

For example, consider the extension of monadic two-variable first-order logic

$\mathcal{FO}^2[]$ with one total preorder $\lesssim^*$.[1] The resulting logic, denoted $\mathcal{FO}^2[\lesssim^*]$, has finite satisfiability problem that is in EXPSPACE [Schwentick and Zeume, 2012]. This is contrasted with the 2-EXPSPACE hardness result for the finite satisfiability of the strictly more expressive logic $\mathcal{FO}^2[<]$. The reason that $\mathcal{FO}^2[<]$ contains $\mathcal{FO}^2[\lesssim^*]$ comes from the fact that it is possible to convert the transitive relation $<$ into the total preorder relation $\lesssim^*$ in $\mathcal{FO}^2[<]$ using the following axiom.

$$\forall x <(x, x) \wedge \forall x \forall y(<(x, y) \vee <(y, x))$$

The first conjunct of this sentence expresses the reflexivity property of relation $<$ and the second conjunct of this sentence expresses the totality property of relation $<$. Therefore, the relation $<$ can only be interpreted as a total preorder with this axiom, according to the definition in the footnote 1.

As noted in Section 2.1, the lower bound complexity result for finite satisfiability of $\mathcal{FO}^2[]$ was established to be NEXPTIME in [Etessami et al., 2002]. Since the logic $\mathcal{FO}^2[\lesssim^*]$ is strictly more expressive than $\mathcal{FO}^2[]$ (extending monadic two-variable logic $\mathcal{FO}^2[]$ with one binary predicate $\lesssim^*$), the lower bound complexity result for $\mathcal{FO}^2[]$ applies to $\mathcal{FO}^2[\lesssim^*]$ as well.

The addition of two independent total preorder relations $\lesssim_1^*$ and $\lesssim_2^*$ to the monadic two-variable first-order logic results in a language, denoted $\mathcal{FO}^2[\lesssim_1^*, \lesssim_2^*]$, that has undecidable finite satisfiability problem, as in the case with $\mathcal{FO}^2[<_1, <_2]$.

Total preorder relations organise the elements of models into sets of linearly ordered equivalence classes. A total preorder relation relates each element within one such equivalence class to every other element of that class. At the same time all elements from a given equivalence class are related to all the elements of the next equivalence classes according to the linear ordering of the equivalence classes that a total preorder relation gives rise to.

Figure 2.3 provides a sketch of how a total preorder relation relates the elements of a model to each other. The ellipses in the diagram represent individual equivalence classes, within which all elements are related to each other by the total preorder relation. At the same time all elements are linearly ordered between the equivalence classes according to the equivalence classes ordering.

This observation allows us to take a given total preorder relation $\lesssim^*$ and define the successor relation of that total preorder, denoted $\lesssim^+$, using first-order logic

[1] A total preorder is a transitive, reflexive and total binary relation.

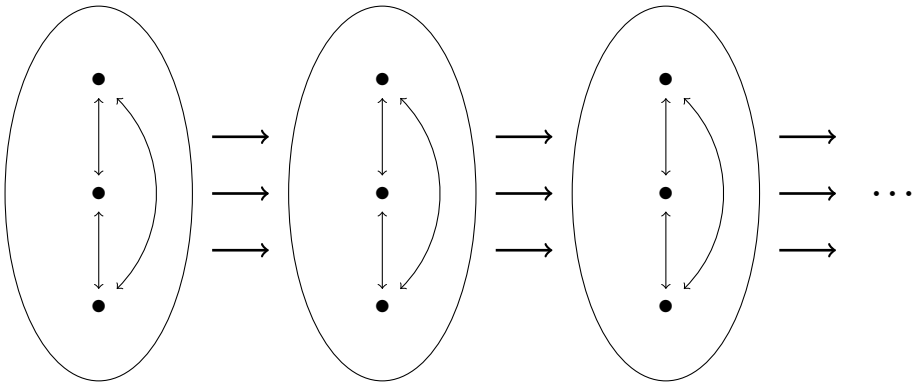


Figure 2.3: A sketch of a model of a total preorder

and three variables x, y and z, in the following way.

$$\forall x \forall y (\lesssim^{+}(x,y) \leftrightarrow (\lesssim^{*}(x,y) \wedge \neg \exists z (\lesssim^{*}(x,z) \wedge \lesssim^{*}(z,y) \wedge z \neq x \wedge z \neq y)))$$

This sentence states that the successor of a total preorder holds between the elements of a model only if there exist no intermediary elements between the given pair of elements with respect to the total preorder relation. As a result, the successor of a total preorder $\lesssim^{+}$ does not have the reflexivity and transitivity properties.

The idea of differentiating between a given relation and the successor of that relation with the superscripts $*$ and $+$ comes from [Benaim et al., 2013] and [Figueira et al., 2014], although in these papers the superscripts are used in a slightly different way than here. Whenever there are results about extending two-variable first-order logic with a certain kind of relation and its successor, the two types of relations are annotated with the superscripts $*$ and $+$ symbols, respectively. This helps to distinguish between a certain relation that has "global reach" in the model due to the inherent transitivity property (hence, the symbol $*$) and the successor version of that relation that only has "local reach" and for which the transitivity property does not hold (hence, the symbol $+$).

The successor relation of a total preorder $\lesssim^{+}$ allows one to check whether an element of a model is immediately preceded with respect to the global linear ordering by another element of the model. That is, element a is the successor of element b, whenever the equivalence class of a is the next equivalence class after the equivalence class of b. It follows that every element of a model has all the elements from the next equivalence class as its successors.

It is not possible to check whether one element is a successor of another element with respect to the total preorder in the language of $\mathcal{FO}^2[\lesssim^*]$. Otherwise, there would exist a sentence θ in the language of $\mathcal{FO}^2[\lesssim^*]$ such that it was possible to define the successor of a total preorder $\lesssim^+$ in the following way.

$$\forall x \forall y (\lesssim^+(x,y) \leftrightarrow (\lesssim^*(x,y) \wedge \theta(x,y)))$$

At the same time, the following sentence in $\mathcal{FO}^2[\lesssim^*]$ can be used to restrict the interpretation of a total preorder relation $\lesssim^*$ to a linear order, by reducing the size of every equivalence class in the model to one element.

$$\forall x \forall y (\lesssim^*(x,y) \wedge \lesssim^*(y,x) \rightarrow x = y)$$

Therefore, taking the above assumption into account, it would be possible to use the sentence θ to define the successor of a linear order relation from the given linear order relation in two-variable first-order logic. This, however, is impossible (for details see the footnote 3 in the Subsection 2.5 dedicated to two-variable logics with linear orders).

Hence, the two-variable fragments of first-order logic with total preorder relations are neither contained in nor contain the corresponding two-variable fragments with the total preorder successor relations.

Extending monadic two-variable first-order logic $\mathcal{FO}^2[]$ with the successor relation of a total preorder $\lesssim^+$ gives rise to a new language, denoted $\mathcal{FO}^2[\lesssim^+]$. The finite satisfiability of $\mathcal{FO}^2[\lesssim^+]$ is EXPSPACE-complete [Manuel and Zeume, 2013]. It is possible to extend $\mathcal{FO}^2[]$ with both the total preorder relation and the successor of that total preorder relation. The complexity of the finite satisfiability of this logic, denoted $\mathcal{FO}^2[\lesssim^+, \lesssim^*]$, is also EXPSPACE-complete [Schwentick and Zeume, 2012]. On the other hand, when a total preorder and the successor of another total preorder are both added to $\mathcal{FO}^2[]$, the finite satisfiability of the logic, denoted $\mathcal{FO}^2[\lesssim_1^+, \lesssim_2^*]$, becomes undecidable [Manuel and Zeume, 2013]. Undecidability of the finite satisfiability problem also holds in the case, when $\mathcal{FO}^2[]$ is extended with two successors of two different total preorders, denoted $\mathcal{FO}^2[\lesssim_1^+, \lesssim_2^+]$ [Manuel and Zeume, 2013].

Unfortunately, the satisfiability problems of the extensions of monadic two-variable first-order logic with total preorders and the successors of total preorders

have not been studied. The same applies for the satisfiability and finite satisfiability problem of the extensions of two-variable first-order logic with total preorders and the successors of total preorders.

The relationship between the complexity classes of the finite satisfiability problem of two-variable logics with total preorder relations and the successors of total preorder relations is shown in Figure 2.4.

The diagram with the complexities of satisfiability of two-variable and monadic two-variable logics with total preorder relations is omitted, because there are not enough complexity results available in this area for a meaningful graphical comparison.

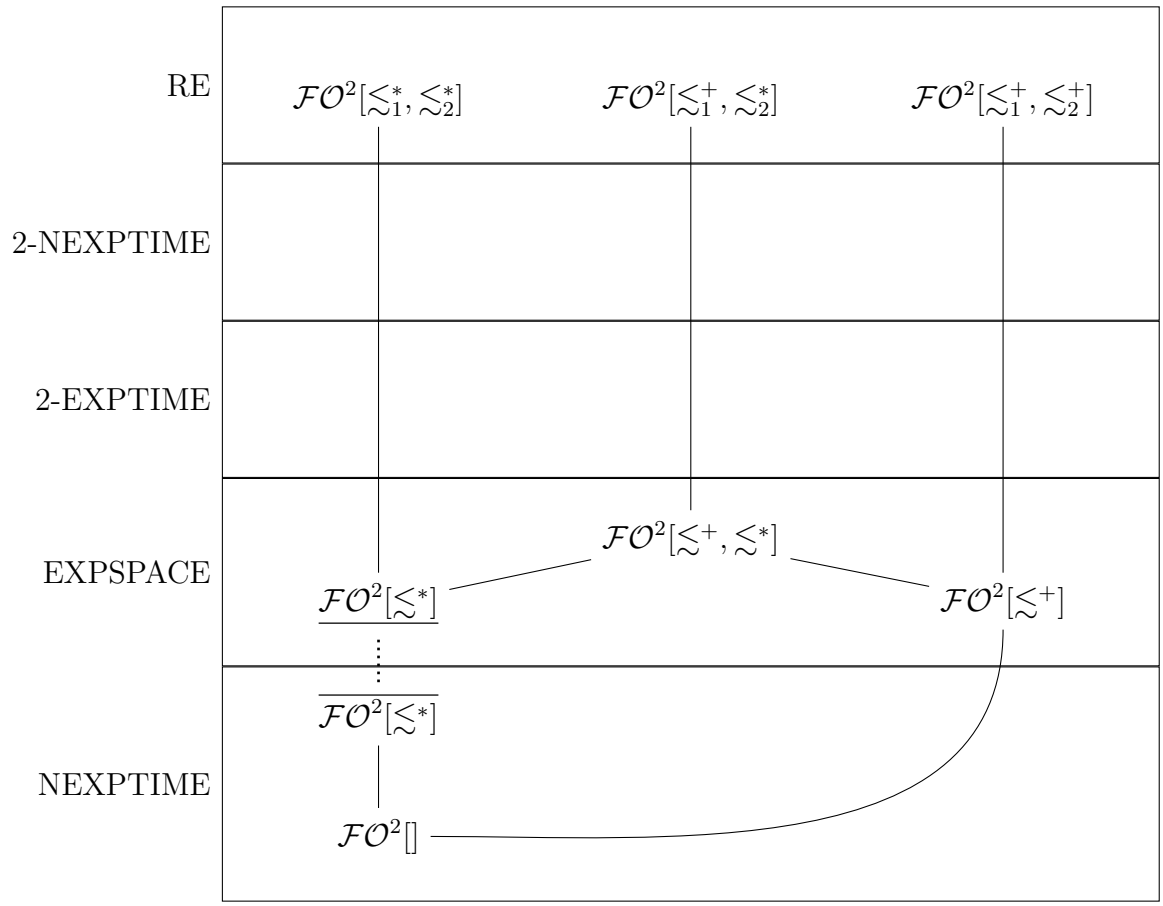


Figure 2.4: Finite satisfiability of monadic two-variable first-order logic with total preorder relations

2.4 Two-variable fragment with equivalence relations

In the previous section it was shown how the restriction of a transitive relation with reflexivity and totality gives rise to a total preorder. Another way to restrict a transitive relation is by adding reflexivity and symmetry properties to form an equivalence relation.

Just like a total preorder, an equivalence relation partitions the model universe into a set of equivalence classes. Figure 2.5 provides a sketch of how an equivalence relation relates the elements of a model to each other. The elipses in the diagram represent individual equivalence classes, within which all elements are related to each other by an equivalence relation.

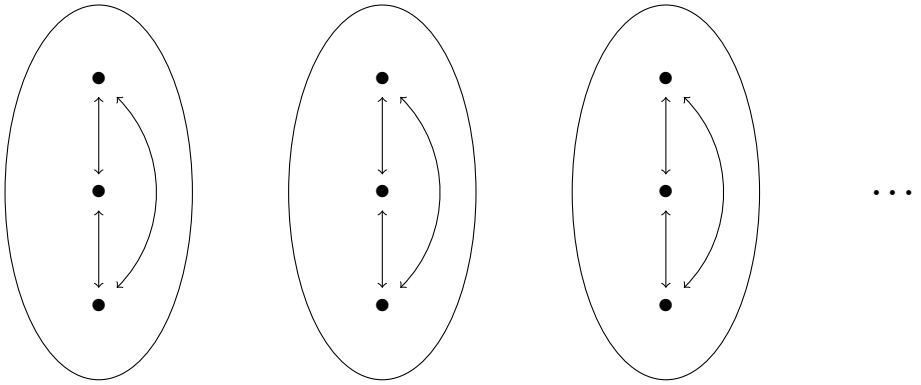


Figure 2.5: A sketch of a model of an equivalence relation

For the purposes of extending two-variable first-order logic, an equivalence relation is strictly less expressive than a total preorder relation. It is possible to use both an equivalence relation $\sim$ and a total preorder relation $\lesssim^*$ in $\mathcal{FO}^2()$ to test whether any two elements of a model belong to the same equivalence class or not, as the following sentence demonstrates.

$$\forall x \forall y (\sim(x, y) \leftrightarrow (\lesssim^*(x, y) \wedge \lesssim^*(y, x))$$

At the same time, unlike a total preorder relation, an equivalence relation does not order elements of the model across the equivalence classes. Hence, it is not possible to determine the relative order of elements from different equivalence classes.

Therefore, it is expected that the substitution of a total preorder relation with

an equivalence relation leads to logics with lower computational complexity of the satisfiability and finite satisfiability problems.

This supposition is confirmed by the fact that the complexity of the finite satisfiability of monadic two-variable logic with two total preorders $\mathcal{FO}^2[\lesssim_1^*, \lesssim_2^*]$ is undecidable, whereas the complexity of the finite satisfiability of monadic two-variable logic with two equivalence relations $\mathcal{FO}^2[\sim_1, \sim_2]$ is 2-NEXPTIME-complete. The same complexity bound holds for the satisfiability of $\mathcal{FO}^2[\sim_1, \sim_2]$. The satisfiability and finite satisfiability of two-variable first-order logic with two equivalence relations, denoted $\mathcal{FO}^2(\sim_1, \sim_2)$, is 2-NEXPTIME-complete too [Kieronski et al., 2012].

Unlike $\mathcal{FO}^2[\lesssim^*]$, monadic two-variable first-order logic with one equivalence relation, denoted $\mathcal{FO}^2[\sim]$, has the finite model property. The finite model property also holds for $\mathcal{FO}^2(\sim)$. In fact, due to the exponential small model property of $\mathcal{FO}^2[\sim]$ and $\mathcal{FO}^2(\sim)$ the satisfiability and finite satisfiability problems of both logics are in NEXPTIME [Kieronski and Otto, 2012].

The addition of three equivalence relations to monadic two-variable first-order logic results in the language, denoted $\mathcal{FO}^2[\sim_1, \sim_2, \sim_3]$, with undecidable satisfiability and finite satisfiability [Kieronski and Otto, 2012].

The problem of determining the complexity of satisfiability and finite satisfiability of the extensions of two-variable first-order logic and monadic two-variable first-order logic with equivalence relations has been fully resolved. Tight complexity bounds for all possible numbers of equivalence relations have been obtained.

Additionally, it was established that satisfiability and finite satisfiability of monadic two-variable first-order logic with one transitive relation and one equivalence relation, denoted $\mathcal{FO}^2[<, \sim]$, are undecidable [Kieronski and Tendera, 2009]. This result is an improvement on the undecidability of satisfiability and finite satisfiability of $\mathcal{FO}^2[<_1, <_2]$, since the equivalence relation $\sim$ from $\mathcal{FO}^2[<, \sim]$ can always be expressed using the second transitive relation $<_2$ from $\mathcal{FO}^2[<_1, <_2]$ by adding reflexivity and symmetry properties to the transitive relation $<_2$ with the following sentence.

$$\forall x <_2(x, x) \wedge \forall x \forall y(<_2(x, y) \rightarrow <_2(y, x))$$

Figure 2.6 demonstrates the relationships between the complexity classes of

the finite satisfiability problem of two-variable first-order logic and monadic two-variable first-order logic with equivalence relations. The diagram with the complexities of satisfiability of two-variable and monadic two-variable logics with equivalence relations is omitted, because such complexity results duplicate the finite satisfiability results of these logics.

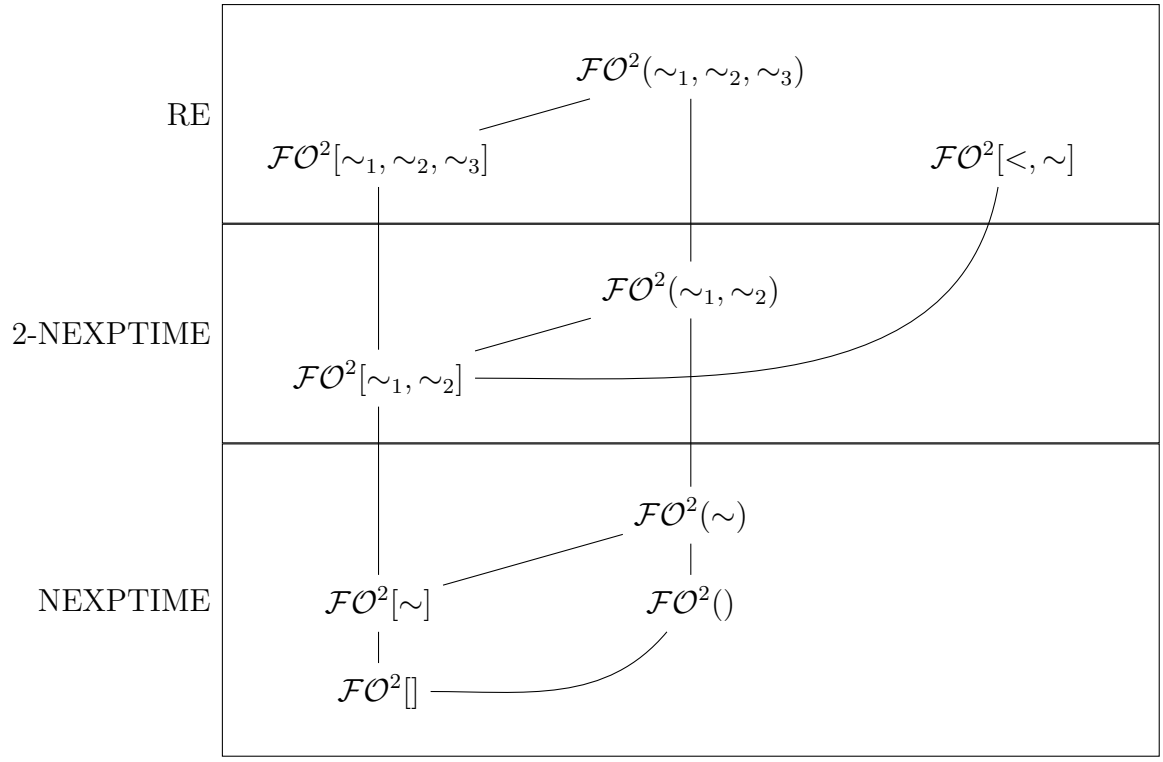


Figure 2.6: Finite satisfiability of two-variable and monadic two-variable first-order logic with equivalence relations

2.5 Two-variable fragment with linear order relations

It is often desirable to restrict a total preorder in such a way that the relative ordering of the elements is preserved. By imposing the antisymmetry restriction on a total preorder relation, a linear order relation is obtained. A linear order [2] is equivalent to a total preorder in which every equivalence class contains exactly one element.

[2] A linear order is a total reflexive antisymmetric transitive binary relation.

Just like in the case with the equivalence relation, the two-variable fragment of first-order logic with a linear order relation is strictly less expressive than the corresponding fragment with the total preorder relation. This is because a total preorder relation $\lesssim^*$ can always be converted to a linear order relation $\leq^*$ by adding the following axiom.

$$\forall x \forall y((\lesssim^*(x,y) \wedge \lesssim^*(y,x)) \rightarrow x = y)$$

Two-variable fragments of first-order logic extended with a linear order do not have the finite model property. This is because the following sentence can be written in two-variable first-order logic with the help of the linear order predicate $\leq^*$ that is satisfied only by infinite models.

$$\forall x \exists y(\leq^*(x,y) \wedge x \neq y)$$

The finite model property states that if a given sentence is satisfied in an infinite model, there exists a finite model that also satisfies the sentence. Therefore, satisfiability and finite satisfiability problems for the two-variable fragments of first-order logic with linear orders do not coincide, as determining the existence of an infinite model that satisfies a given sentence does not imply the existence of a finite model that also satisfies the sentence.

Similarly to how the successor of a total preorder was defined in Section 2.3, the successor relation $\leq^+$ of a linear order $\leq^*$ can be defined in first order logic with the following sentence.

$$\forall x \forall y(\leq^+(x,y) \leftrightarrow (\leq^*(x,y) \wedge \neg \exists z(\leq^*(x,z) \wedge \leq^*(z,y))))$$

However, it is not possible to define the successor relation in first-order logic with just two variables[3] and it is necessary to consider the satisfiability and finite satisfiability of logics with linear order relations and logics with the successors of

[3] If it was possible to define the successor relation of a given linear order $\leq^+$ in two-variable first-order logic, then the complexity of $\mathcal{FO}^2[\leq_1^+, \leq_2^+, \leq_1^*, \leq_2^*]$ would be the same as the complexity of $\mathcal{FO}^2[\leq_1^+, \leq_1^*, \leq_2^*]$. This is because, according to the hypothesis, it would be possible to define the missing relation $\leq_2^+$ from the relation $\leq_2^*$ in the latter logic, making the two logic expressively equivalent. However, the logic $\mathcal{FO}^2[\leq_1^+, \leq_2^+, \leq_1^*, \leq_2^*]$ was shown to be undecidable [Manuel, 2010], whereas $\mathcal{FO}^2[\leq_1^+, \leq_1^*, \leq_2^*]$ was shown to be in EXPSPACE [Schwentick and Zeume, 2012]. Therefore, the two logics do not have the same expressive power and the addition of the extra relation $\leq_2^+$ to the logic $\mathcal{FO}^2[\leq_1^+, \leq_1^*, \leq_2^*]$ makes the resulting logic strictly more expressive.

linear order relations separately.

The decidability of satisfiability and finite satisfiability has been resolved for a number of fragments of two-variable first-order logic with linear order relations. In cases, such as $\mathcal{FO}^2(\leq^+)$ [Charatonik and Witkowski, 2013], $\mathcal{FO}^2(\leq_1^+, \leq_2^+)$ [Charatonik and Witkowski, 2013], $\mathcal{FO}^2(\leq^*)$ [Otto, 2001] and $\mathcal{FO}^2(\leq_1^*, \leq_2^*, \leq_3^*)$ [Kieronski, 2011], the complexity classes of $\mathcal{FO}^2()$ with linear order relations match the complexities of the corresponding fragments of $\mathcal{FO}^2[]$.

Figure 2.7 demonstrates the relationships between the complexity classes of the finite satisfiability problem of monadic two-variable first-order logic with linear order relations and successors of linear order relations.

As can be seen from the diagram, four logics $\mathcal{FO}^2[\leq_1^+, \leq_2^*]$, $\mathcal{FO}^2[\leq_1^*, \leq_2^*]$, $\mathcal{FO}^2[\leq_1^+, \leq_2^+, \leq_1^*]$ and $\mathcal{FO}^2[\leq_1^+, \leq_1^*, \leq_2^*]$ do not have a precise complexity class established for them, as their lower complexity bound does not belong to the same complexity class as their upper complexity bound.

Among these logics of particular interest is the logic $\mathcal{FO}^2[\leq_1^+, \leq_2^+, \leq_1^*]$ that has EXPSPACE lower complexity bound and has its upper complexity bound marked as VAS. This means that the finite satisfiability problem of logic was shown to be equivalent to the problem of reachability in vector addition systems (VAS) in [Manuel and Zeume, 2013]. The satisfiability problem of many other logics and other decision problems in Computer Science were shown to be equivalent in terms of their complexity to VAS. For example, reachability in vector addition systems is equivalent in terms of complexity to the problem of reachability in Petri nets and emptiness of multi-counter automata. The latter problem was shown to be decidable in [Kosaraju, 1982]. Despite knowing that VAS is recursive, establishing the exact upper complexity bound for this famous decision problem has been unsuccessful for many years. Recently in [Leroux and Schmitz, 2015] the first concrete upper bound for VAS was shown to be cubic Ackermann. The best known lower complexity bound for VAS is EXPSPACE [Lipton, 1976], hence the same complexity class is assigned to the finite satisfiability of $\mathcal{FO}^2[\leq_1^+, \leq_2^+, \leq_1^*]$.

The diagram with the complexities of satisfiability of two-variable and monadic two-variable logics with linear order relations is omitted, because there are not enough complexity results available in this area for a meaningful graphical comparison.

An extensive study concerning the complexities of the satisfiability problems of $\mathcal{FO}^2[]$ with the combinations of linear orders and total preorders can be found in

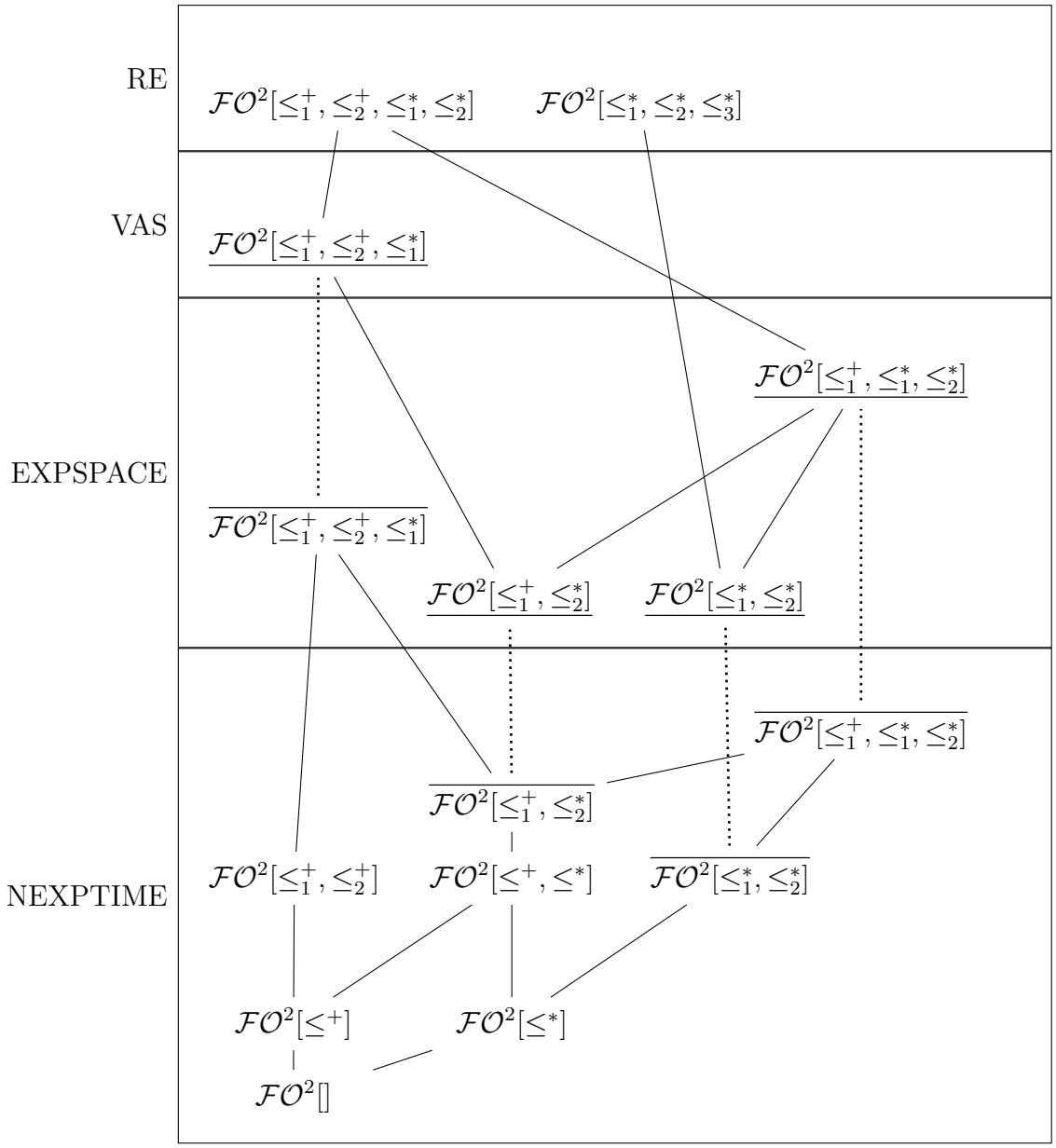


Figure 2.7: Finite satisfiability of monadic two-variable first-order logic with linear order relations

[Schwentick and Zeume, 2012], [Bojańczyk et al., 2006] and [Manuel et al., 2013]. Figure 2.8 summarises the complexity results obtained in these publications.

2.6 Two-variable fragment with tree navigation relations

In Section 2.5 it was discussed how total preorder relations generalise linear orders. Another useful type of restriction that can be placed on a binary relation is the requirement that the relation describes a tree. Formally speaking, a tree relation, denoted $\downarrow^*$, is a partial order such that there is a minimum element with respect to the ordering and for any given element of the model, the set of the elements that the given element relates to by $\downarrow^*$ is well-ordered.

Two-variable first-order logic $\mathcal{FO}^2()$ extended with one tree relation $\downarrow^*$ is called two-variable first-order logic over *unordered trees* and is denoted $\mathcal{FO}^2(\downarrow^*)$ (similarly for $\mathcal{FO}^2[]$). If, in addition to being extended with the $\downarrow^*$ relation, two-variable first-order logic is also extended with a relation $\rightarrow^*$ that for every node in the tree linearly orders the children of that node, then such a logic is called two-variable first-order logic over *ordered trees* and is denoted $\mathcal{FO}^2(\downarrow^*, \rightarrow^*)$ (similarly for $\mathcal{FO}^2[]$).

Figure 2.9 provides an example an interpretation of relation $\downarrow^*$ in a model consisting of elements from a to i: the solid arrows connect pairs of elements for which the relation $\downarrow^*$ holds. Figure 2.10 provides an example of an interpretation of relation $\downarrow^*$ and an interpretation of relation $\rightarrow^*$ in a model consisting of elements from a to i: the dashed arrows connect pairs of elements for which the relation $\rightarrow^*$ holds.

The two relations for navigating in tree graphs $\downarrow^*$ and $\rightarrow^*$ are called the vertical tree navigation relation and the horizontal tree navigation relation, respectively.

It is not possible to define a tree relation in first-order logic. This follows from the fact that connectivity of arbitrary graphs is not $\mathcal{FO}$-definable (Proposition 3.1 in [Libkin, 2004]) and it is possible to express the connectivity property using the vertical tree navigation relation.

This is achieved by stating that a given relation R describes a connected graph if and only if there exists a spanning tree as a subrelation of the symmetric closure

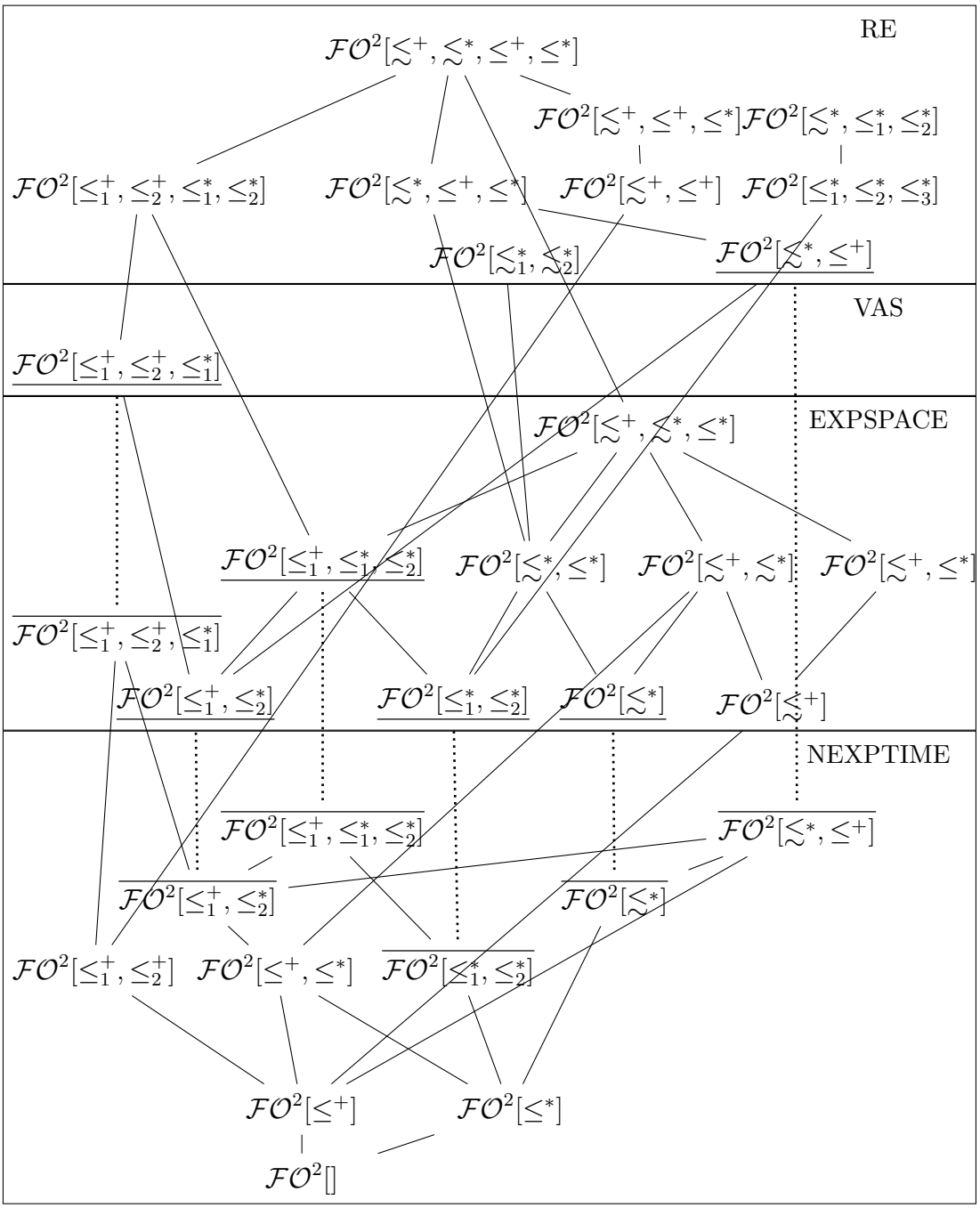


Figure 2.8: Finite satisfiability of monadic two-variable first-order logic with total preorder and linear order relations

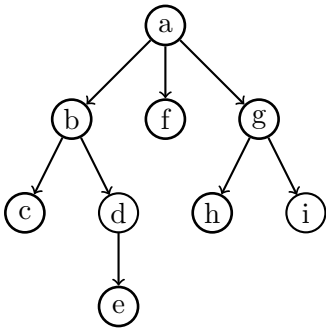


Figure 2.9: An example of relation $\downarrow^*$ interpreted as an unordered tree

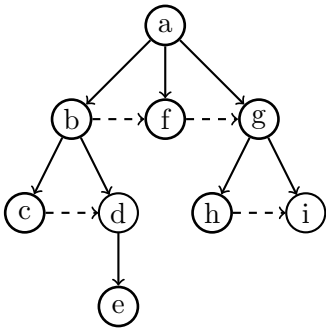


Figure 2.10: An example of relations $\downarrow^*$ and $\rightarrow^*$ interpreted as an ordered tree

of R. Such a statement can be encoded with the following sentence.

$$\forall x \forall y (R(x,y) \rightarrow R^{\#}(x,y))$$

$$\forall x \forall y (R^{\#}(x,y) \rightarrow (R(x,y) \vee (R(y,x)))$$

$$\forall x \forall y (\downarrow^*(x,y) \rightarrow R^{\#}(x,y))$$

Since by definition the tree relation reaches all the elements of the model, the interpretation of the relation R is describing a connected graph if and only if the interpretation of its symmetric closure $R^{\#}$ has a spanning tree as a substructure.

The combination of $\downarrow^*$ and $\rightarrow^*$ relations allows to express a linear order relation $\leq^*$ in two-variable first-order logic. To define a linear order relation $\leq^*$ in terms of the vertical tree navigation relation $\downarrow^*$ and the horizontal tree navigation relation $\rightarrow^*$ in $\mathcal{FO}^2()$, it is sufficient to make $\downarrow^*$ describe a tree in which every node has at most one child, as follows.

$$\forall x \neg \exists y \rightarrow^*(x,y) \tag{2.2}$$

Hence, the two-variable first-order fragments with vertical and horizontal tree navigation relations are strictly more expressive than the corresponding fragments with linear orders.

The vertical tree navigation successor relation $\downarrow^+$ and the horizontal tree navigation successor relation $\rightarrow^+$ are defined similarly to how the successor of a linear order relation $\leq^+$ is defined in terms of the linear order relation $\leq^*$. Evidently, the relations $\downarrow^+$ and $\rightarrow^+$ can express the relation $\leq^+$ in $\mathcal{FO}^2()$ as well, since the

sentence 2.2 can be reformulated as follows.

$$\forall x \neg \exists y \rightarrow^{+} (x, y)$$

Usually, vertical tree navigation successor relations and horizontal tree navigation successor relations are denoted simply $\downarrow$ and $\rightarrow$ in other publications of the research area concerning the satisfiability problem of two-variable first-order logic and its extensions. However, here the superscript $+$ is applied to these two relation symbols for the sake of consistency with the notation used in denoting other types of relations discussed previously in this chapter.

2.6.1 Complexity results

Even though the exact complexity classes for the two-variable fragment with tree navigation and equivalence relations remain unknown, the tight complexity bounds for $\mathcal{FO}^2[]$ with the all combinations of vertical and horizontal navigation predicates were established in [Benaim et al., 2013]. The publication also examines the complexity of all these fragments under the guarded restriction of $\mathcal{FO}^2[]$, denoted $\mathcal{GF}^2[]$. Additionally, this work studies the effect of the *unary alphabet restriction* on the complexity classes of the logics under consideration.

In the models of the sentences of $\mathcal{FO}^2[]$ with unary alphabet restriction every element of the model must be assigned exactly one unary predicate from the set of unary predicates that are featured in the signature of the given sentence that the model interprets. This restriction allows to treat the models of the sentences as data structures labelled by a finite alphabet, where each letter from the alphabet corresponds to a unary predicate from the signature of the sentence. More specifically, a sentence ϕ is satisfied by a model $\mathfrak{A}$ in $\mathcal{FO}^2[]$ with unary alphabet restriction if and only if $\mathfrak{A} \models \phi$ and for every element $e \in \mathfrak{A}$ there is exactly one unary predicate P_j from the signature of ϕ such that $\mathfrak{A} \models P_j(e_i)$. Alternatively, it can be said that the unary predicates from the signature of a given sentence must form a partition on the set of elements of the model that interprets that sentence.

For the sake of brevity, $\mathcal{FO}^2[]$ with unary alphabet restriction is called here *singular monadic two-variable first-order logic* and is denoted $\mathcal{FO}^2_S[]$.

Unary alphabet restriction lessens the expressive power of the logic it applies to, because it disallows multiple unary predicates from the signature of a given

sentence to hold for any element of the models of the sentence. At the same time, for any sentence ϕ of $\mathcal{FO}^2[]$ with unary predicates $P_1, ..., P_k$, unary alphabet restriction can be enforced on the models of ϕ using the following sentence of $\mathcal{FO}^2[]$:

$$\bigwedge \Big\{ \forall x\, P_i(x) \rightarrow \bigwedge \{ \neg\, P_j(x) | 0 \leq j \leq k, j \neq i \} | 0 \leq i \leq k \Big\}$$

$$\forall x \bigvee \{ P_i(x) | 0 \leq i \leq k \}$$

Hence, $\mathcal{FO}^2[]$ or any of its extensions with unary alphabet restriction are strictly less expressive than the corresponding logics without unary alphabet restriction. Similarly for $\mathcal{GF}^2[]$ with unary alphabet restriction.

There are two cases in which the presence of unary alphabet restriction affects the complexity class of the finite satisfiability of the fragment. Firstly, $\mathcal{GF}^2_S[\downarrow^*]$ is PSPACE-complete, whereas the corresponding logic without the unary alphabet restriction $\mathcal{GF}^2[\downarrow^*]$ is EXPSPACE-complete. Secondly, $\mathcal{FO}^2_S[\downarrow^*]$ is NEXPTIME-complete, whereas $\mathcal{FO}^2[\downarrow^*]$ is EXPSPACE-complete.

In all the other cases the complexity class of the finite satisfiability problem of an extension of $\mathcal{FO}^2[]$ with tree navigation relations is the same as the complexity class of the corresponding extension of $\mathcal{FO}^2_S[]$. When the complexity classes of a logic with and without unary alphabet restrictions match, the two logics are represented in the diagram by a single symbol $\mathcal{FO}^2_{(S)}[]$. Otherwise the logic with unary alphabet restriction is situated below the corresponding logic without unary alphabet restriction, because the former is strictly less expressive than the latter.

The fact that the complexities of both guarded and non-guarded monadic two-variable first-order logics has been determined makes it possible to evaluate the role of the guarded restriction in the presence of tree navigation relations.

For instance, $\mathcal{FO}^2[\downarrow^+]$ is NEXPTIME-complete, whereas $\mathcal{GF}^2[\downarrow^+]$ is EXPTIME-complete. Extending $\mathcal{FO}^2[\downarrow^+]$ and $\mathcal{GF}^2[\downarrow^+]$ with the horizontal tree navigation successor relation $\rightarrow^+$ does not change the complexities of the two logics, as the logic $\mathcal{FO}^2[\downarrow^+, \rightarrow^+]$ and $\mathcal{GF}^2[\downarrow^+, \rightarrow^+]$ remain NEXPTIME-complete and EXPTIME-complete, respectively. However, when the horizontal tree navigation relation $\rightarrow^*$ is added to $\mathcal{GF}^2[\downarrow^+]$, the complexity of the resulting logic $\mathcal{GF}^2[\downarrow^+, \rightarrow^*]$ jumps up a complexity class and becomes NEXPTIME-complete. At the same time extending $\mathcal{FO}^2[\downarrow^+]$ with $\rightarrow^*$ results in the logic $\mathcal{FO}^2[\downarrow^+, \rightarrow^*]$ that is in the same complexity class as both $\mathcal{FO}^2[\downarrow^+]$ and $\mathcal{FO}^2[\downarrow^+, \rightarrow^+]$.

As noted above, the complexity of $\mathcal{GF}^2_S[\downarrow^*]$ differs from the complexity $\mathcal{FO}^2_S[\downarrow^*$

]. In all the rest of the cases the presence of the guarded restriction does not affect the complexity bounds of the monadic two-variable first-order logic with tree navigation relations.

Finally, it is worth observing that $\mathcal{GF}^2[\downarrow^*]$ is EXPSPACE-hard and bounds all non-singular fragments that contain $\downarrow^*$ relation from below, whereas the most general logic $\mathcal{FO}^2[\downarrow^+, \downarrow^*, \rightarrow^+, \rightarrow^*]$ is in EXPSPACE and bounds of all the other fragments considered in this paper from above.

Figure 2.11 summarises the complexity results discussed above. The diagram with the complexities of satisfiability of two-variable and monadic two-variable logics with tree navigation relations is omitted, because there are not enough complexity results available in this area for a meaningful graphical comparison.

2.6.2 $\mathcal{FO}^2[]$ with navigation in trees and XPath

XPath is a query language for XML documents and Core-XPath is its navigational fragment [Gottlob et al., 2005]. The study of the properties of Core-XPath is relevant in the context of $\mathcal{FO}^2[]$ with vertical and horizontal tree navigation relations and their successors because, according to [Bojańczyk et al., 2009], in the research on XML and the associated query languages, XML documents are viewed as ordered trees that have a label from a finite set of labels attached to each of its nodes. This makes it possible to interpret XML documents as models of the sentences of $\mathcal{FO}^2[]$ with tree navigation relations, and vice versa. The connection between Core-XPath and $\mathcal{FO}^2[]$ with vertical and horizontal tree navigation relations and their successors was first established in [Marx and de Rijke, 2005]. According to this publication, Core-XPath is equivalent to the monadic two-variable first-order logic $\mathcal{FO}^2[]$ with $\downarrow^*$, $\downarrow^+$, $\rightarrow^*$ and $\rightarrow^+$ relations, denoted $\mathcal{FO}^2[\downarrow^*, \downarrow^+, \rightarrow^*, \rightarrow^+]$.

The complexity results on Core-XPath have been related to the complexity results on $\mathcal{FO}^2[]$ with navigation in trees. In [Marx, 2004] EXPTIME-completeness has been shown for the satisfiability of Core-XPath. Because the translation of a Core-XPath query into the language of $\mathcal{FO}^2[\downarrow^+, \downarrow^*, \rightarrow^+, \rightarrow^*]$ produces a formula of at most exponential size, the 2-EXPTIME complexity upper bound can be immediately derived for the finite satisfiability of the $\mathcal{FO}^2[\downarrow^+, \downarrow^*, \rightarrow^+, \rightarrow^*]$ fragment.

The extension to Core-XPath with the ability to test the attribute values of the selected XML nodes for equality is called Core-Data-XPath. The question

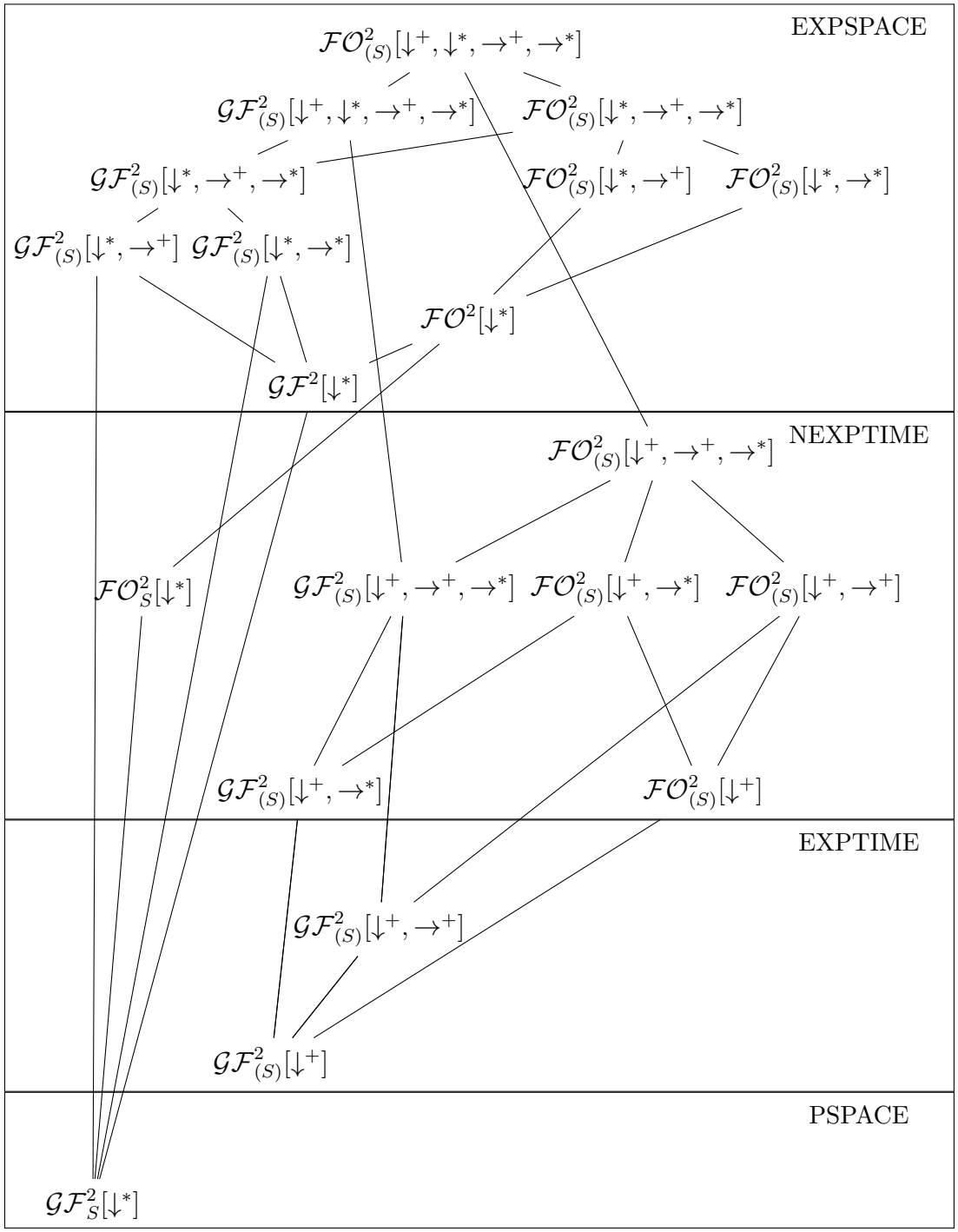


Figure 2.11: Finite satisfiability of two-variable first-order logic with tree navigation relations

of the decidability of Core-Data-XPath was answered negatively in [Geerts and Fan, 2005]. The logical characterisation of Core-Data-XPath in terms of $\mathcal{FO}^2_S[]$ with the four tree navigation predicates $\downarrow^+, \downarrow^*, \rightarrow^+, \rightarrow^*$ and one equivalence relation $\sim$ was described in [Bojańczyk et al., 2009]. This extension of the monadic two-variable first-order logic is strictly less expressive than the XPath fragment it models. At the same time the authors of [Geerts and Fan, 2005] managed to show that the finite satisfiability of $\mathcal{FO}^2_S[\sim, \downarrow^+, \downarrow^*, \rightarrow^+, \rightarrow^*]$ is as hard as the nonemptiness problem of vector addition tree automata. Because the status of the decidability of nonemptiness of vector addition tree automata remains unresolved for several years, finding the upper bound for $\mathcal{FO}^2_S[\sim, \downarrow^+, \downarrow^*, \rightarrow^+, \rightarrow^*]$ is considered to be a difficult research task. The attempt to establish the exact complexity class for the less expressive logic $\mathcal{FO}^2_S[\downarrow^+, \rightarrow^+, \sim]$ was unsuccessful and as of today the finite satisfiability of this fragment is only known to be in 3-NEXPTIME and NEXPTIME-hard [Bojańczyk et al., 2009].

2.7 Two-variable fragment with counting

In the previous sections it was shown how it is possible to increase the expressiveness of the two-variable first-order logic by adding special binary relations that are not definable in $\mathcal{FO}^2()$. Another way to extend the reasoning capabilities of $\mathcal{FO}^2()$ is by allowing the counting quantifiers to appear in the sentences of the logic. The two-variable first-order logic with counting quantifiers, denoted $\mathcal{C}^2()$, consists of the sentences of $\mathcal{FO}^2()$ in which three additional existential quantifiers $\exists_{\leq n}$, $\exists_{=n}$, $\exists_{\geq n}$ can be used. The three counting quantifiers read, correspondingly, as "there exists less than or equal to n elements", "there exists exactly n element" and "there exists equal to or more than n element".

The decidability of satisfiability and finite satisfiability of $\mathcal{C}^2()$ was first demonstrated in [Grädel et al., 1997b]. NEXPTIME upper bound of satisfiability for unary encoding of numbers in counting quantifiers was proved in [Pacholski et al., 1997]. It was also shown in this publication that models of doubly-exponential size can be enforced by the sentences of $\mathcal{C}^2()$, yet the complexity of its finite satisfiability problem is in NEXPTIME. The same complexity upper bound for satisfiability and finite satisfiability of $\mathcal{C}^2()$ with binary encoding of numbers in counting quantifiers was shown in [Pratt-Hartmann, 2005]. Later, a simplified proof of the NEXPTIME upper bound of satisfiability and finite satisfiability

of $\mathcal{C}^2()$ was presented in [Pratt-Hartmann, 2010]. The NEXPTIME complexity upper bound is tight, since $\mathcal{C}^2()$ is strictly more expressive than $\mathcal{FO}^2()$ and the latter is known to be NEXPTIME-hard.

2.7.1 Expressiveness of $\mathcal{C}^2()$

It is only possible to simulate certain uses of counting quantifiers in $\mathcal{FO}^2()$. For instance, the following sentence shows how to enforce the existence of n realisations of a certain formula ϕ in $\mathcal{FO}^2()$ using a set L of n labels[4], rather than the existential quantifier $\exists_{\geq n}$.

$$\exists_{\geq n} x\, \phi(x) \equiv \bigwedge\{\exists x l_i(x) | l_i \in L\} \wedge \bigwedge\{\forall x(l_i(x) \rightarrow \phi(x) | l_i \in L\}$$

The first conjunct of the sentence makes sure that each label from the set L holds for at least one element. The second conjunct of the above sentence makes sure that for every element with a label from the set L, the given formula $\phi(x)$ holds as well. Since the labels from the set L are mutually exclusive, every model satisfying the above sentence must contain at least n elements for which the formula $\phi(x)$ holds.

The addition of counting quantifiers to the two-variable first-order logic, however, allows to express a number of useful properties that cannot be enforced in $\mathcal{FO}^2()$. An example of this is the statement that a given binary relation R can only be interpreted as a partial function. The following sentence encodes this requirement using counting quantifiers in a straightforward manner.

$$\forall x \exists_{\leq 1} y\, R(x, y)$$

In Section 2.5 it was demonstrated how a linear order relation $\leq^*$ can be simulated by one vertical tree navigation predicate $\downarrow^*$ and one horizontal tree navigation predicate $\rightarrow^*$ (or, alternatively, its successor $\rightarrow^+$). Using counting

[4]A set of labels is, basically, a set of formulas over one free variable such that each formula is mutually incompatible (from the satisfiability point of view) with every other formula from the set. Thus, no two formulas from the set of labels can hold for the same element in any model at the same time. For a more precise definition of a set of labels and how it is constructed refer to Chapter 3

quantifiers, $\downarrow^*$ can be turned into a linear order even in the absence of the horizontal tree navigation relations.

$$\forall x \exists_{\leq 1} y \downarrow^*(x, y) \tag{2.3}$$

2.7.2 Complexity results

Combining the two-variable first-order fragment with counting quantifiers results in a more expressive logic and may lead to the increase in the complexity of the finite satisfiability problem.

For instance, $\mathcal{FO}^2(\sim_1, \sim_2)$ is 2-NEXPTIME-complete, whereas $\mathcal{C}^2(\sim_1, \sim_2)$ is undecidable. The fragment $\mathcal{FO}^2(\leq^*)$ is NEXPTIME-complete, whereas $\mathcal{C}^2(\leq^*)$ is equivalent in terms of computational complexity to the problem of reachability in vector addition systems (VAS-complete), which means it is at least EXPSPACE-hard. The logic $\mathcal{FO}^2[\leq_1^*, \leq_2^*]$ is EXPSPACE-complete, whereas $\mathcal{C}^2(\leq_1^*, \leq_2^*)$ is undecidable, although in this case the increase in the complexity may not only be due to the addition of the counting quantifiers, but also due to the fact that $\mathcal{FO}^2[\leq_1^*, \leq_2^*]$ is monadic and $\mathcal{C}^2(\leq_1^*, \leq_2^*)$ is not (no separate hardness result is known for $\mathcal{C}^2[\leq_1^*, \leq_2^*]$).

At the same time, $\mathcal{FO}^2(\downarrow^+)$, $\mathcal{FO}^2[\leq^+, \leq^*]$ and $\mathcal{FO}^2(\sim)$ are all NEXPTIME-complete, and so are their extensions with the counting quantifiers. Interestingly, alleviating the monadic restriction from the NEXPTIME-complete logic $\mathcal{C}^2[\leq^+, \leq^*]$ results in the logic $\mathcal{C}^2(\leq^+, \leq^*)$ that is VAS-complete.

It is not possible to compare the complexity of the finite satisfiability of $\mathcal{C}^2(<)$ and $\mathcal{FO}^2(<)$, because the decidability of finite satisfiability of the latter fragment remains open. However, it is known that the satisfiability problem of $\mathcal{FO}^2(<)$ is in 2-NEXPTIME, and this contrasts with the undecidability of satisfiability of $\mathcal{C}^2(<)$.

Figure 2.12 summarises the finite satisfiability complexity results discussed above. Only the logics with counting, for which a complexity result is known, are mentioned in the diagram along with their non-counting variants. Other two-variable fragments without counting are not displayed for the sake of readability.

The diagram with the complexities of satisfiability of two-variable logics with counting is omitted, because there are not enough complexity results available in this area for a meaningful graphical comparison.

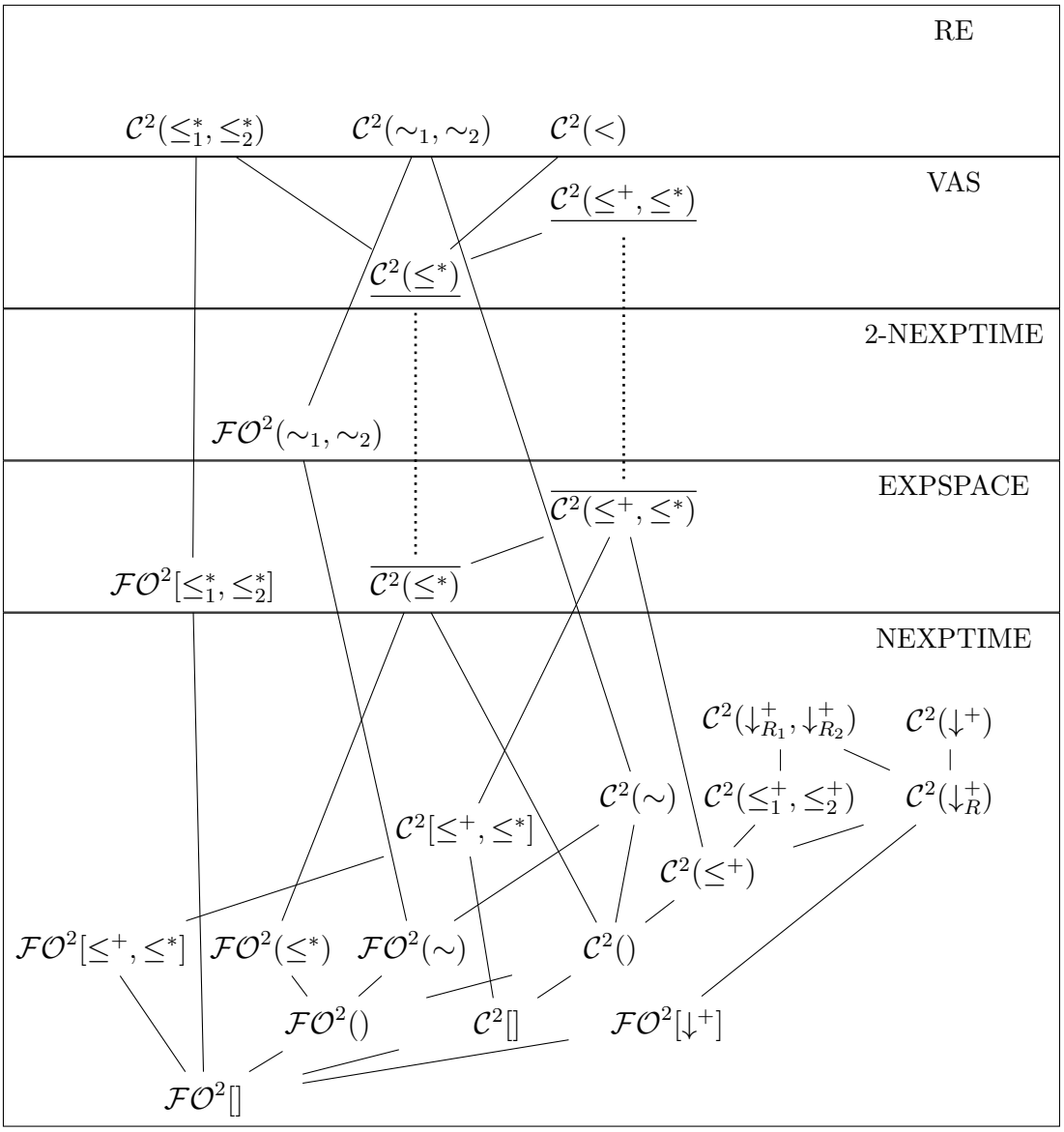


Figure 2.12: Finite satisfiability of two-variable first-order logic with counting

2.7.3 Two-variable first-order logic with counting and tree navigation predicates

In [Charatonik and Witkowski, 2013] the finite satisfiability of $\mathcal{C}^2()$ with the successors of two vertical *ranked* tree navigation relations $\downarrow^+_{R_1}$ and $\downarrow^+_{R_2}$, denoted $\mathcal{C}^2(\downarrow^+_{R_1}, \downarrow^+_{R_2})$, was shown to be decidable in NEXPTIME. Given a sentence ϕ and two natural number r_1 and r_2, it is said that ϕ is (finitely) satisfied in $\mathcal{C}^2(\downarrow^+_{R_1}, \downarrow^+_{R_2})$ if and only if there exists a (finite) model $\mathfrak{A}$ such that $\mathfrak{A} \models \phi$, the predicates $\downarrow^+_{R_1}$ and $\downarrow^+_{R_2}$ are interpreted in $\mathfrak{A}$ as the successors of two independent vertical tree navigation relations and the maximum number of children of every node in the trees described by $\downarrow^+_{R_1}$ and $\downarrow^+_{R_2}$ are r_1 and r_2 respectively.

The result of [Charatonik and Witkowski, 2013] not only demonstrated the NEXPTIME upper complexity bound for the finite satisfiability of $\mathcal{C}^2(\downarrow^+_{R_1}, \downarrow^+_{R_2})$, but also the NEXPTIME upper bound for the finite satisfiability of $\mathcal{FO}^2[\leq^+_1, \leq^+_2]$, which at the time of the publication of [Charatonik and Witkowski, 2013] was unknown. The fact that $\mathcal{C}^2(\downarrow^+_{R_1}, \downarrow^+_{R_2})$ is strictly more expressive than $\mathcal{FO}^2[\leq^+_1, \leq^+_2]$ follows from the same argument that was described in Subsection 2.7.1 and demonstrated by Equation 2.3.

The requirement to know the width or the rank of the tree in advance is limiting the expressive power of relation $\downarrow^+_R$ as compared to the normal successor of a vertical tree navigation relation $\downarrow^+$. This is because any sentence of the two-variable first-order logic with counting and vertical tree navigation relations, denoted $\mathcal{C}^2(\downarrow^+)$, can easily be converted to a sentence of the two-variable first-order logic with counting and vertical ranked tree navigation relations that has the same satisfiability status by adding the following axiom to the given sentence.

$$\forall x \exists_{\leq k} \downarrow^+ (x, y)$$

For a given rank of the tree r the above sentence restricts all interpretations of the successor of the vertical tree navigation relation to the successor of the vertical navigation in r-ranked trees. Because of this, the proof of the NEXPTIME upper bound for the finite satisfiability of $\mathcal{C}^2(\downarrow^+)$ that is found in Chapter 4 subsumes the NEXPTIME complexity result for the finite satisfiability of $\mathcal{C}^2(\downarrow^+_R)$ that follows from the work of [Charatonik and Witkowski, 2013].

Besides the complexity of the finite satisfiability of $\mathcal{C}^2(\downarrow^+_{R_1}, \downarrow^+_{R_2})$ established in [Charatonik and Witkowski, 2013] or $\mathcal{C}^2(\downarrow^+)$ established in this thesis, no

other satisfiability or finite satisfiability results about the extensions of $\mathcal{C}^2()$ or $\mathcal{C}^2[]$ with vertical or horizontal tree navigation relations or their successors are known.

Chapter 3

Terminology

This chapter contains the definition of the logic $\mathcal{C}^2(\downarrow^+)$, for which the decidability of finite satisfiability is proved to be in NEXPTIME in the next chapter. Terminological apparatus that is necessary for the presentation of that proof is laid out here as well.

For a set of elements S, for a binary relation R on S, the *graph* of R consists of the set of vertices S and the set of edges $\{(e_1, e_2)|e_1 \in S, e_2 \in S, R(e_1, e_2)\}$. For a given model $\mathfrak{A}$, the *graph* of $\mathfrak{A}$ is the graph of the binary predicate $\downarrow^+$ interpreted in $\mathfrak{A}$ (when $\downarrow^+$ is not in the signature of $\mathfrak{A}$, the graph of $\mathfrak{A}$ contains no edges). A tree graph is a directed acyclic connected graph such that there exists exactly one node in that graph that has no incoming edges and every other node in that graph has exactly one incoming edge. A forest is a directed graph in which every weakly connected component is a tree graph. A given model $\mathfrak{A}$ is called *a forest model* when the graph of $\mathfrak{A}$ is a forest.

The set of sentences of two-variable first-order logic with counting quantifiers and local navigation in forests, denoted $\mathcal{C}^2(\downarrow^+)$, coincides with the set of sentences of two-variable first-order logic with counting quantifiers $\mathcal{C}^2()$. A given sentence $\phi \in \mathcal{C}^2()$ is satisfied by a finite model $\mathfrak{A}$ in $\mathcal{C}^2(\downarrow^+)$ if and only if ϕ is satisfied by $\mathfrak{A}$ in $\mathcal{C}^2()$ and $\mathfrak{A}$ is a forest model.

This definition of $\mathcal{C}^2(\downarrow^+)$ is equivalent to the way the combination of two-variable first-order logic with counting quantifiers and one successor of a vertical tree navigation relation is presented in Section 2.7.3 of the Background chapter. For the remainder of this section we assume that ϕ is a sentence of $\mathcal{C}^2()$ and $\mathfrak{A}$ is a finite model that satisfies ϕ in $\mathcal{C}^2(\downarrow^+)$.

The sentence ϕ is said to be *in normal form*, if it matches the following pattern.

$$\forall x \forall y(\alpha(x,y) \vee x = y) \wedge \bigwedge_{1 \leq i \leq M} \forall x \exists_{\triangleleft c_i} y(\beta_i(x,y) \wedge x \neq y) \wedge \bigwedge_{1 \leq j \leq K} \exists_{\triangleleft d_j} x \gamma_j(x)$$

In this context $\alpha(x,y)$, $\beta_i(x,y)$ and $\gamma_j(x)$ are formulas of $\mathcal{C}^2()$ that do not contain any quantifiers or equality symbols, $\triangleleft$ stands for either $\leq$, $\geq$, or $=$, c_i and d_j are positive natural numbers expressed using binary coding, M is the total number of conjuncts in ϕ with the $\forall x \exists y_{\triangleleft c_i}$ quantifier prefix and K is the total number of conjuncts in ϕ with the $\exists x_{\triangleleft d_j}$ quantifier prefix. For each given sentence ϕ the value of number Z is defined to be equal to $C \cdot M$, where $C = max(\{c_i | 1 \leq i \leq M\})$.

For any sentence of $\mathcal{C}^2()$ it is always possible to find another sentence in normal form in polynomial time such one sentence is satisfiable in $\mathcal{C}^2()$ if and only if the other sentence is satisfiable in $\mathcal{C}^2()$ [Grädel and Otto, 1999]. For the remainder of this section we assume that ϕ is in normal form.

The *signature* of a given model $\mathfrak{A}$ is the set of all predicates featured in $\mathfrak{A}$ and is denoted $\Sigma(\mathfrak{A})$. The *signature* of ϕ, written as $\Sigma(\phi)$, is the set of unary and binary relation symbols that appear in ϕ. The size of ϕ is denoted $|\phi|$.

The *rough size* of ϕ is equal to $|\Sigma(\phi)|$ plus the size of the binary encoding of the sum of all c_i and d_i (the sum of the values of the numbers c_i and d_i, not the sum of the sizes of the binary coding of these values) in ϕ. The following lemma follows from the proof of Theorem 1 in [Pratt-Hartmann, 2010].

Lemma 1. *Given a sentence $\phi \in \mathcal{C}^2()$, the finite satisfiability of ϕ can be determined in nondeterministic exponential time with respect to its rough size.*

For a given number n a normal form sentence $\phi \in \mathcal{C}^2$ can be of exponential length with respect to n and at the same time have a polynomial rough size with respect to n. This can be possible when the size of the signature of ϕ is polynomial with respect to n and every value c_i and d_i that appears in the counting quantifiers of ϕ is at most exponential with respect to n. This fact about the notion of rough size is going to be crucial in proving the upper complexity bound of the finite satisfiability algorithm of $\mathcal{C}^2(\downarrow^+)$ in the next chapter. In particular, for a given sentence of $\mathcal{C}^2(\downarrow^+)$ of length n an auxiliary sentence is going to be constructed as part of the finite satisfiability algorithm run such that the length of the auxiliary sentence is exponential with respect to n and the rough size of the auxiliary sentence is polynomial with respect to n. Then, Lemma 1 is

going to be envoked to determine the satisfiability status of this auxiliary sentence in at most nondeterministic exponential amount of time with respect to n, which is essential for getting a tight upper complexity bound on the running time of the finite satisfiability algorithm.

For a given n-ary predicate R and some given n-tuple of variables $\overline{x}$, a *literal* is the formula $R(\overline{x})$ or the formula $\neg R(\overline{x})$.

A *1-type* of ϕ is any maximally consistent set of literals over the tuples of variables (x) and (x, x) and the set of unary and binary predicates from the signature $\Sigma(\phi)$. For example, if the signature of a given sentence ϕ is $\{R^1, P^1, Q^2\}$, then the set $\{R(x), \neg P(x), \neg Q(x, x)\}$ is an example of a 1-type of ϕ. Another example of a 1-type of ϕ is $\{\neg R(x), \neg P(x), Q(x, x)\}$. The set of all 1-types over the signature $\Sigma(\phi)$ is denoted $\tau(\phi)$.

Given a model $\mathfrak{A}$ and an element of the model e, the 1-type of the element e consists of the set of all the unary literals R_u from the signature of $\mathfrak{A}$ such that $\mathfrak{A} \models R_u(e)$ and all the binary literals R_b from the signature of $\mathfrak{A}$ such that $\mathfrak{A} \models R_b(e, e)$. The 1-type of $e \in \mathfrak{A}$ is denoted $\mathrm{tp}^{\mathfrak{A}}[e]$. A 1-type τ_i is said to be *realised* in $\mathfrak{A}$ if there exists an element $e \in \mathfrak{A}$ such that $\mathrm{tp}^{\mathfrak{A}}[e] = \tau_i$.

For technical reasons that become apparent in Chapter 4, certain kinds of 1-types have to be distinguished in a given model. For any $\tau_i \in \tau(\phi)$, if there exists only one realisation of τ_i in $\mathfrak{A}$, then τ_i is called a *royal* 1-type. If there exist more than $5 \cdot Z$ realisations of τ_i in $\mathfrak{A}$, then τ_i is called a *populous* 1-type. The 1-types that are neither royal nor populous are called *noble*.

A *2-type* of ϕ is a maximally consistent set of literals over the tuples of variables (x), (y), (x, x), (y, y), (x, y) and (y, x) and the set of unary and binary predicates from the signature $\Sigma(\phi)$, excluding the literal $x = y$. For example, if the signature of a given sentence ϕ is $\{R^1, P^1, Q^2\}$, then the set $\{R(x), \neg P(x), \neg R(y), \neg P(y)\ , \neg Q(x, x), Q(x, y), Q(y, x), Q(y, y)\}$ is an example of a 2-type of ϕ. Another example of a 2-type of ϕ is $\{\neg R(x), \neg P(x), \neg R(y), P(y), Q(x, x), Q(x, y), \neg Q(y, x),\ \neg Q(y, y)\}$. The set of all 2-types over the signature $\Sigma(\phi)$ is denoted as $\pi(\phi)$.

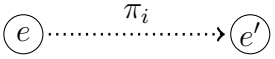
Given a model $\mathfrak{A}$ and a pair of elements of the model (e_1, e_2), the 2-type of the elements (e_1, e_2) consists of the set of all the unary literals R_u^x from the signature of $\mathfrak{A}$ in variable x such that $\mathfrak{A} \models R_u^x(e_1)$ and all the unary literals R_u^y from the signature of $\mathfrak{A}$ in variable y such that $\mathfrak{A} \models R_u^y(e_2)$, as well all the binary literals R_b from the signature of $\mathfrak{A}$ such that $\mathfrak{A} \models R_b(e_1, e_2)$. The 2-type of (e_1, e_2) in $\mathfrak{A}$

is denoted $\mathrm{tp}^{\mathfrak{A}}[e_1, e_2]$. A 2-type π_i is said to be *realised* in $\mathfrak{A}$ if there exists a pair of distinct elements $e_1 \in \mathfrak{A}, e_2 \in \mathfrak{A}$ such that $\mathrm{tp}^{\mathfrak{A}}[e, e'] = \pi_i$.

The 1-types and 2-types can be represented as formulas in the language of $\mathcal{C}^2()$ as the conjunctions of all their literals.

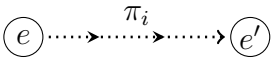
Recalling that for a given sentence ϕ in normal form M signifies the number of conjuncts with the $\forall x \exists y_{\lhd c_i}$ quantifier prefix and β_j designates one of the quantifier-free subformulas of ϕ that are preceeded by this quantifier prefix, the following applies.

For any 2-type $\pi_i \in \pi(\phi)$, if $\models \pi_i(x, y) \rightarrow \neg(\beta_j(x, y) \vee \beta_j(y, x))$ for all $1 \leq j \leq M$, then π_i is called a *silent 2-type*. The silent 2-type π_i between the elements e and e' is depicted as follows.

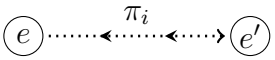


For any pair of elements $e \in \mathfrak{A}, e' \in \mathfrak{A}$, the element e *sends a message* to the element e' when $\mathfrak{A} \models \beta_i(e, e')$ for at least one β_i.

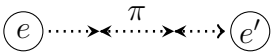
For any 2-type $\pi_i \in \pi(\phi)$, if $\models \pi_i(x, y) \rightarrow \beta_j(x, y)$ for some $1 \leq j \leq M$, then π_i is called *forward message-type*. The forward message-type π_i between the elements e and e' is depicted as follows.



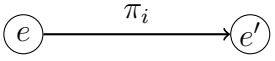
For any 2-type $\pi_i \in \pi(\phi)$, if $\models \pi_i(x, y) \rightarrow \beta_j(y, x)$ for some $1 \leq j \leq M$, then π_i is called *reverse message-type*. The reverse message-type π_i between the elements e and e' is depicted as follows.



An *invertible message-type* is both a forward message-type and a reverse message-type. The invertible message-type π between the elements e and e' is depicted as follows.



For any 2-type $\pi_i \in \pi(\phi)$, if $\models \pi_i(x, y) \rightarrow \downarrow^+ (x, y)$, then π_i is called *edge-type*. The edge-type π_i between the elements e and e' is depicted as follows.



The set of all the silent 2-types over the signature $\Sigma(\phi)$ is denoted $s(\phi)$. The set of all message-types (forward, reverse and invertible) over the signature $\Sigma(\phi)$ is denoted $\mu(\phi)$. The set of all forward message-types over the signature $\Sigma(\phi)$ is denoted $\mu^{\rightarrow}(\phi)$. The set of all reverse message-types over the signature $\Sigma(\phi)$ is

denoted $\mu^{\leftarrow}(\phi)$. The set of all invertible message-types over the signature $\Sigma(\phi)$ is denoted $\mu^{\leftrightarrow}(\phi)$. The set of all edge-types over the signature $\Sigma(\phi)$ is denoted $\varepsilon(\phi)$.

Denote by N the sum of the number of forward message-types and invertible message-types in $\Sigma(\phi)$. The *star-type* of an element $e \in \mathfrak{A}$ is the 1-type of e and an N-tuple $\sigma = (v_1, ..., v_N)$ of natural numbers, where for every forward or invertible message-type $\mu_j \in \mu^{\rightarrow}(\phi) \cup \mu^{\leftrightarrow}(\phi)$, the corresponding number v_j is calculated as follows:

$$v_j = |\{e' \in \mathfrak{A} \setminus \{e\} : \mathrm{tp}^{\mathfrak{A}}[e, e'] = \mu_j\}|$$

The definition of a star-type is adopted from [Benedikt et al., 2012]. In essence, a star-type of an element records the 1-type of that element and the number of times the element sends each of the forward and invertible message-types to the other elements of the model. A star-type of an element $e \in \mathfrak{A}$ is denoted $\mathrm{st}^{\mathfrak{A}}[e]$. The set of all star-types in the signature $\Sigma(\phi)$ is denoted $st(\phi)$. A star-type st_j consisting of the 1-type τ_j and the N-tuple $(v_1, ..., v_N)$ can be expressed as a formula of $\mathcal{C}^2()$ over one free variable x as follows

$$st_j(x) = \tau_j(x) \wedge \bigwedge \{\exists_{=v_i} y \mu_i(x, y) \mid \mu_i \in \mu^{\rightarrow}(\phi) \cup \mu^{\leftrightarrow}(\phi), 1 \leq i \leq N\}$$

Given a sentence ϕ and a model $\mathfrak{A}$, writing $\mathfrak{A}|\Sigma(\phi)$ (or, alternatively $\mathfrak{A}|\phi$) designates the restriction of the model $\mathfrak{A}$ to the signature $\Sigma(\phi)$.

For a pair of models $\mathfrak{A}$ and $\mathfrak{A}'$, we write $\mathfrak{A} \approx \mathfrak{A}'$, when $|\mathfrak{A}| = |\mathfrak{A}'|$ (the universes of the models coincide) and for every $e \in \mathfrak{A}$, $\mathrm{st}^{\mathfrak{A}}[e] = \mathrm{st}^{\mathfrak{A}'}[e]$.

A *tree-like cycle* is a directed graph that consists of a single cycle subgraph and a number of tree subgraphs rooted at the nodes of the cycle. Figure 3.1 demonstrates an example of a tree-like cycle. The elements a, b, f form the cycle of that tree-like cycle graph. The nodes b and f are the roots of trees of that tree-like cycle graph.

The model $\mathfrak{A}$ is called *arboreal* when the graph of $\mathfrak{A}$ consists of a positive number of connected components that are trees and a number of connected components that are tree-like cycles.

The forest of $\mathfrak{A}$ is the set of the the connected components of the graph of $\mathfrak{A}$ that are trees. An edge between two elements of $\mathfrak{A}$ that belong to the forest of $\mathfrak{A}$ is called a *forest edge*. Since every tree-like cycle connected component contains

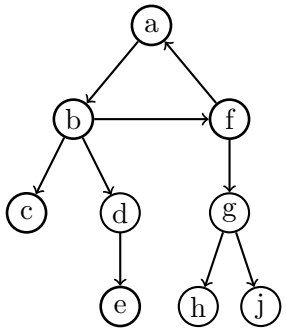


Figure 3.1: An example of a tree-like cycle

a cycle as its subgraph, the number of tree-like cycle connected components and the number of cycle subgraphs is always the same in the graph of an arboreal model $\mathfrak{A}$. An edge between two elements of $\mathfrak{A}$ that belong to a cycle of $\mathfrak{A}$ is called a *cycle edge*. Every element of an arboreal model $\mathfrak{A}$ either belongs to the forest of $\mathfrak{A}$ or to the tree-like cycles of $\mathfrak{A}$.

The model $\mathfrak{A}$ is called *alternating* when for every path in the graph of length 3 the nodes that belong to the path are assigned a distinct 1-type. The model $\mathfrak{A}$ is called *ϕ-differentiated* when every 1-type $\tau_i \in \tau(\phi)$ realised in $\mathfrak{A}$ is either a king 1-type or a populous 1-type. The model $\mathfrak{A}$ is called *ϕ-populated* when every populous 1-type $\tau_i \in \tau(\phi)$ is realised more than $5 \cdot Z$ in the forest of $\mathfrak{A}$. The model $\mathfrak{A}$ is called *ϕ-rewirable* when for every star-type $s_i \in s(\phi)$, if s_i is realised in $\mathfrak{A}$, then s_i is realised in the forest of $\mathfrak{A}$.

The model $\mathfrak{A}$ is called *ϕ-chromatic* when for every element $e \in \mathfrak{A}$ the following two conditions hold: 1) there does not exist an element $e' \in \mathfrak{A}$ such that $\mathrm{tp}^{\mathfrak{A}}[e] = \mathrm{tp}^{\mathfrak{A}}[e']$ and $\mathrm{tp}^{\mathfrak{A}}[e, e'] \in \mu^{\leftrightarrow}(\phi)$; 2) there does not exist a pair of elements $e' \in \mathfrak{A}$ and $e'' \in \mathfrak{A}$ such that $\mathrm{tp}^{\mathfrak{A}}[e'] = \mathrm{tp}^{\mathfrak{A}}[e'']$, $\mathrm{tp}^{\mathfrak{A}}[e, e'] \in \mu^{\leftrightarrow}(\phi)$ and $\mathrm{tp}^{\mathfrak{A}}[e, e''] \in \mu^{\leftrightarrow}(\phi)$.

For a given set of unary predicates $\mathtt{P}$, *a label* is the conjunction of a maximally consistent set of literals over the variable x and the set of predicates $\mathtt{P}$. In essence, a label in the signature $\mathtt{P}$ is nothing else but a 1-type over $\mathtt{P}$.

For a given natural number n we can asociate a sentence ω that defines the set of labels L of size n. This is achieved by introducing the set of fresh unary predicates $\mathtt{P}$ of size $\lfloor log_2^n \rfloor + 1$ and generating the sentence ω that prevents the realisation of $2^{|\mathtt{P}|} - n$ labels in that signature. Since the total number of labels that are possible in the signature $\mathtt{P}$ is $2^{|\mathtt{P}|}$, the sentence ω allows for at most n labels to be realised in every model $\mathfrak{A} \models \omega$.

Every element of every model $\mathfrak{A} \models \omega$ is always assigned exactly one label

from L. This is because the unary predicates that are used to generate labels are part of the signature of ω. Therefore the 1-type of every element of $\mathfrak{A}$ is going to contain some combination of these unary predicates, and satisfy a particular label, as a result. Essentially labels are used for colouring the elements of the model with unique and mutually exclusive combinations of unary predicates.

Chapter 4

Finite satisfiability of $\mathcal{C}^2(\downarrow^+)$

4.1 Preliminaries

This chapter presents the proof that the finite satisfiability of the sentences of $\mathcal{C}^2(\downarrow^+)$ can be established in nondeterministic exponential time. In broad terms this chapter demonstrates a reduction of the problem of establishing the finite satisfiability of a given sentence ϕ in $\mathcal{C}^2(\downarrow^+)$ to the problem of establishing the finite satisfiability of the sentence ω_ϕ in $\mathcal{C}^2()$, which is derived from ϕ in a predetermined way.

In order to describe how the sentence ω_ϕ is derived from ϕ we need to introduce the notion of a translator. A *translator* is a nondeterministic exponential time algorithm that accepts as input a sentence of $\mathcal{C}^2()$ in normal form and for each possible run returns a sentence of $\mathcal{C}^2()$ in normal form such that the rough size of the output sentence is polynomial in the rough size of the input sentence. For a given translator Ω, for a given sentence ϕ, a nondeterministic run of Ω on ϕ that produces the output sentence ω is denoted $\omega = \Omega(\phi)$. The definition $\Omega(\phi) := \omega_1 \wedge \omega_2 \wedge ... \wedge \omega_i$ expresses the fact that the translator Ω takes ϕ as input and returns the sentence $\omega_1 \wedge \omega_2 \wedge ... \wedge \omega_i$ as output, for some set of sentences $\omega_1, \omega_2, ..., \omega_i$. Defining functions as translators helps keep a bound on the rough size of the output sentences that these functions generate. Informally speaking, the sentence generated by a translator can be used safely in the proof without running the risk of increasing the complexity class of the finite satisfiability algorithm of $\mathcal{C}^2(\downarrow^+)$.

The sentence ω_ϕ is generated from ϕ nondeterministically by the translator Ω (defined in Section 4.2.8), which takes the sentence ϕ as input. It is demonstrated in this chapter that a given sentence ϕ is finitely satisfiable in $\mathcal{C}^2(\downarrow^+)$ if and only

if the algorithm Ω, given ϕ as input, nondeterministically produces a sentence ω_ϕ as output, such that ω_ϕ is finitely satisfiable in $\mathcal{C}^2()$. This is stated formally as the following theorem.

Theorem 1. *There exists a sentence $\omega_\phi = \Omega(\phi)$ that is finitely satisfiable in $\mathcal{C}^2()$ if and only if ϕ is finitely satisfiable in $\mathcal{C}^2(\downarrow^+)$.*

Because the algorithm Ω is a translator, the rough size of the generated sentence ω_ϕ is always polynomial with respect to the size of ϕ. Due to Lemma 1, given that the finite satisfiability of $\mathcal{C}^2()$ is in NEXPTIME, the same upper bound for the finite satisfiability of $\mathcal{C}^2(\downarrow^+)$ follows.

Forward implication of Theorem 1

The fact that the left part of the biconditional in Theorem 1 is sufficient for the right part follows from the way Ω is defined: the generated sentence ω_ϕ consists of a set of conjuncts, the sentence ϕ is one of the conjuncts of ω_ϕ and the rest of the conjuncts of ϕ can always be satisfied by extending any tree model of ϕ in a straightforward manner. This is described in more detail in Section 4.2.8 as the proof of Lemma 7.

Reverse implication of Theorem 1

The fact that the left part of the biconditional in Theorem 1 is necessary for the right part follows from demonstrating how an arbitrary model of ω_ϕ can always be converted into a model of ϕ that is also a forest. The technique of transforming an arbitrary model of ω_ϕ into a forest model of ϕ is adapted from [Charatonik and Witkowski, 2013].

The process of the transformation of a model of ω_ϕ begins with recognising that, due to the way ω_ϕ is generated by Ω, such a model must possess a number of specific properties. These properties and the way they are imposed on the models of sentences generated by Ω are described in the subsections of Section 4.2. Each of these subsections defines how a part of the sentence ω_ϕ is created that is responsible for one of the model properties, and in the end of this section all these parts are put together and the definition of the translator Ω is provided.

One of these model properties is the arboreal property, which maintains that the graph of the model possessing such a property consists of a set of trees and a set of tree-like cycles. The model can be modified then by swapping the 2-types

between certain pairs of elements, while preserving the finite satisfiability of the resulting model with respect to the sentence ω_ϕ. As the result of this model surgery, a tree-like cycle gets embedded in an existing tree, becoming one of its branches. Repeating this process for each tree-like cycle converts the given model into a model that has a graph consisting only of trees and no tree-like cycles.

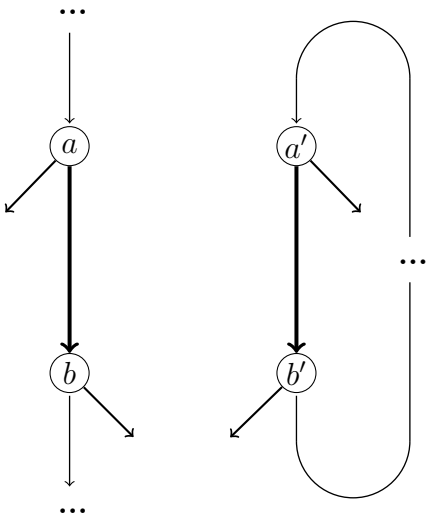


Figure 4.1: Before changing 2-types

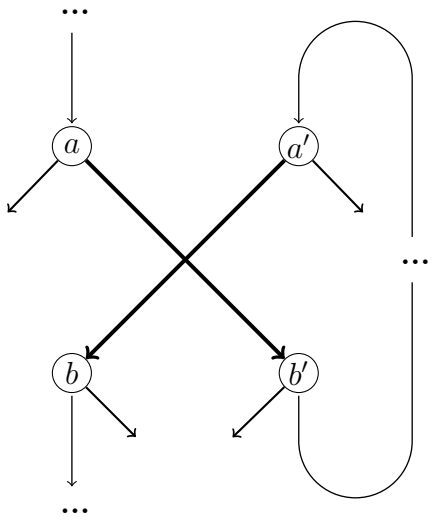


Figure 4.2: After changing 2-types

Section 4.3 is dedicated to the proof of a fairly elaborate lemma that is necessary for handling a certain type of rewiring cases in Section 4.4. The lemma in question guarantees that for any given element of a model it is possible to find another element of the right kind, such that the two elements are assigned a silent 2-type. Later on this lemma is referred to in the part of the proof that is handling different model surgery cases, where the fact about the existence of the required element is invoked.

Section 4.4 is where, using the preparations from the previous sections, it is shown that if there exists a model of a sentence generated by Ω, there exists a forest model of that sentence, as well.

Firstly, it is demonstrated in the section that the process of rewiring a model with a given set of properties preserves these properties after a tree-like cycle gets embedded into the tree. This is necessary so that any number of rewirings can take place using the same procedure, until all tree-like cycles that exist in the model are embedded into the tree components of the model.

Secondly, a suitable edge of the forest component of the model and a suitable edge of the tree-like cycle of the model are located for carrying out the rewiring,

as demonstrated on page 78 in Figure 4.13 and Figure 4.14 of Section 4.4. The existence of such edges is due to the rewirable property of the model under consideration, imposed on that model by the sentence ω_{rew} that is described in Section 4.2.7.

Thirdly, each individual case of possible 2-types that can appear between the selected edges of the graph is considered. It is shown that in any possible scenario the rewiring process leads to the model that preserves the relevant properties that it had before the rewiring, including the finite satisfiability with respect to the sentence ϕ.

Section 4.5 summarises the result of the model rewiring process as the proof of Lemma 12. This leads to the statement of Theorem 2, that is the main result of this chapter.

4.2 Auxiliary sentences and translators

Every model that satisfies the sentences generated by the translator Ω has to have the arboreal, alternating, chromatic, differentiated, populated and rewirable properties (with some qualification for the signature that these properties apply to, as detail in Subsection 4.2.8). The definition of the translator Ω depends on a set of auxiliary sentences and translators, each of which generates a sentence that forces one the above mentioned properties on its models.

4.2.1 The sentence ω_{arb} and the arboreal property

The sentence ω_{arb} makes sure that every model of that sentence has the arboreal property.

$$\omega_{arb} = \omega^1_{arb} \wedge \omega^2_{arb} \wedge \omega^3_{arb}$$

The sentence ω^1_{arb} prevents the existence of cycles consisting of one and two nodes in the graphs of $\downarrow^+$ for technical reasons.

$$\omega^1_{arb} = \forall x \neg \downarrow^+(x,x) \wedge \forall x \forall y (\downarrow^+(x,y) \rightarrow \neg \downarrow^+(y,x))$$

The sentence ω^2_{arb} makes sure that there always exists at least one root node in the graph of $\downarrow^+$.

$$\omega^2_{arb} = \exists x \forall y \neg \downarrow^+(y,x)$$

The sentence ω^3_{arb} prevents the existence of more than one parent for every node in the graph of $\downarrow^+$.

$$\omega^3_{arb} = \forall x \forall y (\downarrow^+(x,y) \leftrightarrow \uparrow^+(y,x)) \wedge \forall x \exists_{\leq 1} y \uparrow^+(x,y)$$

The rough size of the sentence ω_{arb} can be considered polynomial with respect to the rough size of ϕ as the length of ω_{arb} is constant.

4.2.2 The sentence ω_{alt} and the alternating property

The sentence ω_{alt} makes sure that every model satisfying it has the alternating property.

The sentence ω_{alt} assigns one of the three colours to each element of the model of ω_{alt} so that the colour of each element is different from the colour of its parent and its children and the colours of its children and its parent are different from each other.

$$\omega_{alt} = \omega^1_{alt} \wedge \omega^2_{alt} \wedge \omega^3_{alt} \wedge \omega^4_{alt} \wedge \omega^5_{alt} \wedge \omega^6_{alt} \wedge \omega^7_{alt}$$

$$\omega^1_{alt} = \forall x \left(red\,(x) \vee green\,(x) \vee blue\,(x) \right)$$

$$\omega^2_{alt} = \forall x \left(red\,(x) \rightarrow (\neg green\,(x) \wedge \neg blue\,(x)) \right)$$

$$\omega^3_{alt} = \forall x \left(green\,(x) \rightarrow (\neg red\,(x) \wedge \neg blue\,(x)) \right)$$

$$\omega^4_{alt} = \forall x \left(blue\,(x) \rightarrow (\neg red\,(x) \wedge \neg green\,(x)) \right)$$

$$\omega^5_{alt} = \forall x \forall y \left(\left(red\,(x) \wedge \downarrow^+(x,y) \right) \rightarrow green\,(y) \right)$$

$$\omega^6_{alt} = \forall x \forall y \left(\left(green\,(x) \wedge \downarrow^+(x,y) \right) \rightarrow blue\,(y) \right)$$

$$\omega^7_{alt} = \forall x \forall y \left(\left(blue\,(x) \wedge \downarrow^+(x,y) \right) \rightarrow red\,(y) \right)$$

The rough size of the sentence ω_{alt} can be considered polynomial with respect to the rough size of ϕ as the length of ω_{alt} is constant.

4.2.3 The translator Ω_{chr} and the chromatic property

The translator Ω_{chr} accepts as input a sentence $\phi \in \mathcal{C}^2()$ and makes sure that every model satisfying the sentence $\omega_{chr} = \Omega_{par}(\phi)$ has the ω_{chr}-chromatic property.

$$\Omega_{chr}(\phi) := \phi \wedge \omega^1_{chr} \wedge \omega^2_{chr} \wedge \omega^3_{chr}$$

Firstly, the sentences ω_1 and ω_2 ensure that the sentences ω_{chr} can only be satisfied by the models that are ω_{chr}-chromatic. This is achieved by ω_1 and ω_2 encoding the two requirements of the chromaticity.

The sentence ω^1_{chr} makes sure that for any pair of elements of the same 1-type, the 2-type of that pair of elements is not an invertible message-type.

$$\omega^1_{chr} = \bigwedge\{\forall x \forall y((\tau_i(x) \wedge \tau_i(y)) \rightarrow \neg\, \pi_j(x,y)) \mid \tau_i \in \tau(\omega_{chr}), \pi_j \in \mu^{\leftrightarrow}(\omega_{chr})\}$$

The sentence ω^2_{chr} makes sure that every element can send at most one invertible message type to the elements of the same 1-type.

$$\omega^2_{chr} = \bigwedge\{\forall x \exists_{\leq 1} y(\tau_i(y) \wedge \bigvee\{\pi_j(x,y) \mid \pi_j \in \mu^{\leftrightarrow}(\omega_{chr})\}) \mid \tau_i \in \tau(\omega_{chr})\}$$

Secondly, the sentence ω_3 ensures that for any model satisfying ϕ there exists a model in the signature $\Sigma(\phi) \cup \Sigma(\omega_{chr})$ that meets the constraints of ω^1_{chr} and ω^2_{chr}. This is achieved by ω_3 defining the set of $Z^2 + 1$ labels L_{chr}.

The predicates used for encoding L_{chr} are the only new predicates added by the translator Ω_{chr} to the signature of the sentence ϕ. This allows us to define the signature of the sentence ω_{chr} as $\Sigma(\phi) \cup \Sigma(\omega^3_{chr})$.

The rough size of the sentences generated by Ω_{chr} can be considered polynomial with respect to the rough size of ϕ, because their length is at most exponential with respect to the rough size of ϕ and the size of their signature is at most polynomial with respect to the rough size of ϕ. The set L_{chr} contains an exponential number of labels, but it is necessary to introduce only a logarithmic number of new unary predicates to encode the labels.

4.2.4 The translator Ω_{diff} and the differentiated property

The translator Ω_{diff} accepts as input a sentence $\phi \in \mathcal{C}^2()$ and makes sure that every model satisfying any sentence $\omega_{diff} = \Omega_{diff}(\phi)$ has the ω_{diff}-differentiated property.

$$\Omega_{diff}(\phi) := \phi \wedge \omega^1_{diff} \wedge \omega^2_{diff} \wedge \omega^3_{diff}$$

Firstly, the number N is nondeterministically chosen between 0 and $|\mathcal{P}(\tau(\phi))|$. Then the sentence ω^1_{diff} defines the set of $5 \cdot Z \cdot N$ labels L_{diff}.

Two sets K and P are nondeterministically chosen as the elements of the set $\mathcal{P}(\tau(\Sigma(\phi) \cup \Sigma(\omega_{diff})))$ such that $K \cup P = \tau(\phi)$ and $K \cap P = \emptyset$. They are called

respectively the set of king 1-types and the set of populous 1-types.

The sentence ω^2_{diff} makes sure that the elements of king 1-types are realised at most once.

$$\omega^2_{diff} = \bigwedge\{\forall x \forall y((\tau_i(x) \wedge \tau_i(y)) \rightarrow x = y) \mid \tau_i \in K\}$$

The sentence ω^3_{diff} makes sure that the elements of populous 1-types are realised at least $5 \cdot Z$ times.

$$\omega^3_{diff} = \bigwedge\{\exists_{>5\cdot Z} x \tau_i(x) | \tau_i \in P\}$$

If there exists a model $\mathfrak{A} \models \phi$, then the number N can be nondeterministically selected to be equal to the number of noble 1-types realised in $\mathfrak{A}$. The model $\mathfrak{A}'|_{\Sigma(\phi)} = \mathfrak{A}$ can be produced from $\mathfrak{A}$ by assigning a unique label from L_{diff} to every element of a noble 1-type in $\mathfrak{A}$. As the result, $\mathfrak{A}' \models \phi \wedge \omega_{diff}$ and every 1-type of $\mathfrak{A}'$ is either realised fewer than two times or more than $5 \cdot Z$ times.

At the same time, for any sentence $\omega_{diff} = \Omega_{diff}(\phi)$, for any model $\mathfrak{A} \models \omega_{diff} \wedge \phi$, the elements of $\mathfrak{A}$ are either realised one time or more than $5 \cdot Z$ times.

The rough size of the sentences generated by Ω_{diff} can be considered polynomial with respect to the rough size of ϕ, because their length is at most exponential with respect to the rough size of ϕ and the size of their signature is at most polynomial with respect to the rough size of ϕ. The values that appear in the counting quantifiers of the these sentences is also at most exponential with respect to ϕ, and the binary encoding of their sum is, thus, polynomial with respect to ϕ. The set L_{diff} contains an exponential number of labels, but it is necessary to introduce only a logarithmic number of new unary predicates to encode the labels.

4.2.5 The translator Ω_{frst} and labelling forest elements

The translator Ω_{frst} accepts as input a sentence ϕ that is a disjunction of a set of labels L_I and makes sure that every model satisfying any sentence $\omega_{frst} = \Omega_{frst}(L_I)$ does not have any labels from L_I realised outside its forest.

$$\Omega_{frst}(\phi) := \omega^1_{frst} \wedge \omega^2_{frst} \wedge \omega^3_{frst} \wedge \omega^4_{frst} \wedge \omega^5_{frst} \wedge \omega^6_{frst} \wedge \omega^7_{frst} \wedge \omega^8_{frst} \wedge \omega^9_{frst} \wedge \omega^{10}_{frst}$$

The sentence ω^1_{frst} defines the set of $2 \cdot |L_I|$ labels L_C. The sentence ω^2_{frst}

defines the set of $3 \cdot |L_I|$ labels L_O.

The binary relation $\Downarrow$ is nondeterministically generated on the set $L_I \cup L_C$ such that the graph of $\Downarrow$ is a forest. The labels from the set $L_I \cup L_C$ are put into an arbitrary (but deterministically generated) one-to-one correspondence with the labels from the set L_O. This is achieved by defining the bijective relation $\Leftrightarrow$ on the set $(L_I \cup L_C) \times L_O$.

The sentences ω^3_{frst}, ω^4_{frst}, ω^5_{frst} make sure that each of the L labels can only be realised in the forest of the model of the output sentence.

$$\omega^3_{frst} = \bigwedge\{\forall x \forall y((\downarrow^+(x,y) \wedge l_i(y)) \rightarrow (o_m(x) \vee c_k(x) \vee l_j(x)) \mid \\ l_i \in L_I, l_j \in L_I, c_k \in L_C, o_m \in L_O, \Downarrow(l_i,l_j), \Downarrow(l_i,c_k), \Leftrightarrow(l_i,o_m)\}$$

$$\omega^4_{frst} = \bigwedge\{\forall x \forall y((\downarrow^+(x,y) \wedge c_i(y)) \rightarrow (o_m(x) \vee c_k(x) \vee l_j(x)) \mid \\ c_i \in L_C, o_m \in L_O, \Leftrightarrow(c_i,o_m), c_k \in L_C, \Downarrow(c_i,c_k), l_j \in L_I, \Downarrow(c_i,l_j)\}$$

$$\omega^5_{frst} = \bigwedge\{\forall x \forall y((\downarrow^+(x,y) \wedge o_i(y)) \rightarrow (o_i(x) \vee c_j(x)) \mid \\ o_i \in L_O, c_j \in L_C, \Leftrightarrow(o_i,c_j)\}$$

The sentences ω^6_{frst}, ω^7_{frst} and ω^8_{frst} assign an additional unary predicate to each of the labelled elements of the model.

$$\omega^6_{frst} = \bigwedge\{\forall x(c_i(x) \rightarrow c(x)) \mid c_i \in \overline{L_C}\}$$

$$\omega^7_{frst} = \bigwedge\{\forall x(l_i(x) \rightarrow l(x)) \mid l_i \in \overline{L_I}\}$$

$$\omega^8_{frst} = \bigwedge\{\forall x(o_i(x) \rightarrow o(x)) \mid o_i \in \overline{L_O}\}$$

The sentence ω_9 prevents the paths of O-labelled elements from branching.

$$\omega^9_{frst} = \forall x \exists_{\leq 1} y(\downarrow^+(x,y) \wedge o(y))$$

The sentence ω^{10}_{frst} prevents more than one L, C or O label from being assigned to the same element.

$$\omega^{10}_{frst} = \forall x((o(x) \rightarrow \neg(l(x) \vee c(x))) \wedge (l(x) \rightarrow \neg(c(x) \vee o(x))) \wedge (c(x) \rightarrow \neg(l(x) \vee o(x))))$$

The rough size of the sentences generated by Ω_{frst} can be considered polynomial with respect to the rough size of ϕ, because their length is at most exponential with respect to the rough size of ϕ and the size of their signature is at most polynomial with respect to the rough size of ϕ. The sets L_C and L_O each contain an exponential number of labels, but it is necessary to introduce only a logarithmic number of new unary predicates to encode the labels.

4.2.6 The translator Ω_{pop} and the populated property

The translator Ω_{pop} accepts as input a sentence $\phi \in \mathcal{C}^2()$ and makes sure that every model satisfying any sentence $\omega_{pop} = \Omega_{pop}(\phi)$ has the ϕ-populated property.

$$\Omega_{pop}(\phi) := \phi \wedge \omega^1_{pop} \wedge \omega^2_{pop} \wedge \omega^3_{pop} \wedge \omega^4_{pop} \wedge \omega^5_{pop}$$

The set T is nondeterministically chosen as an element of the set $\mathcal{P}(\tau(\phi))$. It represents the set of populous 1-types realised in some model of ϕ.

The sentence ω^1_{pop} defines the set of $5 \cdot Z \cdot |T|$ labels L_{pop}.

The relation $\Leftrightarrow$ is defined on the set $T \times L_{pop}$ in such a way that for each $\tau_i \in T$ there exists exactly $5 \cdot Z$ labels $l_j \in L_{pop}$ such that $\Leftrightarrow(\tau_i, l_j)$.

The sentences ω^2_{pop}, ω^3_{pop} and ω^4_{pop} make sure that the elements of the populous 1-types are realised more than $5 \cdot Z$ times among the forest elements.

The sentence ω^2_{pop} makes sure that the labels L_{pop} can only be realised in the forest.

$$\omega^2_{pop} = \Omega_{frst}(\bigvee L_{pop})$$

The sentence ω^3_{pop} makes sure that each of the L_{pop} labels is realised exactly one time.

$$\omega^3_{pop} = \bigwedge \{\forall x \exists_{=1} y\, l_i(y) \mid l_i \in L_{pop}\}$$

The sentence ω^4_{pop} makes sure that the elements of the populous 1-types are realised at least $5 \cdot Z$ times.

$$\omega^4_{pop} = \bigwedge \{\forall x(l_i(x) \rightarrow \tau_j(x)) | l_i \in L_{pop}, \tau_j \in T, l_i \Leftrightarrow \tau_j\}$$

The sentence ω^5_{pop} prevents any 1-type that is not in T from being populous.

$$\omega^5_{pop} = \bigwedge \exists_{\leq 5 \cdot Z} x \tau_i(x) | \tau_i \in \tau(\phi), \tau_i \notin T)$$

The rough size of the sentences generated by Ω_{pop} can be considered polynomial with respect to the rough size of ϕ, because their length is at most exponential with respect to the rough size of ϕ and the size of their signature is at most polynomial with respect to the rough size of ϕ. The values that appear in the counting quantifiers of the these sentences is also at most exponential with respect to the rough size of ϕ, and the binary encoding of their sum is, thus, polynomial with respect to the rough size of ϕ. The set L_{pop} contains an exponential number of labels, but it is necessary to introduce only a logarithmic number of new unary predicates to encode the labels. The rough size of the sentence ω^2_{pop} is polynomial with respect to the rough size of ϕ, because Ω_{frst} is a translator and the rough size of $\bigvee L_{pop}$ is polynomial with respect to the rough size of ϕ.

4.2.7 The translator Ω_{rew} and the rewirable property

The translator Ω_{rew} accepts as input a sentence $\phi \in \mathcal{C}^2()$ and makes sure that every model satisfying any sentence $\omega_{rew} = \Omega_{rew}(\phi)$ has the ϕ-rewirable property.

$$\Omega_{rew}(\phi) := \phi \wedge \omega^1_{rew} \wedge \omega^2_{rew} \wedge \omega^3_{rew} \wedge \omega^4_{rew}$$

A set of star-types S is nondeterministically chosen as an element of the set $\mathcal{P}(st(\phi))$ such that the cardinality of S is at most exponential with respect to ϕ. The sentence ω^1_{rew} defines the set of $|S|$ labels L_{rew}.

In order to put the elements of the set L_{rew} into a one-to-one correspondence with the elements of S the bijective relation $\Leftrightarrow$ is generated on the set $L_{rew} \times S$ in an arbitrary way.

The sentence ω^2_{rew} makes sure that each of the $L_{rew} \setminus \{l_0\}$ labels is realised only in the forest of $\mathfrak{A}$. The empty label has to be allowed to realise anywhere in the model, otherwise some models of ϕ that are forests may not satisfy some of the sentences $\omega_{rew} = \Omega_{rew}(\phi)$.

$$\omega^2_{rew} = \Omega_{frst}(\bigvee L_{rew} \setminus \{l_0\})$$

The sentence ω^3_{rew} prevents any L_{rew} labels from being assigned to the root nodes of the forest.

$$\omega^3_{rew} = \forall x \forall y(\neg \downarrow^+(y, x) \rightarrow \bigwedge\{\neg l_i(x) \mid l_i \in L_{rew}\})$$

The sentence ω^4_{rew} makes sure that each element realising a label from L_{rew} is assigned a star-type that corresponds to the label according to the relation $\Leftrightarrow$.

$$\omega^4_{rew} = \bigwedge\{\forall x(l_i(x) \to st_j(x) \mid l_i \in L_{rew}, st_j \in S, l_i \Leftrightarrow st_j\}$$

The sentence ω^5_{rew} makes sure that each of the L_{rew} labels is realised exactly once.

$$\omega^5_{rew} = \bigwedge\{\forall x \exists_{=1} y\, l_i(y) \mid l_i \in L_{rew}\}$$

The sentence ω^6_{rew} makes sure that only the star-types from the set S are allowed to be realised in $\mathfrak{A}$. Otherwise, some star-types that do not belong to S may be realised outside the forest of the model of ω_{rew}.

$$\omega^6_{rew} = \bigwedge\{\forall x \neg st_i(x) \mid st_i \in st(\phi) \setminus S\}$$

The rough size of the sentences generated by Ω_{rew} can be considered polynomial with respect to the rough size of ϕ, because their length is at most exponential with respect to the rough size of ϕ and the size of their signature is at most polynomial with respect to the rough size of ϕ. The set L_{rew} contains an exponential number of labels, but it is necessary to introduce only a logarithmic number of new unary predicates to encode the labels. The rough size of the sentence ω^2_{rew} is polynomial with respect to the rough size of ϕ, because Ω_{frst} is a translator and the rough size of $\bigvee L_{rew} \setminus \{l_0\}$ is polynomial with respect to the rough size of ϕ.

4.2.8 The translator Ω

Consider the following definition of the translator Ω from the other sentences and translators described in this section.

$\Omega_1(\phi) := \phi \wedge \omega_{arb} \wedge \omega_{alt}$

$\Omega_2(\phi) := \omega_{chrom}$, where $\omega_{chrom} = \Omega_{chrom}(\omega_2)$ and $\omega_2 = \Omega_1(\phi)$.

$\Omega_3(\phi) := \omega_{diff}$, where $\omega_{diff} = \Omega_{diff}(\omega_3)$ and $\omega_3 = \Omega_2(\phi)$.

$\Omega_4(\phi) := \omega_{pop}$, where $\omega_{pop} = \Omega_{pop}(\omega_4)$ and $\omega_4 = \Omega_3(\phi)$.

$\Omega(\phi) := \omega_{rew}$, where $\omega_{rew} = \Omega_{rew}(\omega_5)$ and $\omega_5 = \Omega_4(\phi)$.

The translator Ω forces certain properties on the models of the sentences that it generates. The following lemmas describe the signature that each of these properties applies to.

Lemma 2. *For any sentence $\phi \in \mathcal{C}^2()$, for any $\omega = \Omega_1(\phi)$, for any model $\mathfrak{A} \models \omega$, the model $\mathfrak{A}$ is arboreal and alternating.*

Proof. Self-evident from the definitions of arboreal and alternating model properties and the definitions of ω_{arb} and ω_{alt} sentences. □

Lemma 3. *For any sentence $\phi \in \mathcal{C}^2()$, for any $\omega = \Omega_2(\phi)$, for any model $\mathfrak{A} \models \omega$, the model $\mathfrak{A}$ is arboreal, alternating and ω-chromatic.*

Proof. The output sentence of the translator Ω_{chrom} guarantees the chromatic property for any model satisfying it. The arboreal and alternating properties of the model are preserved, as they are signature independent. □

Lemma 4. *For any sentence $\phi \in \mathcal{C}^2()$, for any $\omega = \Omega_3(\phi)$, for any model $\mathfrak{A} \models \omega$, the model $\mathfrak{A}$ is arboreal, alternating, ω-chromatic and ω-differentiated.*

Proof. Since Ω_{diff} translator does not contain any $\forall x \exists_{\lhd c_i} y \beta_i(x, y)$ clauses, no new invertible message-types can arise in $\mathfrak{A}$ as compared to $\mathfrak{A}|\omega_2$, where ω_2 is the sentence that is generated by the Ω_2 translator as part of the application of Ω_3 to ϕ. Therefore, the chromatic property defined by the output of $\Omega_2(\phi)$ in this case extends to the signature of ω as well. □

Lemma 5. *For any sentence $\phi \in \mathcal{C}^2()$, for any $\omega = \Omega_4(\phi)$, for any model $\mathfrak{A} \models \omega$, the model $\mathfrak{A}$ is arboreal, alternating, ω_3-chromatic, ω_3-differentiated and ω_3-populated, where ω_3 is the sentence that is generated by the Ω_3 translator as part of the application of Ω_4 to ϕ.*

Proof. For any given sentence $\phi \in \mathcal{C}^2()$, the translator Ω_{pop} generates a sentence $\omega = \Omega_{pop}(\phi)$ that guarantees the populated property only up to the signature of ϕ, not ω as a whole. This is contrasted with the translator Ω_{diff}, that guarantees the differentiated property for the signature of its output sentence, not only its input sentence. Therefore, the combination of all the above mentioned properties is justified for the signature of the sentence that is the result of the execution of Ω_3 in this case. □

Lemma 6. *For any sentence $\phi \in \mathcal{C}^2()$, any $\omega = \Omega(\phi)$ and any model $\mathfrak{A} \models \omega$, model $\mathfrak{A}$ is arboreal, alternating, ω_3-chromatic, ω_3-differentiated, ω_3-populated and ω_3-rewirable, where ω_3 is the sentence that is generated by the Ω_3 translator as part of the application of Ω to ϕ.*

Proof. The execution of Ω_{pop} translator results in a sentence that has a signature containing new predicates as compared to the signature of the sentence that it accepts as input. Therefore, the result of the execution of Ω produces a sentence that guarantees the rewirable property up to the signature of ω_4, where ω_4 is the result of the execution of Ω_4, not ω_3, where ω_3 is the result of the execution of Ω_3 in this case. However, the rewirable property is preserved under taking substructures of models for a reduced signature. Therefore, the ω_4-differentiated property of models satisfying ω also guarantees the ω_3-differentiated property for such models. □

It is also necessary to note that for a given sentence ϕ for every forest model that satisfies ϕ the translator Ω can generate an sentence that can accommodate such a model. This is a necessary condition for both directions of the biconditional of the Theorem 1 to hold.

Lemma 7. *If ϕ is finitely satisfiable in $\mathcal{C}^2(\downarrow^+)$, then some sentence $\omega = \Omega(\phi)$ is finitely satisfiable in $\mathcal{C}^2()$.*

Each of the properties described in this chapter is compatible with models that are forests. That is, any model of ϕ that is a forest can always be extended by appropriate new predicates in order to be satisfied by at least one of the sentences ω that the translator Ω can possibly generate on input ϕ.

4.3 Existence of silent pairs

In the previous section it was shown how to define a translator that generates sentences that place specific restrictions on their models.

However, before the rewiring of the edges of the model can take place, there has be an additional requirement with regards to the edges that can participate in the rewiring. More specifically, there has to be a restriction on the star-types of the child elements of these edges. Namely, for each of the elements b and b', as depicted in Figure 4.1, there has to be an element of a correct 1-type that connects to b (respectively b') with a silent 2-type. This requirement is formally expressed in this section as Lemma 10, which depends on Lemma 9, which in turn depends on Lemma 8.

The formulation and proof of Lemma 8 starts with the following definition. For any sentence $\phi \in \mathcal{C}^2()$, any 1-type $\tau_i \in \tau(\phi)$, any model $\mathfrak{M} \models \phi$ and any

element $m \in \mathfrak{M}$ of 1-type τ_i, the expression $free^{\mathfrak{M}}_{\tau_i}(m)$ denotes the set of elements $m' \in \mathfrak{M}$ of 1-type τ_i for which there exists an element $m'' \in \mathfrak{A}$ such that:

1. $\mathrm{tp}^{\mathfrak{M}}[m''] = \mathrm{tp}^{\mathfrak{M}}[m]$
2. $m \neq m''$
3. m'' is in the forest of $\mathfrak{M}$
4. $\mathrm{tp}^{\mathfrak{A}}[m, m']$ is not a forward message-type
5. $\mathrm{tp}^{\mathfrak{A}}[m', m'']$ is a silent 2-type

Lemma 8. *For any sentence $\phi \in \mathcal{C}^2()$, any sentence $\omega_3 = \Omega_3(\phi)$, any ω_3-populated, ω_3-rewirable model $\mathfrak{A} \models \omega_3$, any non-royal element $e \in \mathfrak{A}$ and any non-royal 1-type $\tau_1 \in \tau(\omega_3)$, there exist at least $2 \cdot Z$ elements in $free^{\mathfrak{A}}_{\tau_i}(e)$.*

Proof. For any element e, let R be the set $\{e', |\mathrm{tp}^{\mathfrak{A}}[e'] = \tau_1, \mathrm{tp}^{\mathfrak{A}}[e, e'] \notin \mu^{\rightarrow}(\omega_3)\}$. The size of R is always at least $4 \cdot Z$. This is because there are at least $5 \cdot Z$ elements e' of 1-type τ_1, since $\mathfrak{A}$ is ω_3-differentiated, and because e sends at most Z messages to them.

Let T be the set $\{e''|\mathrm{tp}^{\mathfrak{A}}[e''] = \mathrm{tp}^{\mathfrak{A}}[e], e''$ belongs to the forest of $\mathfrak{A}\}$. The size of T is at least $5 \cdot Z$. This is because $\mathfrak{A}$ is ω_3-populated.

Let K be the set $\{(e', e'') \in R \times T|\mathrm{tp}^{\mathfrak{A}}[e', e''] \notin s(\omega_3)\}$. The size of K is at most $|R| \cdot Z + |T| \cdot Z$. This is because each element from T sends at most $M \cdot C$ messages to the elements of R, and vice versa.

Let L be the set $R \times T$. The size of L is $|R| \cdot |T|$.

Let S be the set $\{(e', e'') \in R \times T|\mathrm{tp}^{\mathfrak{A}}[e', e''] \in s(\omega_3)\}$. The minimum number of elements of the set S is $|L| - |K|$.

Let F be the set $\{e' \in R|\exists e'' \in T\ \mathrm{tp}^{\mathfrak{A}}[e', e''] \in s(\omega_3)\}$. Each of the elements of R can participate in at most $|T|$ of the pairs from S. Therefore, the minimum number of elements of the set F is at least $|S|/|T|$.

As per the definitions above, the value of the expression $|S|/|T|$ is greater than or equal to $(|R| \cdot |T| - |T| \cdot Z - |R| \cdot Z)/|T|$, which according to the lower bounds on $|R|$ and $|T|$ is always greater than $2 \cdot Z$. Therefore, the size of the set F and, correspondingly, $free^{\mathfrak{A}}_{\tau_1}(e)$ is at least $2 \cdot Z$. □

Lemma 8 and the following terminology are used to formulate and prove Lemma 9. For a given model $\mathfrak{M}$, an element $m \in \mathfrak{M}$ and a 1-type τ_i, the

expression $silent^{\mathfrak{M}}_{\tau_i}(m)$ denotes the set of elements $m' \in \mathfrak{A}$ such that $\mathrm{tp}^{\mathfrak{M}}[m, m'] \in s(\phi)$ and $\mathrm{tp}^{\mathfrak{M}}[m'] = \tau_i$.

Lemma 9. *For any sentence $\phi \in \mathcal{C}^2()$, any sentence $\omega_3 = \Omega_3(\phi)$, any ω_3-populated, ω_3-rewirable model $\mathfrak{A} \models \omega_3$, and any pair of edges (a, b) and (a', b') of $\mathfrak{A}$ that have the same 2-type there exists an ω_3-populated, ω_3-rewirable model $\mathfrak{A}' \models \omega_3$, such that:*

1. $\mathfrak{A} \approx \mathfrak{A}'$
2. *the number of cycles in $\mathfrak{A}'$ is not more than the number of cycles in $\mathfrak{A}$*
3. *if (a, b) is an edge in a tree component of $\mathfrak{A}$, then (a, b) is an edge in a tree component of $\mathfrak{A}'$;*

   if (a', b') is an edge in a tree component of $\mathfrak{A}$, then (a', b') is an edge in a tree component of $\mathfrak{A}'$;
4. *if (a, b) is an edge in a tree-like cycle component of $\mathfrak{A}$, then (a, b) is an edge in a tree-like cycle component of $\mathfrak{A}'$;*

   if (a', b') is an edge in a tree-like cycle component of $\mathfrak{A}$, then (a', b') is an edge in a tree-like cycle component of $\mathfrak{A}'$;
5. *there is an element $c \in silent^{\mathfrak{A}'}_{\tau_1}(b)$, where $\tau_1 = \mathrm{tp}^{\mathfrak{A}}[a]$*
6. *if there is an element $c' \in silent^{\mathfrak{A}}_{\tau_1}(b')$, then $c' \in silent^{\mathfrak{A}'}_{\tau_1}(b')$*

Proof. According to Lemma 8, the size of the set $free^{\mathfrak{A}}_{\tau_1}(b)$ is at least $2 \cdot Z$. If there is an element $c' \in silent^{\mathfrak{A}}_{\tau_2}(b')$, define $\Upsilon = free^{\mathfrak{A}}_{\tau_1}(b) \setminus \{c'\}$, else $\Upsilon = free^{\mathfrak{A}}_{\tau_1}(b)$. Since the model $\mathfrak{A}$ is finite, it is always possible to determine if for a give element $b' \in \mathfrak{A}$ there exists an element $c' \in silent^{\mathfrak{A}}_{\tau_1}(b')$ or not by inspecting the 2-types between the element b' and all the other elements of the model. The remainder of the proof is independent of whether $\Upsilon = free^{\mathfrak{A}}_{\tau_1}(b) \setminus \{c'\}$ (if the element $c' \in silent^{\mathfrak{A}}_{\tau_2}(b')$ exists) or $\Upsilon = free^{\mathfrak{A}}_{\tau_1}(b)$.

The following list of statements about $\mathfrak{A}$ represents various conditions that can either be true or false for the model. By considering every combination of the truth values of these statements, we show that either a given combination is impossible or that in a given combination the required model $\mathfrak{A}'$ can always be produced from $\mathfrak{A}$.

A There is an element $c \in \Upsilon$ such that $\text{tp}^{\mathfrak{A}}[b, c] \in s(\omega_3)$ (Figure 4.3).

B There is an element $c \in \Upsilon$ that is not a child of element b.

C There is an element $c \in \Upsilon$ such that there is an element $e_1 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(c)$ that is not a descendant of c.

D There exist distinct elements $c \in \Upsilon$, $e_1 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(c)$, $e_2 \in \Upsilon$ and $e_3 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(e_2)$ such that $\text{tp}^{\mathfrak{A}}[c, e_3] \in s(\omega_3)$.

E There exist distinct elements $c \in \Upsilon$, $e_1 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(c)$, $e_2 \in \Upsilon$ and $e_3 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(e_2)$ such that $\text{tp}^{\mathfrak{A}}[c, e_3] \in \mu^{\rightarrow}(\omega_3)$.

F There exist distinct elements $c \in \Upsilon$, $e_1 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(c)$, $e_2 \in \Upsilon$ and $e'_3 \in \mathit{silent}^{\mathfrak{A}}_{\tau_2}(e_2)$ such that $\text{tp}^{\mathfrak{A}}[c, e_3] \in \mu^{\leftarrow}(\omega_3)$.

The following list of cases shows how $\mathfrak{A}'$ can be constructed from $\mathfrak{A}$.

Case: A is true. Since $\text{tp}^{\mathfrak{A}}[b, c]$ is a silent 2-type, c is the required element of 1-type τ_2. The model $\mathfrak{A}'$ is the same as the model $\mathfrak{A}$.

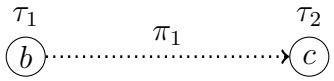


Figure 4.3: Case 1

Case: A is false, B is true. The assumption "not A" implies that for all elements $c \in \Upsilon$ either $\text{tp}[b, c] \in \mu^{\rightarrow}(\omega_4)$, or $\text{tp}[b, c] \in \mu^{\leftarrow}(\omega_4)$, or both. However, according to the part 3 of the definition of $\mathit{free}^{\mathfrak{A}}_{\tau_1}(b)$ in Lemma 8, for all $c \in \mathit{free}^{\mathfrak{A}}_{\tau_1}(b)$ it is the case that $\text{tp}[b, c] \notin \mu^{\rightarrow}(\omega_4)$ Therefore, it can be assumed in the remaining cases that for all $\text{tp}[b, c] \in \mu^{\leftarrow}(\omega_4)$ holds for all $c \in \Upsilon$.

Because $c \in \Upsilon$, there is a forest element e_1 of 1-type τ_1 such that $\text{tp}^{\mathfrak{A}}[c, e_1]$ is a silent 2-type (Figure 4.4).

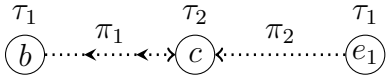


Figure 4.4: Before rewiring (Case 2)

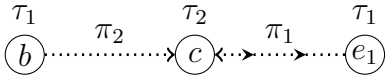


Figure 4.5: After rewiring (Case 2)

A new model $\mathfrak{A}'$ can be created from the existing model $\mathfrak{A}$ by assigning the message-type $\text{tp}^{\mathfrak{A}}[b, c]$ to the pair of elements (e_1, c) and assigning the silent

2-type $\text{tp}^{\mathfrak{A}}[e_1, c]$ to the pair of elements (b, c) (Figure 4.5). In the new model the element b is connected with a silent 2-type to the element c.

Case: A is false, B is false, C is true. Every element $c \in \Upsilon$ is a child of b in this case (Figure 4.6). A new model $\mathfrak{A}'$ can be created from the existing model $\mathfrak{A}$ by assigning the message-type $\text{tp}^{\mathfrak{A}}[b, c]$ to the pair of elements (e_1, c) and assigning the silent 2-type $\text{tp}^{\mathfrak{A}}[e_1, c]$ to the pair of elements (b, c) (Figure 4.7). Since e_1 is not a descendant of c, no new cycles are created in the graph as a result of the reassignment of the edge from (b, c) to (e_1, c). At the same time in the new model element b is connected with a silent 2-type to the element c. The ω_3-rewirable property of $\mathfrak{A}$ is preserved by $\mathfrak{A}'$ in this case, because $e_1 \in first(A)$.

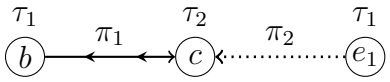


Figure 4.6: Before rewiring (Case 3)

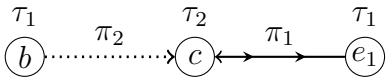


Figure 4.7: After rewiring (Case 3)

Case: A is false, B is false, C is false, D is true. This case is not possible, as the pair of elements (e_3, c) can not be assigned a silent 2-type (Figure 4.8). This is because e_3 is not a descendant of c, since by assumption *not C* element e_3 is a descendant of e_2 and e_2 is not a descendant of c (e_2 and c are both children of b). At the same time, by the assumption *not C* all elements in $silent^{\mathfrak{A}}_{\tau_2}(c)$ are descendants of c.

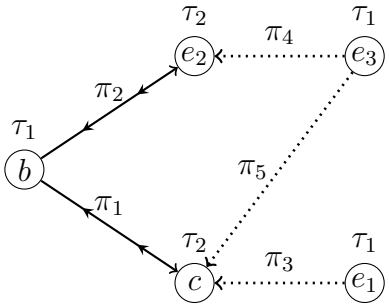


Figure 4.8: Case 4

Case: A is false, B is false, C is false, D is false, E is true. There is no tree edge from element c to element e_3, because element e_3 is a descendant of element e_2 and element c is not a descendant of element e_2 (Figure 4.9). The 2-type $\text{tp}^{\mathfrak{A}}[e_3, c]$ can be assigned to the pair of elements (e_3, e_2) and

$\mathrm{tp}^{\mathfrak{A}}[b, c]$ can be assigned to the pair of elements (e_3, c). As the result of this transformation of $\mathfrak{A}$ the pair of elements (b, c) can be assigned the silent 2-type π_4 (Figure 4.10). As in the previous case, because element e_5 is not a descendant of element e_2 in $\mathfrak{A}$, no cycles are created as a result of the assignment of π_1 to the pair of elements (e_3, c).

The ω_3-rewirable property of $\mathfrak{A}$ is preserved by $\mathfrak{A}'$ in this case, because e_1 is a forest element.

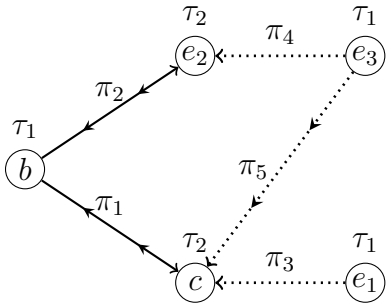


Figure 4.9: Before rewiring (Case 6)

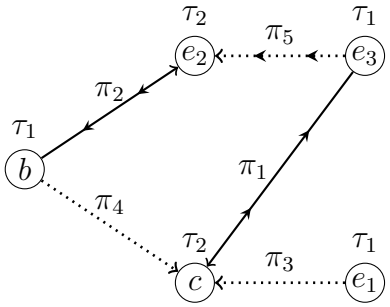


Figure 4.10: After rewiring (Case 6)

Case: A is false, B is false, C is false, D is false, E is false, F is true. In this case $\mathrm{tp}^{\mathfrak{A}}[e_3, c]$ can be assigned to the pair of elements (e_1, c) and $\mathrm{tp}^{\mathfrak{A}}[b, c]$ can be assigned to the pair of elements (e_3, c). As the result of this transformation of $\mathfrak{A}$ the pair of elements (b, c) can be assigned the silent 2-type π_3 (Figure 4.12). Because element e_3 is not a descendant of element c, no cycles are created as a result of the assignment of π_1 to the pair of elements (e_3, c).

The ω_4-rewirable property of $\mathfrak{A}$ is preserved in $\mathfrak{A}'$ in this case, because e_3 is a forest element and the 2-type π_1 stays in the forest of $\mathfrak{A}'$.

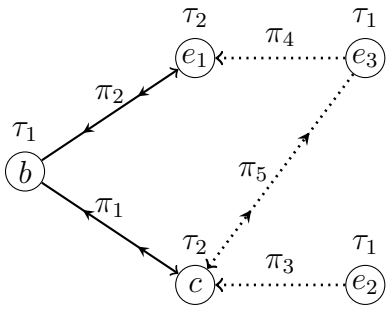


Figure 4.11: Before rewiring (Case 5)

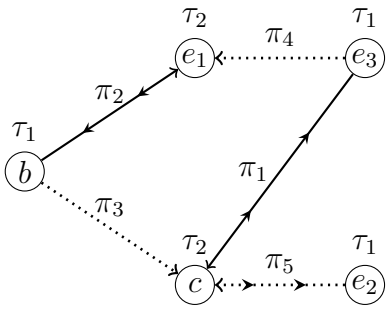


Figure 4.12: After rewiring (Case 5)

Case: A is false, B is false, C is false, D is false, E is false, F is false. To complete the proof we show that it is impossible that none of A, B, C, D, E and F are true at the same time.

This case states that for every element $c \in \Upsilon$, for every element $e_1 \in \Upsilon$ and for every element $e_2 \in silent^{\mathfrak{A}}_{\tau_2}(e_1)$ the 2-type $\mathrm{tp}^{\mathfrak{A}}[c, e_2]$ is an invertible message-type.

According to Lemma 8, there are at least $2 \cdot Z$ elements in Υ. This means that for any element $c \in \Upsilon$ there are at least $2 \cdot Z - 1$ elements $e_1 \in \Upsilon$ for which there exists at least one element $e_2 \in silent^{\mathfrak{A}}_{\tau_1}(e_1)$.

From the above two statements it follows that element c participates in at least $2 \cdot Z$ invertible message types. However, this is impossible, since no element can send more than Z messages.

□

Lemma 10 is explicitly invoked in the beginning of Section 4.4 that deals with the model rewiring and the elimination of cycles in a given model.

Lemma 10. *For any sentence $\phi \in \mathcal{C}^2()$, for any sentence $\omega_3 = \Omega_3(\phi)$, if there exists an ω_3-populated, ω_3-rewirable model $\mathfrak{A} \models \omega_3$ with a positive number of cycles, then there exists an ω_3-populated, ω_3-rewirable model $\mathfrak{A}' \models \omega_3$ such that either:*

1. the number of cycles in $\mathfrak{A}'$ is less than the number of cycles in $\mathfrak{A}$

or

2. the number of cycles in $\mathfrak{A}'$ is equal to the number of cycles in $\mathfrak{A}$ and there exists a forest edge (a, b) and a cycle edge (a', b'), such that $\mathrm{st}^{\mathfrak{A}'}[b] = \mathrm{st}^{\mathfrak{A}'}[b']$ and there are two elements c and c' such that $\mathrm{tp}^{\mathfrak{A}'}[b, c] \in s(\omega_3)$, $\mathrm{tp}^{\mathfrak{A}'}[b', c'] \in s(\omega_3)$ and $\mathrm{tp}^{\mathfrak{A}'}[c] = \mathrm{tp}^{\mathfrak{A}'}[c'] = \mathrm{tp}^{\mathfrak{A}'}[a] = \mathrm{tp}^{\mathfrak{A}'}[a']$.

Proof. According to Lemma 4, $\mathfrak{A}$ is arboreal and there exists a positive number of trees in its graph. Since $\mathfrak{A}$ is ω_3-rewirable, it is also the case that for every element in every cycle of $\mathfrak{A}$ there exists an element in the forest of $\mathfrak{A}$ such that the two elements have the same star-type.

Firstly, Lemma 9 is applied to the pair of edges (a, b) and (a', b'). After the invocation of Lemma 9, the ω_3-populated, ω_3-rewirable model $\mathfrak{A}^* \models \omega_3$ is produced from model $\mathfrak{A}$ such that $\mathfrak{A}^* \approx \mathfrak{A}$.

Secondly, if the number of cycles of $\mathfrak{A}^*$ is less than the number of cycles of $\mathfrak{A}$, then the first condition of the Lemma 10 is met and $\mathfrak{A}^*$ is the desired model $\mathfrak{A}'$. Otherwise, the application of Lemma 9 results in a model $\mathfrak{A}^*$ that has the same number of tree-like cycles as $\mathfrak{A}$. It is known in this case that (a, b) remains to be a tree edge in $\mathfrak{A}^*$ and (a', b') remains to be a cycle edge in $\mathfrak{A}^*$.

Moreover, Lemma 9 guarantees that there exists and element c, such that $\mathrm{tp}^{\mathfrak{A}^*}[c] = \mathrm{tp}^{\mathfrak{A}^*}[a] = \mathrm{tp}^{\mathfrak{A}^*}[a']$ and $\mathrm{tp}^{\mathfrak{A}^*}[b, c] \in s(\omega_3)$. If there is no element c', such that $\mathrm{tp}^{\mathfrak{A}^*}[c'] = \mathrm{tp}^{\mathfrak{A}^*}[a] = \mathrm{tp}^{\mathfrak{A}^*}[a']$ and $\mathrm{tp}^{\mathfrak{A}^*}[b', c'] \in s(\omega_3)$, then Lemma 9 is applied again, but this time to the pair of edges (a', b') and (a, b) of $\mathfrak{A}^*$. In the end, the model $\mathfrak{A}^*$ is obtained such that for some tree edge (a, b) and some cycle edge (a', b') in $\mathfrak{A}^*$ that share the same 2-type, there exist two elements c and c' such that $\mathrm{tp}^{\mathfrak{A}^*}[c] = \mathrm{tp}^{\mathfrak{A}^*}[c'] = \mathrm{tp}^{\mathfrak{A}^*}[a] = \mathrm{tp}^{\mathfrak{A}^*}[a']$, $\mathrm{tp}^{\mathfrak{A}^*}[b, c] \in s(\omega_3)$ and $\mathrm{tp}^{\mathfrak{A}^*}[b', c'] \in s(\omega_3)$. This is the required model $\mathfrak{A}'$.

□

4.4 Model rewiring

Given the results established in the previous sections about the properties of the models of the sentences that the translator Ω generates, it is possible to describe how every cycle in these models can be eliminated by splicing the cycles into the forest components of the model graphs.

Lemma 11. *For any sentence $\phi \in \mathcal{C}^2()$, for any sentence $\omega_3 = \Omega_3(\phi)$, if there exists an ω_3-populated ω_3-rewirable model $\mathfrak{A} \models \omega_3$ with a positive number of cycles, then there exists an ω_3-populated ω_3-rewirable model $\mathfrak{A}' \models \omega_3$ such that the number of cycles in $\mathfrak{A}'$ is less than the number of cycles in $\mathfrak{A}$.*

Proof. Suppose there exists a ω_3-rewirable ω_3-populated model $\mathfrak{A} \models \omega_3$ with a positive number of cycles.

Lemma 10 is applied to the model $\mathfrak{A}$ to obtain the model $\mathfrak{A}'$.If $\mathfrak{A}'$ has fewer cycles than $\mathfrak{A}$, then the conditions of the proof are met and $\mathfrak{A}'$ is the desired model.

Otherwise, $\mathfrak{A}$ has the same number of cycles as $\mathfrak{A}$ and there is a forest edge (a, b) and a cycle edge (a', b'), such that $\mathrm{st}^{\mathfrak{A}'}[b] = \mathrm{st}^{\mathfrak{A}'}[b']$ and there are two elements c and c' such that $\mathrm{tp}^{\mathfrak{A}'}[b, c] \in s(\omega_3)$, $\mathrm{tp}^{\mathfrak{A}'}[b', c'] \in s(\omega_3)$ and $\mathrm{tp}^{\mathfrak{A}'}[c] = \mathrm{tp}^{\mathfrak{A}'}[c'] = \mathrm{tp}^{\mathfrak{A}'}[a] = \mathrm{tp}^{\mathfrak{A}'}[a']$. Then the required model $\mathfrak{A}'$ is produced from $\mathfrak{A}$ by selecting a cycle and splicing it into a tree in the following way.

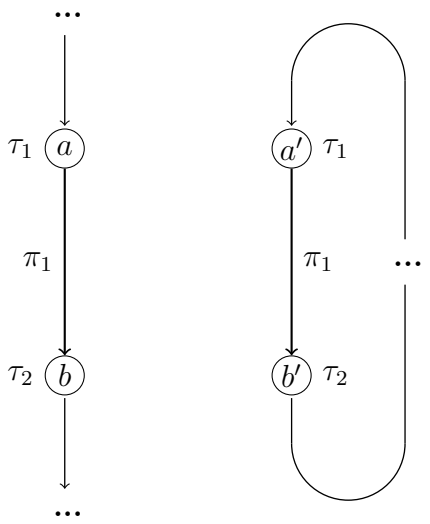


Figure 4.13: Before changing 2-types

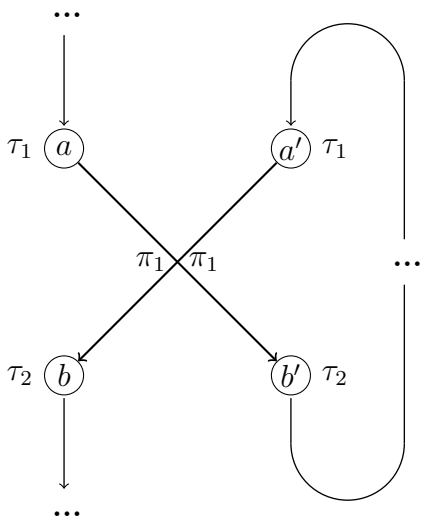


Figure 4.14: After changing 2-types

Given the above configuration, it is possible to make a local modification to the model and embed the cycle containing the edge (a', b') into the tree containing the edge (a, b). This is achieved by discarding the 2-type π_1 between the pair of elements (a', b') and assigning π_1 to the pair of elements (a', b) instead. At the same time the 2-type π_1 is discarded for the pair of elements (a, b) and assigned to the pair of elements (a, b').

The result of this alteration is pictured in Figure 4.14. The edges of the cycle that contained (a', b') are now embedded in the tree (element a is the parent of element b', element b is a child of element a', element a' is a descendant of element b' and element b is a descendant of element a).

However, after the above alteration of $\mathfrak{A}$ the resulting model $\mathfrak{A}'$ is incomplete for two reasons. Firstly, the pairs of elements (a, b) and (a', b') require the assignment of new 2-types, since the original 2-types for these pairs of elements have been discarded during the rewiring. Secondly, the assignment of 2-type π_1 to (a, b') and (a', b) overwrites the information about the original 2-types between these pairs of elements. *This may result in* $\mathfrak{A}'$ *violating the existential constraints of* ω_3. The following list of cases demonstrates how to complete the construction of $\mathfrak{A}'$ so that it preserves satisfiability of ω_4.

1. In Case 1 before the rewiring of the edges (Figure A.1) the 2-types $\mathrm{tp}[a, b']$ and $\mathrm{tp}[b, a']$ are both silent.

   Figure A.2 shows the assignment of 2-types to the elements after the rewiring

of the edges. In this configuration the 2-type π_1 is assigned to pair of elements (a, b) before the rewiring and is assigned to the pair of elements (a, b') after the rewiring. Similarly for (a', b') and (a', b).

Assigning π_2 to the pair of elements (a, b) and π_3 to the pair of elements (a', b') completes the construction of the new model $\mathfrak{A}'$. It is possible to change the assignment of π_2 from (a', b) to (a, b), because $\mathrm{tp}[a] = \mathrm{tp}[a']$ (similarly for π_3).

As the result, all four elements have the same number of existential witnesses as before the rewiring. At the same time the assignment of all 2-types to the elements of the model is specified and is consistent with the 1-types of these elements.

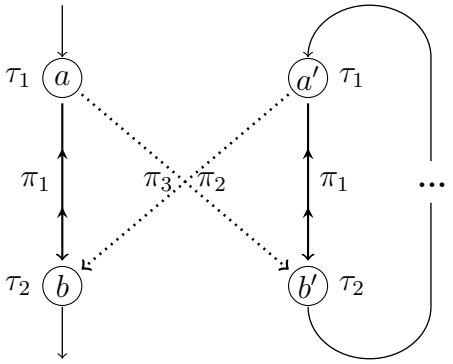


Figure 4.15: Case 1 before rewiring

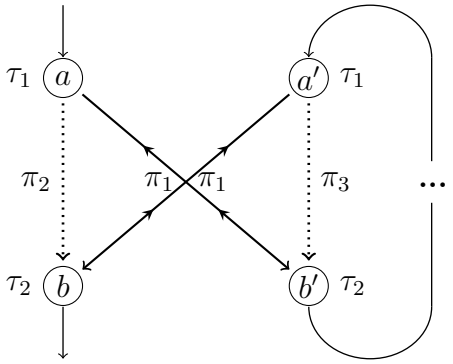


Figure 4.16: Case 1 after rewiring

2. Case 2 before the rewiring (Figure A.3) is similar to Case 1, except that the 2-type π_2 is a forward message-type and not a silent 2-type.

   After the rewiring (Figure A.4) the 2-type π_1 is assigned to the pair of elements (a, b') in place of π_2. This means that element b' is no longer the existential witness of element a. To provide element a with the missing existential witness, π_2 gets assigned to the pair of elements (a, b). The assignment of the silent 2-type π_3 to the pair of elements (a', b') completes the construction of the model $\mathfrak{A}'$.

3. Case 3 (Figure A.5 and Figure A.6) is analogous to Case 2, except that it is π_3 that is the forward message-type and π_2 is the silent 2-type.

   Because the rewiring in Case 3 is carried out in a similar manner to Case 2, such cases are called symmetric.

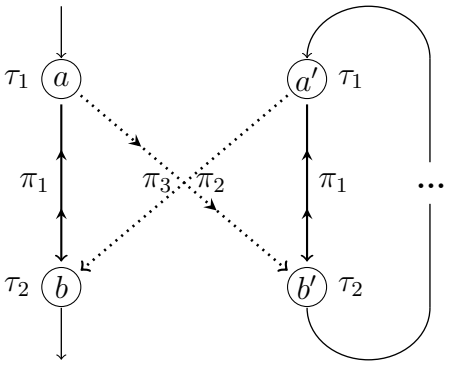


Figure 4.17: Case 2 before rewiring

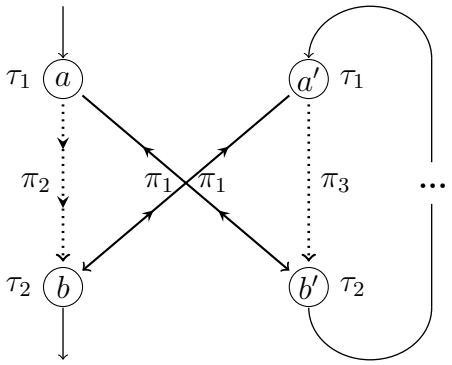


Figure 4.18: Case 2 after rewiring

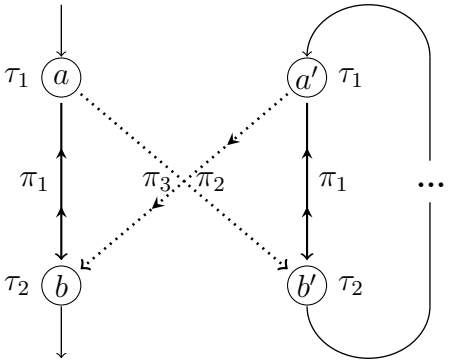


Figure 4.19: Case 3 before rewiring

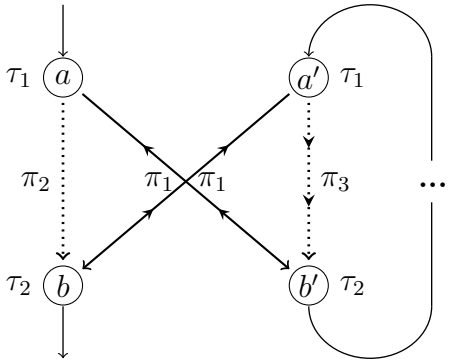


Figure 4.20: Case 3 after rewiring

4. Case 4 (Figure A.7 and Figure A.8) combines Case 2 and Case 3, since both π_2 and π_3 are forward message-types.

5. Case 5 (Figure A.9 and Figure A.10) is similar to Case 2 except that π_3 is not a forward message-type, but a reverse message-type.

   In order to provide element b with an existential witness after the rewiring, π_3 is assigned to the pair of elements (a, b). Is is possible to change the source element of π_2 from element a' to element a, because $\mathrm{tp}[a] = \mathrm{tp}[a']$.

6. Case 6 is symmetric to Case 5. Because of this, the diagram for Case 6 is omitted.

7. Case 7 (Figure A.13 and Figure A.14) is the combination of Case 5 and Case 6 with respect to the non-invertible message-types π_2 and π_3.

8. Case 8 (Figure A.15 and Figure A.16) describes a situation in which π_2 is a forward message-type and π_3 is a reverse message-type. In this configuration the rewiring cannot be carried out as a combination of the previous cases,

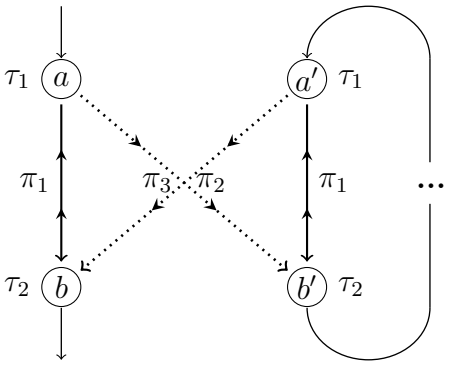


Figure 4.21: Case 4 before rewiring

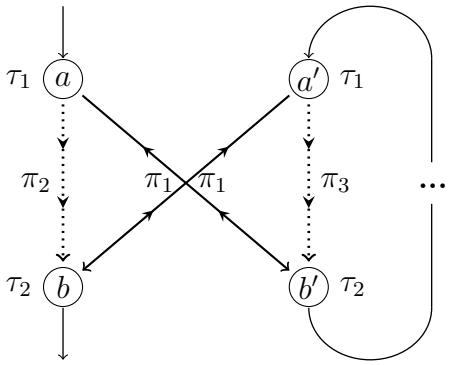


Figure 4.22: Case 4 after rewiring

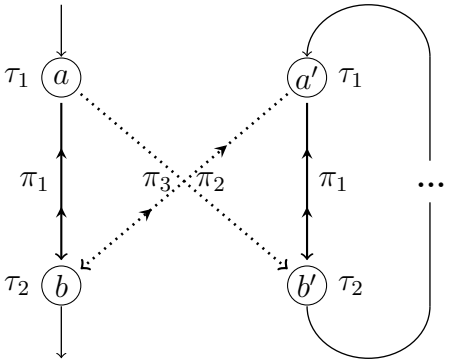


Figure 4.23: Case 5 before rewiring

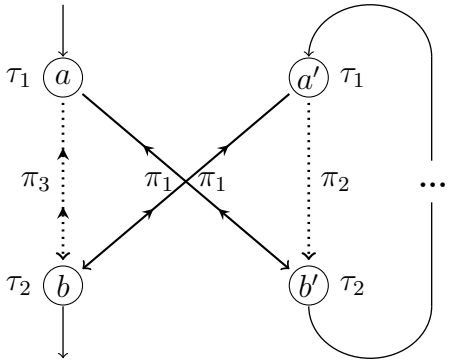


Figure 4.24: Case 5 after rewiring

since the pair of elements (a, b) cannot be assigned π_2 and π_3 at the same time.

However, for element b there exists an element d of 1-type τ_1 such that $\mathrm{tp}[b, d]$ is silent. This fact allows us to assign π_2 to the pair of elements (a, b) and π_3 to the pair of elements (b, d). As the result of this rewiring the requirements for existential witnesses of elements a and b are satisfied.

9. Case 9 is symmetric to Case 8. Because of this, the diagram for Case 9 is omitted.

10. In Case 10 the 2-type π_2 is an invertible message-type and the 2-type π_3 is silent. This means that elements a and b' use each other as existential witnesses in π_2. It is not possible to assign π_2 either to the pair of elements (a, b) after the rewiring, since this would alter the number of existential witnesses of elements b and b'.

    Since $\mathrm{tp}[b'] = \mathrm{tp}[b]$, there is an element c such that $\mathrm{tp}[c, b]|_{\omega_1} = \mathrm{tp}[a, b']|_{\omega_1}$.

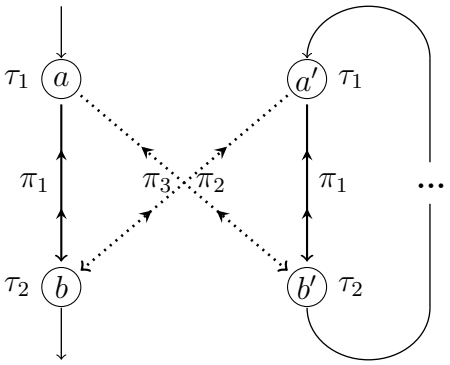


Figure 4.25: Case 7 before rewiring

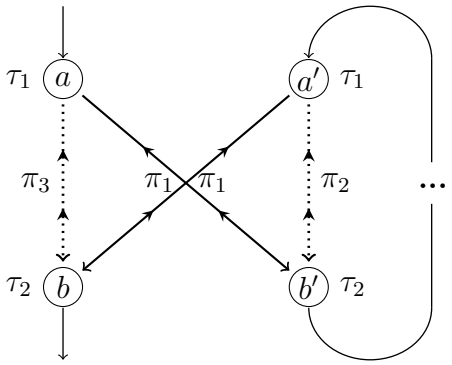


Figure 4.26: Case 7 after rewiring

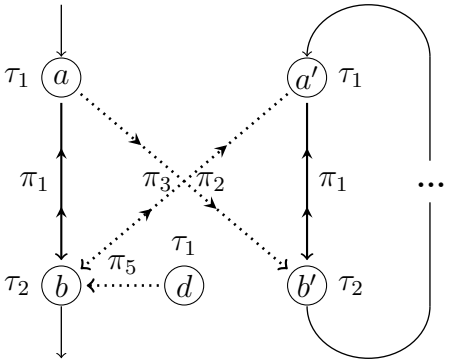


Figure 4.27: Case 8 before rewiring

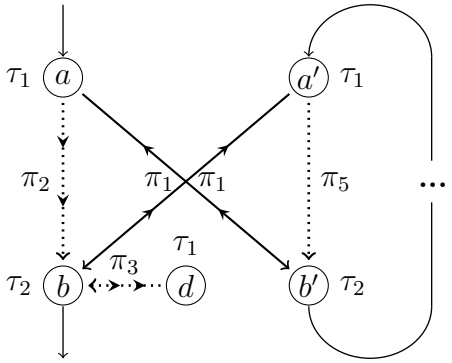


Figure 4.28: Case 8 after rewiring

The existence of the element c makes it possible to perform the rewiring in this configuration. There are three possibilities with regards to $\mathrm{tp}[c, b']$, as it can be a silent 2-type, a forward message-type or a reverse message-type. This 2-type cannot be an invertible message-type due to the fact that $\mathfrak{A}$ is ω_3-chromatic, $\mathrm{tp}[a] = \tau[b']$ and $\mathrm{tp}[a, b']$ is already an invertible message-type. Each of these three cases has to be addressed individually.

(a) In Case 10(a) (Figure A.19 and Figure A.20) the 2-type $\mathrm{tp}[c, b']$ is silent. Hence, the 2-type π_2 can be assigned to (a, b) and the 2-type π_2' can be assigned to (a, b), providing the missing existential witnesses to the elements a and b without changing the number of existential witnesses of element c.

Even though the 2-types π_2 and π_2' are the same only up to the ω_3 signature, the element b does not lose any existential witnesses after π_2' is reassigned to (c, b'). At the same time, no additional messages are sent by the element b as the result of the assignment of π_2 to (a, b).

The silent types π_4 and π_3 are assigned to the pairs of elements (c, b) and (a', b'), completing the construction of the model.

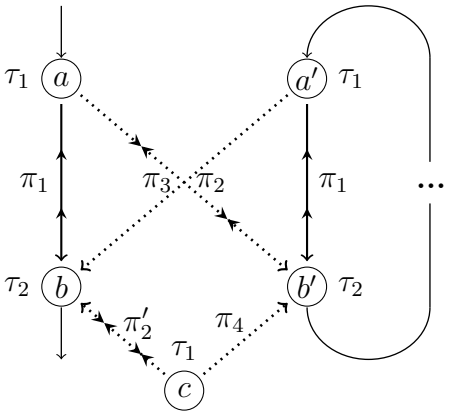


Figure 4.29: Case 10(a) before rewiring

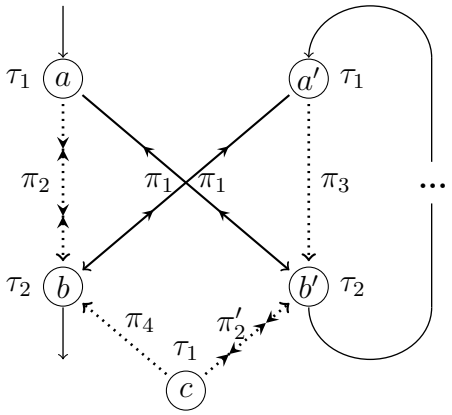


Figure 4.30: Case 10(a) after rewiring

(b) In Case 10(b) (Figure A.21 and Figure A.22) the 2-type $\mathrm{tp}[c, b']$ is a reverse message-type. There cannot be a tree edge from the element c to the element b' before the rewiring, because b' already has an incoming tree edge from the element a'. After the rewiring, π_4 is assigned to (a', b') and element a' becomes the existential witness of element b'.

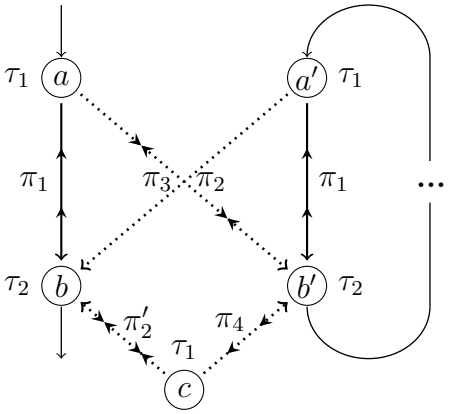


Figure 4.31: Case 10(b) before rewiring

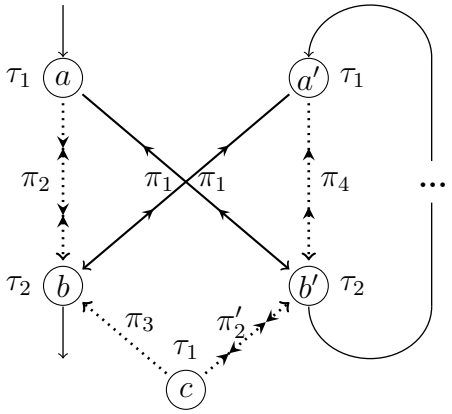


Figure 4.32: Case 10(b) after rewiring

(c) In Case 10(c) (Figure A.23 and Figure A.24) the 2-type $\mathrm{tp}[c, b']$ is a forward message-type. There cannot be a tree edge from the element b' to the element c, because if this was the case, then $\mathrm{tp}[a'] \neq \mathrm{tp}[c]$, due to the alternating property of $\mathfrak{A}$, and this contradicts the assumption that $\mathrm{tp}[a, b] = \mathrm{tp}[a', b']$. After the rewiring, π_4 is assigned to (c, b) and the element b becomes the existential witness of element c.

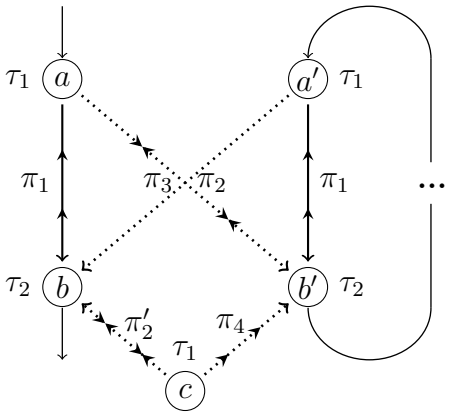


Figure 4.33: Case 10(c) before rewiring

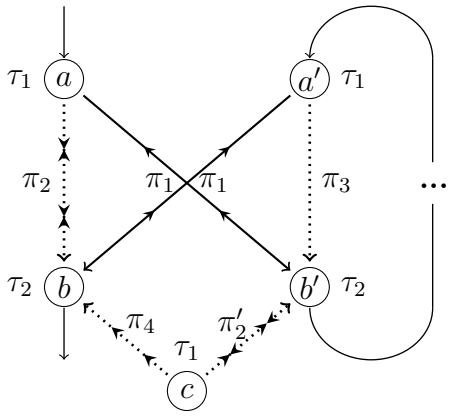


Figure 4.34: Case 10(c) after rewiring

11. Case 11 is analogous to Case 10. The only difference is that the 2-type $\mathrm{tp}[a', b]$ is a forward message-type and not a silent 2-type. The rewiring of this configuration consists of three distinct cases as well.

    (a) In Case 11(a) (Figure A.25 and Figure A.26) the assignment of the 2-types is similar to Case 10(a), except that after the rewiring the element a' uses the element b' as the existential witness in the context of the π_3.

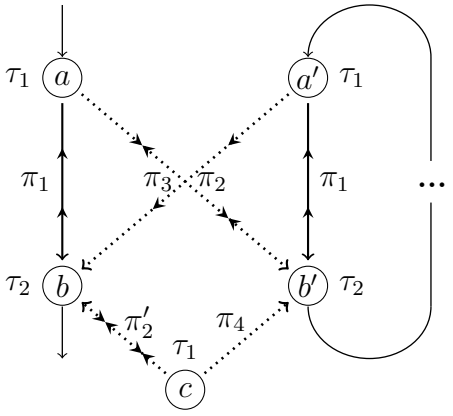


Figure 4.35: Case 11(a) before rewiring

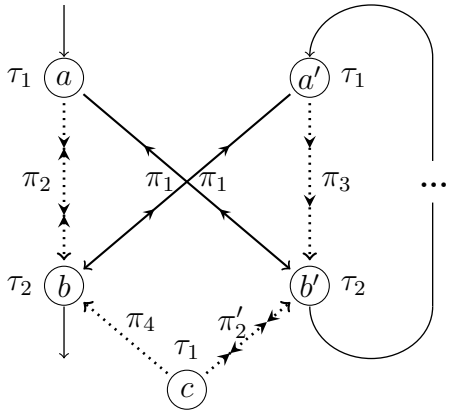


Figure 4.36: Case 11(a) after rewiring

    (b) Case 11(b) (Figure A.27 and Figure A.28) does not allow for the rewiring to be carried out in the same way as in case Case 10(b), because (a', b') is assigned π_3. This prevents the assignment of the 2-type π_4 to that pair of elements. However, for the element b' there exists an element d of the 1-type τ_1 such that $\mathrm{tp}[b, d]$ is silent. This

makes it possible to assign π_4 to the pair of elements (b', d) and provide the element b' with the missing existential witness.

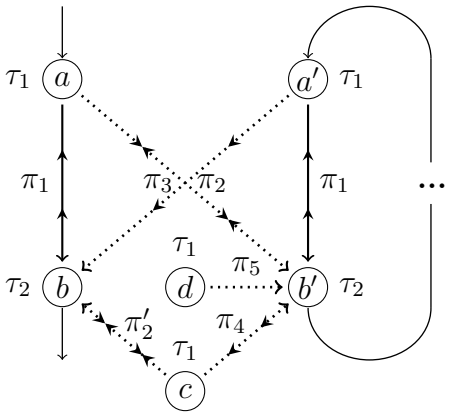


Figure 4.37: Case 11(b) before rewiring

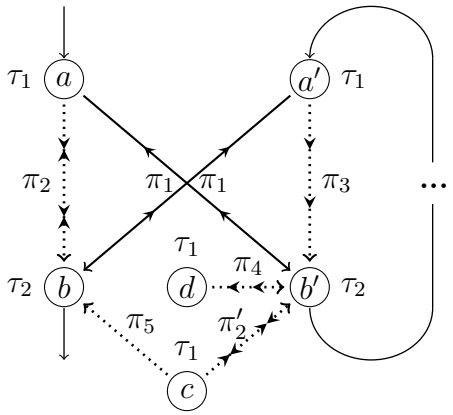


Figure 4.38: Case 11(b) after rewiring

(c) In Case 11(c) (Figure A.29 and Figure A.30) the assignment of the 2-types is analogous to Case 10(c). As in Case 11(a), after the rewiring the element c uses the element b as the existential witness in the context of π_4.

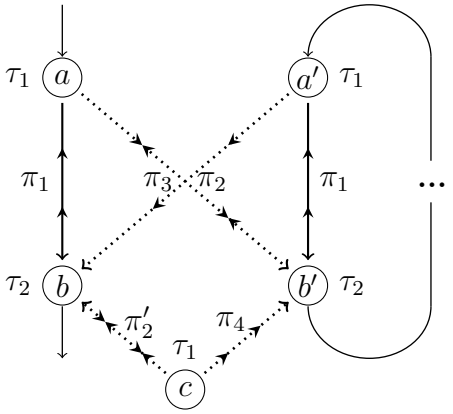


Figure 4.39: Case 11(c) before rewiring

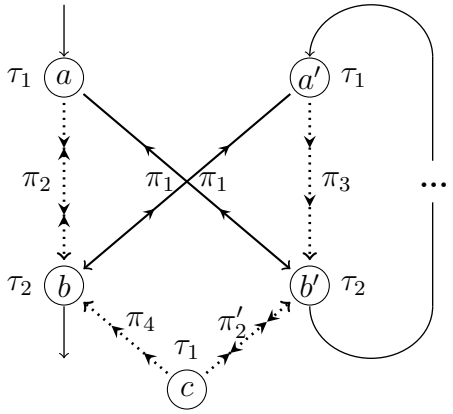


Figure 4.40: Case 11(c) after rewiring

12. Case 12 is similar to Case 11, except that π_3 is not a forward message-type, but a reverse message-type.

(a) In Case 12(a) (Figure A.35 and Figure A.32) after the rewiring the element b uses the element c as the existential witness in the context of π_3.

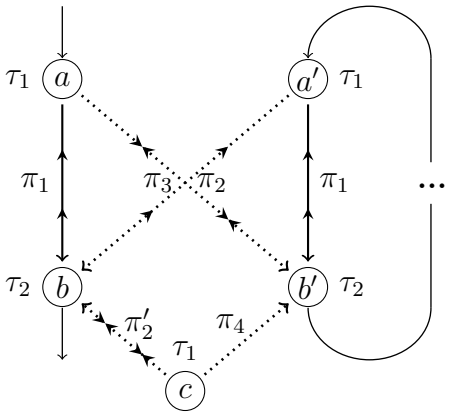


Figure 4.41: Case 12(a) before rewiring

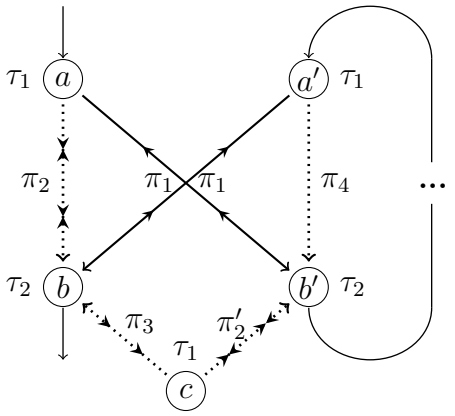


Figure 4.42: Case 12(a) after rewiring

(b) Case 12(b) (Figure A.33 and Figure A.34) is similar to Case 11(c). In this configuration the non-invertible message π_3 is relocated during the rewiring from the pair of elements (a', b) to the pair of elements (c, b). In addition to that, the element b' uses the element a' after the rewiring as the existential witness for the 2-type π_4.

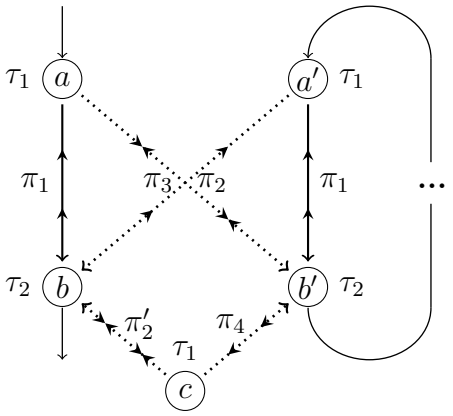


Figure 4.43: Case 12(b) before rewiring

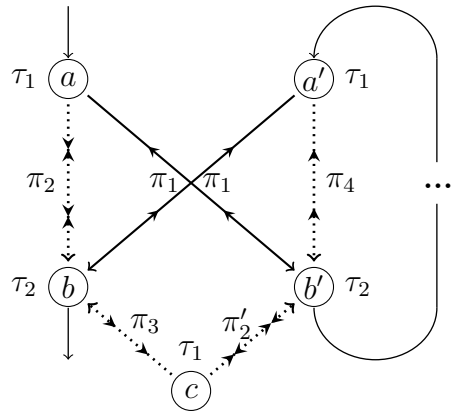


Figure 4.44: Case 12(b) after rewiring

(c) Case 12(c) is similar to case Case 11(b) in that two the auxiliary elements c and d are required to satisfy the existential witness requirements of all the elements in the configuration.

After the rewiring the forward message-type π_4 can only be assigned to the pair of elements (c, b). As the result, the element d becomes the existential witness of the element b in the context of π_3.

13. Case 13 is symmetrical in each of the three instances to Case 10.

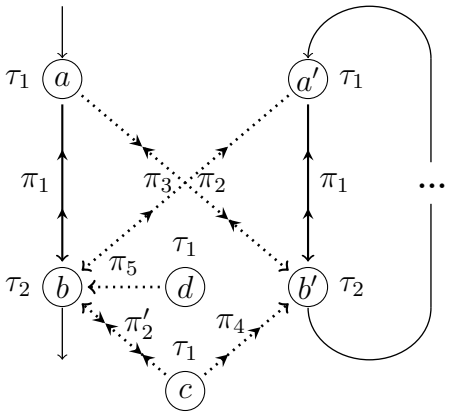


Figure 4.45: Case 12(c) before rewiring

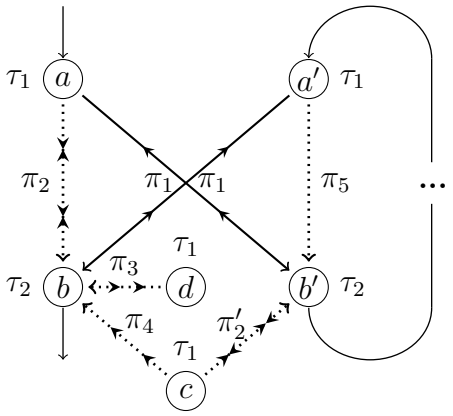


Figure 4.46: Case 12(c) after rewiring

14. Case 14 is symmetrical in each of the three instances to Case 11.

15. Case 15 is symmetrical in each of the three instances to Case 12.

16. Case 16 contains invertible message-types between both (a, b') and (a', b). If $\mathrm{tp}[a, b'] \neq \mathrm{tp}[a', b]$, then there is an element c such that $\mathrm{tp}[a', b] = \mathrm{tp}[b, c]$. However, this contradicts the ω_3-chromatic property of $\mathfrak{A}$. Therefore, $\mathrm{tp}[a, b'] = \mathrm{tp}[a', b]$.

    The above fact allows us to assign the 2-types of (a, b') and (a', b) to the pairs of elements (a, b) and (a', b') and satisfy the requirements for existential witnesses of each of the four elements.

17. Case 17 is the same as Case 1, except that the 2-type π_1 is an invertible message-type. The final assignment of the 2-types is identical between the two cases. In a similar manner Case 18 to Case 25 repeat the rewiring

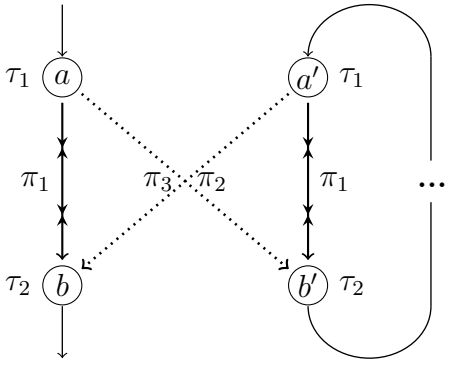


Figure 4.47: Case 17 before rewiring

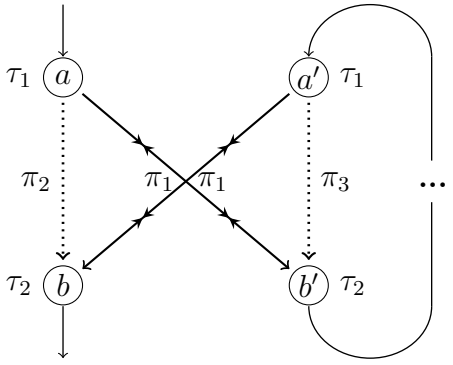


Figure 4.48: Case 17 after rewiring

of Case 2 to Case 9. These cases are added to the appendix only for the

purpose of completeness and do not introduce any new ways of allocating 2-types after the rewiring of the model. This allows us to cover the rewiring of the edges that are not only the reverse message-types, but also the edges that are the invertible message-types.

Only nine such cases are considered in total. It is not possible for π_1 to be an invertible message-type and an invertible message-type to occur between the pairs of elements (a', b) or (a, b') at the same time. This is due to the fact that $\mathfrak{A}$ is ω_3-chromatic and any element of the model $\mathfrak{A}$ cannot send invertible message-types to two or more elements of the same 1-type.

□

4.5 Summary

Finally, the desired complexity result can be formulated, following the discussion of the relevant lemmas in the previous sections of this chapter.

Lemma 12. *If there exists a sentence $\omega = \Omega(\phi)$ that is finitely satisfiable in $\mathcal{C}^2()$, then ϕ is finitely satisfiable in $\mathcal{C}^2(\downarrow^+)$.*

Proof. If ω is finitely satisfiable in some tree model, the implication of the lemma holds trivially, because ϕ is one of the conjuncts of ω. Otherwise, ω is finitely satisfiable in some model $\mathfrak{A}$ that contains a positive number of cycles. Consider the model $\mathfrak{A}_* = \mathfrak{A}|\Sigma(\omega_3)$, where ω_3 is generated by Ω_3 as part of the application of Ω to ϕ. Evidently $\mathfrak{A}_*$ is ω_3-rewirable ω_3-populated and $\mathfrak{A}_* \models \omega_3$. According to Lemma 11, $\mathfrak{A}_*$ can be modified repetitively until a model $\mathfrak{A}'_* \models \omega_3$ is produced that contains no cycles. The required tree model $\mathfrak{A}' \models \phi$ is then defined as $\mathfrak{A}' = \mathfrak{A}'_*|\Sigma(\phi)$. □

Theorem 2. *Finite satisfiability of $\mathcal{C}^2(\downarrow^+)$ is in NEXPTIME.*

Proof. The decidability of finite satisfiability of $\mathcal{C}^2$ is a consequence of Lemma 7 and Lemma 12. From the definition of a translator and Lemma 1 the NEXPTIME complexity bound for the finite satisfiability of $\mathcal{C}^2(\downarrow^+)$ follows. □

The core of the proof of the decidability of finite satisfiability of $\mathcal{C}^2(\downarrow^+)$ resides in the observation that this problem can be transformed into the problem of finite satisfiability of $\mathcal{C}^2()$. Given a sentence ϕ it is sufficient to generate an

auxiliary sentence based on ϕ such that their conjunction is going to be finitely satisfiable in $\mathcal{C}^2()$ whenever ϕ itself is finitely satisfiable in some forest model. This tells us of the power of the language of $\mathcal{C}^2()$ to speak of the models of its own sentences. Through defining numerous restrictions on the models of a given sentence it is possible to arrive to a situation when any of the models that passes all these restrictions can be turned into the model of a desired kind by a series of predetermined local transformations.

The key novelty of this proof is found in how the translator Ω_{frst} is defined in Subsection 4.2.5. In general it is not feasible to describe the structures of an unbounded finite size with a bounded number of sentences. However, by nondeterministically guessing as the relation $\Downarrow$ the graph minor of the forest component of a model that satisfies the given sentence, it is possible to describe the relative position of the elements with the desired 1-types in the forest component.

Chapter 5

Conclusions

Analysis of existing complexity results

The systematic analysis of the existing results in the area of satisfiability of two-variable first-order logic and its extensions that was carried out in Chapter 2 leads to two main conclusions.

On one hand side, this field of research is well structured. There is a clear theme of taking the basic properties of binary relations and graphs that appear most often in mathematics and that cannot be expressed in the languages $\mathcal{FO}^2()$ or $\mathcal{FO}^2[]$ themselves and adding them to the two-variable logic as special predicates. Because transitivity is a component of most of these properties of relations and because transitivity is not expressible in $\mathcal{FO}^2()$, the available results in the field can be readily compared to each other in terms of the expressiveness of the logics under study and their corresponding complexities.

From this point of view the structure of the field can be expanded in the future to include new properties of binary relations that have not been considered before or that have very few results available. Examples of such properties can be deterministic transitive closure [Charatonik et al., 2014], strict partial order [Kieronski et al., 2014] or the connectedness property that requires the graph of a given binary predicate to be connected (which has never been considered before).

One important type of binary relations for which no decidability results have been obtained is partial orders. Resolving the decidability of satisfiability and finite satisfiability for two-variable first-order logic and monadic two-variable first-order logic with one or more partial order relations and their successor relations would bridge the gap between strictly more expressive two-variable logics with

transitive relations and strictly less expressive two-variable logics with linear orders.

On the other hand, many subareas of research in the field of decidability of satisfiability of two-variable logics lack a significant number of results concerning important general fragments. For example, extending guarded two-variable fragment with special binary predicates has been partially investigated for transitive relations, linear order relations, equivalence relations and horizontal and vertical tree navigation relations. But no results have been obtained for guarded two-variable logics with total preorders.

Conditions for completing the research in two-variable logics

After the publication of [Börger et al., 2001] the quest for establishing the complexity of satisfiability of all prefix fragments of first-order logic has been finalised. A prospect of achieving a comparable state of affairs in the area of decidability of satisfiability of two-variable first-order logic with special binary predicates is possible.

One subfield of this area that has been fully completed concerns extending two-variable logics with equivalence relations. The precise complexity classes for both satisfiability and finite satisfiability has been established for two-variable first-order logic, monadic two-variable first-order logic, guarded two-variable first-order logic and two-variable first-order logic with counting with equivalence relations, including the languages: $\mathcal{FO}^2[\sim]$, $\mathcal{FO}^2[\sim_1, \sim_2]$, $\mathcal{FO}^2[\sim_1, \sim_2, \sim_3]$, $\mathcal{FO}^2(\sim)$, $\mathcal{FO}^2(\sim_1, \sim_2)$, $\mathcal{FO}^2(\sim_1, \sim_2, \sim_3)$, $\mathcal{FO}^2(\widehat{\sim})$, $\mathcal{FO}^2(\widehat{\sim}_1, \widehat{\sim}_2)$, $\mathcal{FO}^2[<, \sim]$, $\mathcal{GF}^2(\sim_1, \sim_2)$, $\mathcal{GF}^2[\sim_1, \sim_2, \sim_3]$, $\mathcal{C}^2(\sim)$, $\mathcal{C}^2(\sim_1, \sim_2)$. The minimum number of equivalence predicates that leads to undecidability has been established for every type of logic (monadic, guarded, counting), making it unnecessary to consider logics with a higher number of equivalence relations.

If what has been achieved for the two-variable logics with equivalence relations is carried out for the rest of the types of special binary predicates, the broad goal of the whole field of research into the decidability of satisfiability of two-variable logics can largely be called completed.

Complexity of two-variable logic with tree-navigation relations

The first most immediate outstanding problem coming from the work undertaken in this thesis is establishing finite satisfiability of two-variable first-order logic with

counting and two independent successors of two vertical tree navigation relations, denoted $\mathcal{C}^2(\downarrow_1^+, \downarrow_2^+)$. The proof of this result follows naturally from the work by [Charatonik and Witkowski, 2013] and the proof in Chapter 4. Employing the techniques from both of the works is expected to yield the same NEXPTIME upper complexity bound, as for the finite satisfiability of $\mathcal{C}^2(\downarrow^+)$.

The second problem that the work in this thesis leads to is determining the status of satisfiability problem for $\mathcal{C}^2(\downarrow^+)$ and $\mathcal{C}^2(\downarrow_1^+, \downarrow_2^+)$. It is estimated that an NEXPTIME upper complexity bound applies to the two problems, as nothing leads to suggest that these problems reside in a higher complexity class.

Finally, it is not known at the moment what number of independent vertical tree navigation relations or their successors is required to be added to $\mathcal{FO}^2[]$ for the logic to become undecidable. As it was demonstrated in [Benaim et al., 2013], adding two independent vertical tree navigation relations and the corresponding successor relations to the monadic two-variable first-order logic results in a language with EXPSPACE-complete finite satisfiability problem. Therefore, it makes sense to consider the question if the presence of three or more independent vertical tree navigation relations would make the finite satisfiability problem of two-variable first-order logic with counting non-recursive.

It should also be noted that closing the complexity gap of the finite satisfiability problem of $\mathcal{FO}^2[\sim, \downarrow^+, \rightarrow^+]$ remains highly desirable and, perhaps, can be achieved in the future by showing NEXPTIME upper bound for the finite satisfiability of $\mathcal{C}^2(\sim, \downarrow^+, \rightarrow^+)$.

Bibliography

[Benaim et al., 2013] Benaim, S., Benedikt, M., Charatonik, W., Kieronski, E., Lenhardt, R., Mazowiecki, F., and Worrell, J. (2013). Complexity of two-variable logic on finite trees. In *Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 74–88. Springer.

[Benedikt et al., 2012] Benedikt, M., Lenhardt, R., and Worrell, J. (2012). Verification of two-variable logic revisited. In *Ninth International Conference on Quantitative Evaluation of Systems*, pages 114–123. IEEE Computer Society.

[Bojańczyk et al., 2009] Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., and Segoufin, L. (2009). Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3).

[Bojańczyk et al., 2006] Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., and David, C. (2006). Two-variable logic on words with data. In *21th Annual Symposium on Logic in Computer Science*, pages 7–16. ACM.

[Börger et al., 2001] Börger, E., Grädel, E., and Gurevich, Y. (2001). *The Classical Decision Problem*. Perspectives in mathematical logic. Springer.

[Charatonik et al., 2014] Charatonik, W., Kieroński, E., and Mazowiecki, F. (2014). Decidability of weak logics with deterministic transitive closure. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 29:1–29:10. ACM.

[Charatonik and Witkowski, 2013] Charatonik, W. and Witkowski, P. (2013). Two-variable logic with counting and trees. In *28th Annual Symposium on Logic in Computer Science*, pages 73–82. IEEE Computer Society.

[Ehrenfeucht, 1961] Ehrenfeucht, A. (1961). An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49(2):135–140.

[Etessami et al., 2002] Etessami, K., Vardi, M., and Wilke, T. (2002). First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295.

[Figueira et al., 2014] Figueira, D., Figueira, S., and Areces, C. (2014). Basic model theory of XPath on data trees. *17th International Conference on Database Theory*, pages 50–60.

[Fürer, 1984] Fürer, M. (1984). The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In *Logic and Machines: Decision Problems and Complexity*, volume 171 of *Lecture Notes in Computer Science*, pages 312–319. Springer.

[Geerts and Fan, 2005] Geerts, F. and Fan, W. (2005). Satisfiability of XPath queries with sibling axes. In *10th International Conference on Database Programming Languages*, pages 122–137. Springer.

[Gottlob et al., 2005] Gottlob, G., Koch, C., and Pichler, R. (2005). Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491.

[Grädel et al., 1997a] Grädel, E., Kolaitis, P., and Vardi, M. (1997a). On the decision problem for two-variable first-order logic. *The Bulletin of Symbolic Logic*, 3(1):53–69.

[Grädel and Otto, 1999] Grädel, E. and Otto, M. (1999). On logics with two variables. *Theoretical Computer Science*, 224(1):73–113.

[Grädel et al., 1997b] Grädel, E., Otto, M., and Rosen, E. (1997b). Two-variable logic with counting is decidable. In *12th Annual Symposium on Logic in Computer Science*, pages 306–317. IEEE Computer Society.

[Grädel et al., 1998] Grädel, E., Otto, M., and Rosen, E. (1998). Undecidability results on two-variable logics. In *14th Symposium on Theoretical Aspects of Computer Science*, pages 249–260. Springer.

[Kahr et al., 1962] Kahr, A., Moore, E., and Wang, H. (1962). Entscheidungsproblem reduced to the $\forall\exists\forall$ case. *Proceedings of the National Academy of Sciences of the United States of America*, 48(3):365–377.

[Kieroński, 2003] Kieroński, E. (2003). The two-variable guarded fragment with transitive guards is 2EXPTIME-hard. In *International Conference on Foundations of Software Science and Computation Structures*, pages 299–312. Springer.

[Kieronski, 2005] Kieronski, E. (2005). Results on the guarded fragment with equivalence or transitive relations. In *Computer Science Logic*, volume 3634 of *Lecture Notes in Computer Science*, pages 309–324. Springer.

[Kieronski, 2011] Kieronski, E. (2011). Decidability issues for two-variable logics with several linear orders. In *CSL*, volume 12 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 337–351. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[Kieronski et al., 2012] Kieronski, E., Michaliszyn, J., Pratt-Hartmann, I., and Tendera, L. (2012). Two-variable first-order logic with equivalence closure. In *27th Annual Symposium on Logic in Computer Science*, pages 431–440. IEEE Computer Society.

[Kieronski et al., 2014] Kieronski, E., Michaliszyn, J., Pratt-Hartmann, I., and Tendera, L. (2014). Two-variable first-order logic with equivalence closure. *SIAM Journal on Computing*, 43(3):1012–1063.

[Kieronski and Otto, 2012] Kieronski, E. and Otto, M. (2012). Small substructures and decidability issues for first-order logic with two variables. *The Journal of Symbolic Logic*, 77(3):729–765.

[Kieronski and Tendera, 2009] Kieronski, E. and Tendera, L. (2009). On finite satisfiability of two-variable first-order logic with equivalence relations. In *24th Annual Symposium on Logic In Computer Science*, pages 123–132. IEEE Computer Society.

[Kosaraju, 1982] Kosaraju, S. R. (1982). Decidability of reachability in vector addition systems (preliminary version). In *14th Annual ACM Symposium on Theory of Computing*, pages 267–281. ACM.

[Leroux and Schmitz, 2015] Leroux, J. and Schmitz, S. (2015). Reachability in vector addition systems demystified. *CoRR*, abs/1503.00745.

[Lewis, 1980] Lewis, H. R. (1980). Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21(3):317–353.

[Libkin, 2004] Libkin, L. (2004). *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer.

[Lipton, 1976] Lipton, R. J. (1976). *The reachability problem requires exponential space*. Research Report 62. Department of Computer Science, Yale University.

[Manuel, 2010] Manuel, A. (2010). Two variables and two successors. In *35th International Conference on Mathematical Foundations of Computer Science*, pages 513–524. Springer.

[Manuel et al., 2013] Manuel, A., Schwentick, T., and Zeume, T. (2013). A short note on two-variable logic with a linear order successor and a preorder successor. *CoRR*, abs/1306.3418.

[Manuel and Zeume, 2013] Manuel, A. and Zeume, T. (2013). Two-Variable Logic on 2-Dimensional Structures. In *22nd EACSL Annual Conference on Computer Science Logic*, volume 23 of *Leibniz International Proceedings in Informatics*, pages 484–499. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[Marx, 2004] Marx, M. (2004). XPath with conditional axis relations. In *International Conference on Extending Database Technology*, volume 2992 of *Lecture Notes in Computer Science*, pages 477–494. Springer.

[Marx and de Rijke, 2005] Marx, M. and de Rijke, M. (2005). Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46.

[Moore, 1988] Moore, G. H. (1988). The emergence of first-order logic. *History and philosophy of modern mathematics*, 11:95–135.

[Mortimer, 1975] Mortimer, M. (1975). On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140.

[Otto, 2001] Otto, M. (2001). Two variable first-order logic over ordered domains. *The Journal of Symbolic Logic*, 66(2):685–702.

[Pacholski et al., 1997] Pacholski, L., Szwast, W., and Tendera, L. (1997). Complexity of two-variable logic with counting. In *12th Annual Symposium on Logic in Computer Science*, pages 318–327. IEEE Computer Society.

[Pratt-Hartmann, 2005] Pratt-Hartmann, I. (2005). Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395.

[Pratt-Hartmann, 2010] Pratt-Hartmann, I. (2010). The two-variable fragment with counting revisited. In *17th International Conference on Logic, Language, Information and Computation*, pages 42–54. Springer.

[Pratt-Hartmann, 2017] Pratt-Hartmann, I. (2017). The finite satisfiability problem for two-variable, first-order logic with one transitive relation is decidable. *CoRR*, abs/1707.05558.

[Schwentick and Zeume, 2012] Schwentick, T. and Zeume, T. (2012). Two-variable logic with two order relations. *Logical Methods in Computer Science*, 8(1).

[Scott, 1962] Scott, D. (1962). A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477.

[Szwast and Tendera, 2013] Szwast, W. and Tendera, L. (2013). FO2 with one transitive relation is decidable. In *30th International Symposium on Theoretical Aspects of Computer Science*, volume 20 of *Leibniz International Proceedings in Informatics*, pages 317–328. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[Turing, 1936] Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265.

Appendix A

Rewiring cases

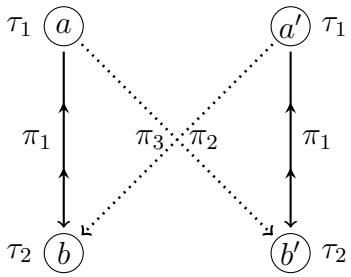


Figure A.1: Case 1 before rewiring

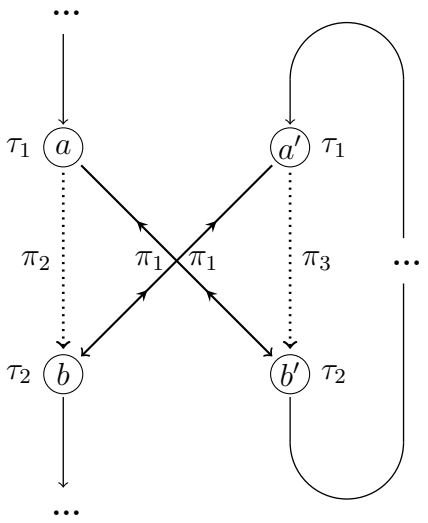


Figure A.2: Case 1 after rewiring

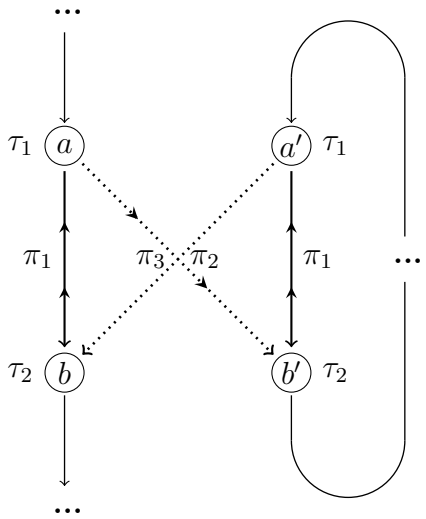


Figure A.3: Case 2 before rewiring

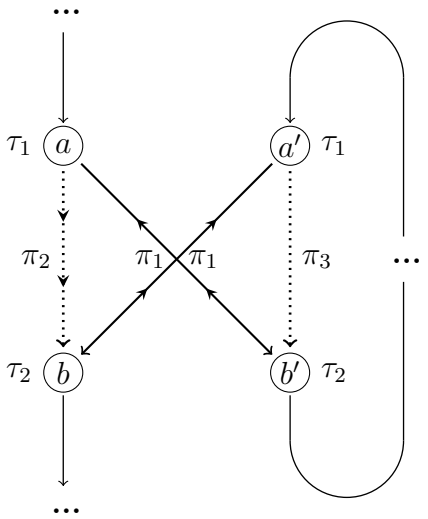


Figure A.4: Case 2 after rewiring

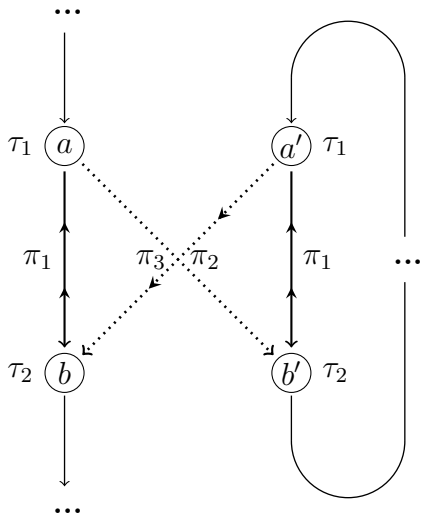


Figure A.5: Case 3 before rewiring

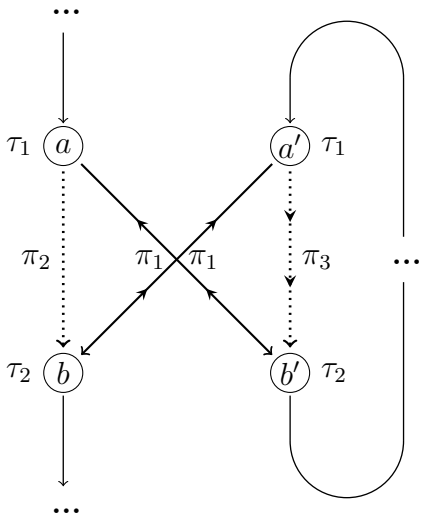


Figure A.6: Case 3 after rewiring

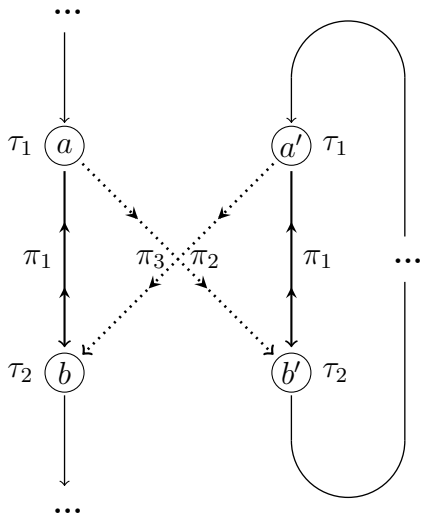


Figure A.7: Case 4 before rewiring

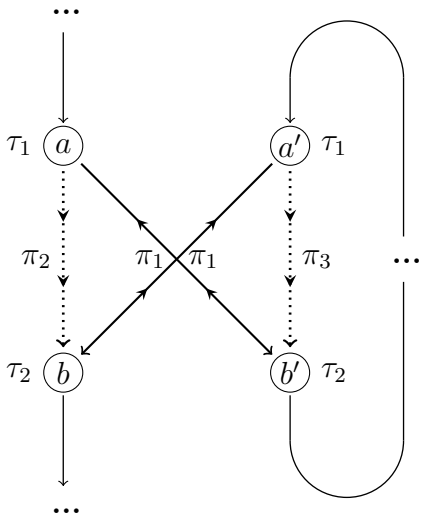


Figure A.8: Case 4 after rewiring

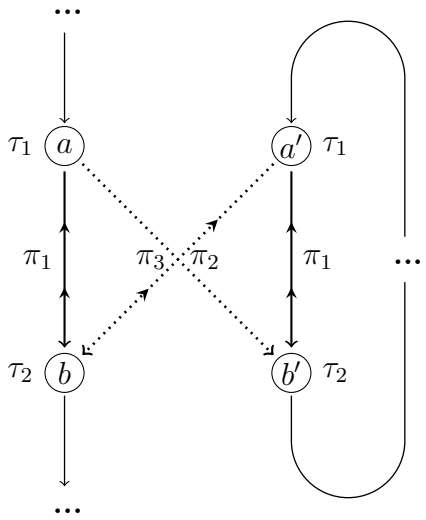


Figure A.9: Case 5 before rewiring

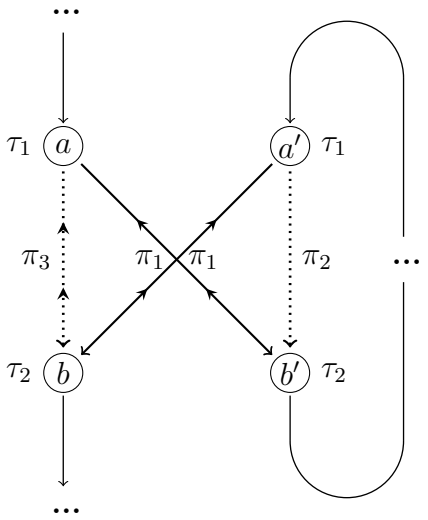


Figure A.10: Case 5 after rewiring

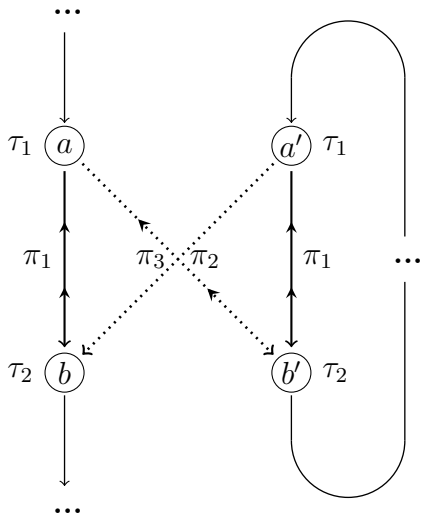


Figure A.11: Case 6 before rewiring

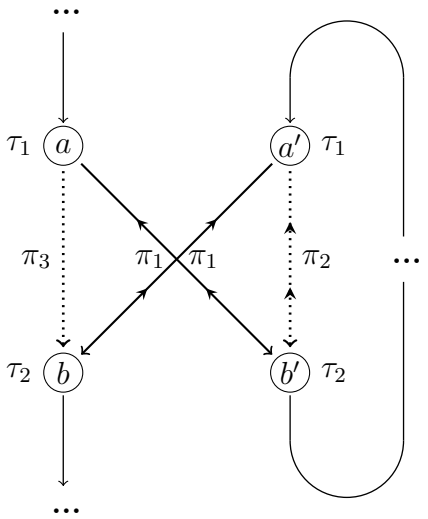


Figure A.12: Case 6 after rewiring

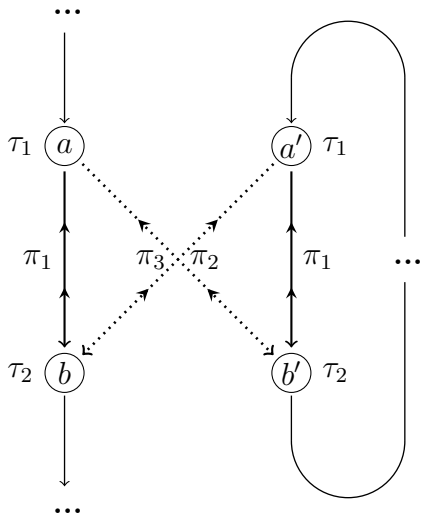


Figure A.13: Case 7 before rewiring

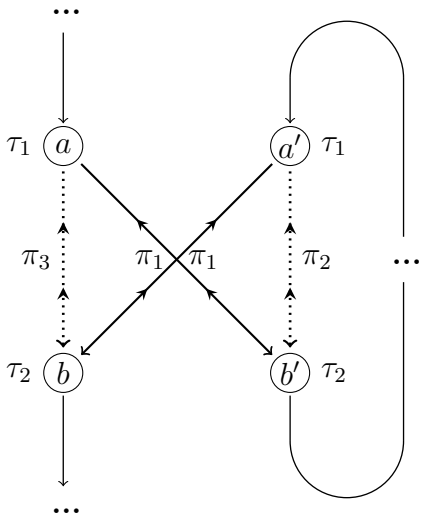


Figure A.14: Case 7 after rewiring

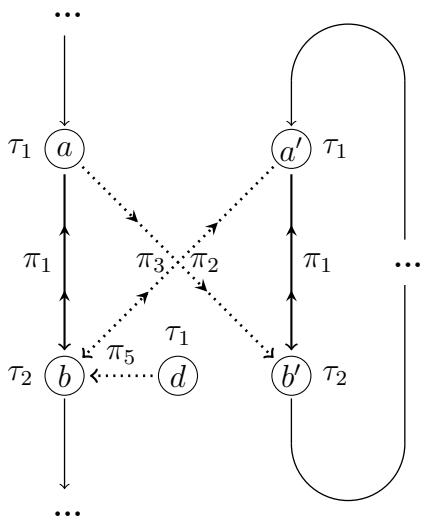


Figure A.15: Case 8 before rewiring

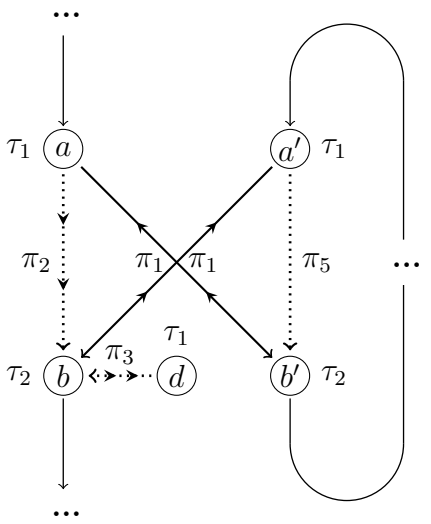


Figure A.16: Case 8 after rewiring

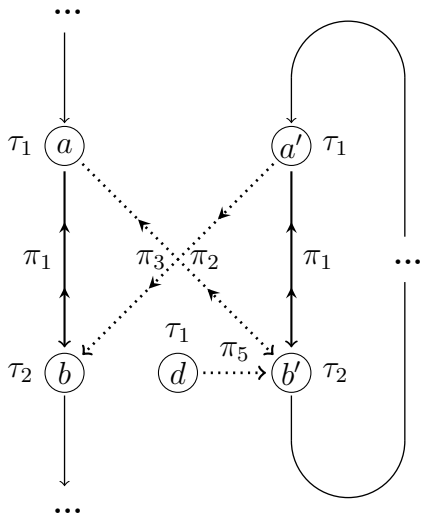


Figure A.17: Case 9 before rewiring

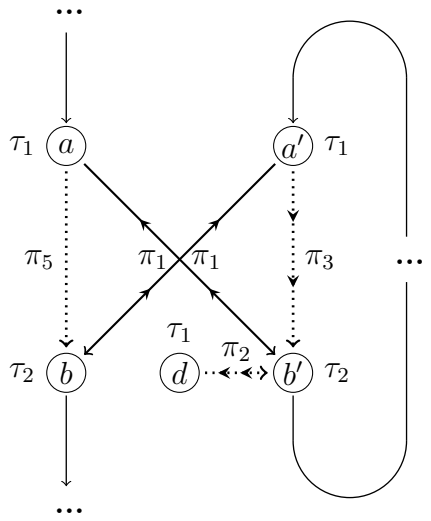


Figure A.18: Case 9 after rewiring

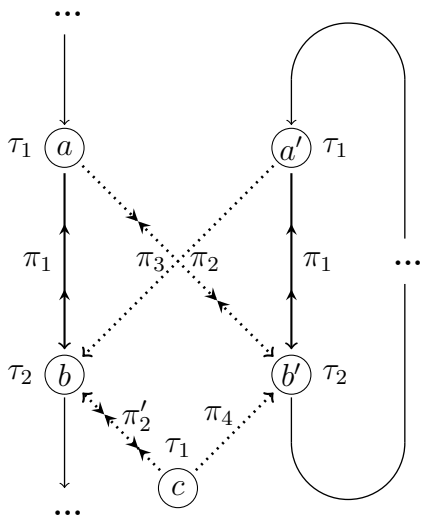


Figure A.19: Case 10 (a) before rewiring

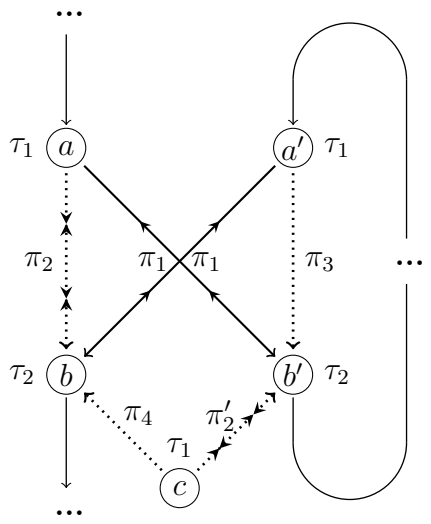


Figure A.20: Case 10 (a) after rewiring

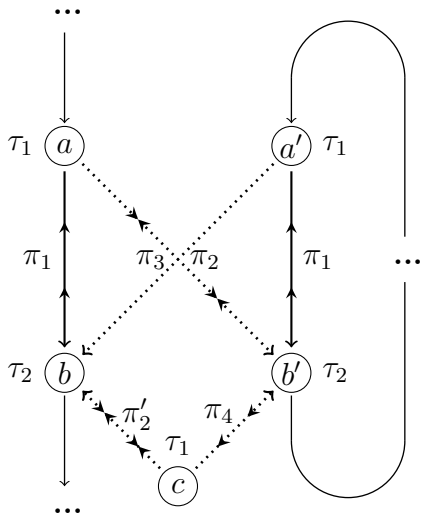


Figure A.21: Case 10 (b) before rewiring

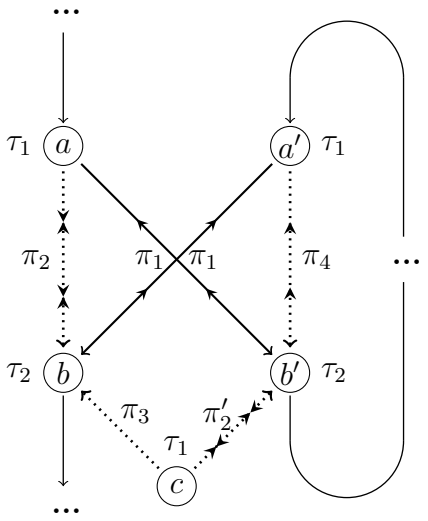


Figure A.22: Case 10 (b) after rewiring

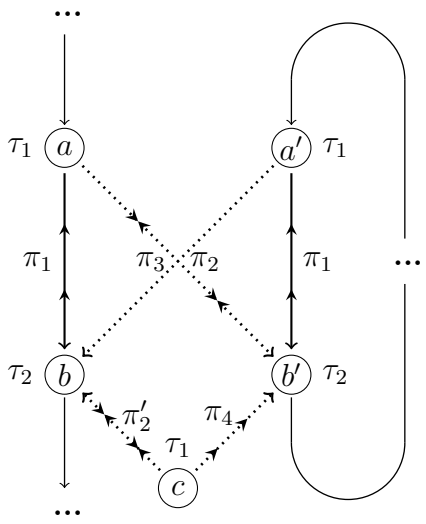


Figure A.23: Case 10 (c) before rewiring

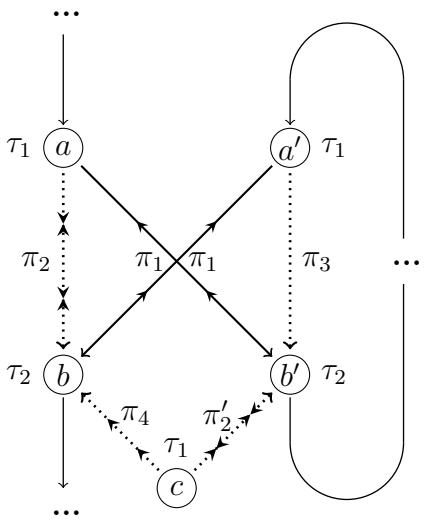


Figure A.24: Case 10 (c) after rewiring

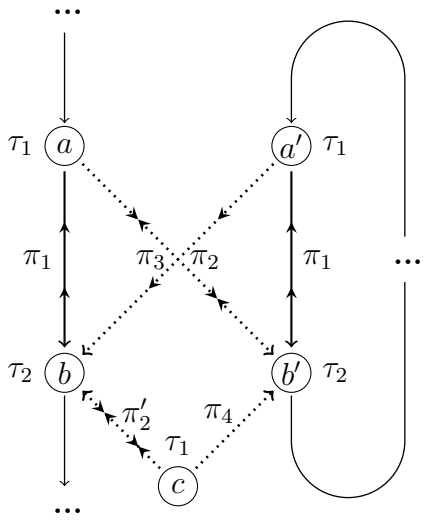


Figure A.25: Case 11 (a) before rewiring

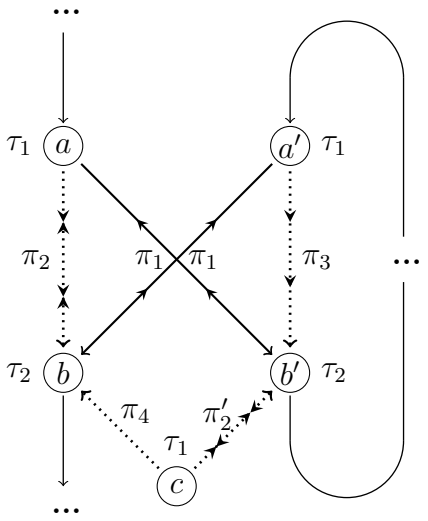


Figure A.26: Case 11 (a) after rewiring

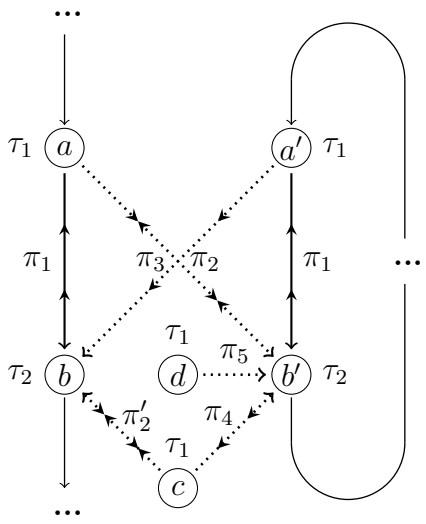


Figure A.27: Case 11 (b) before rewiring

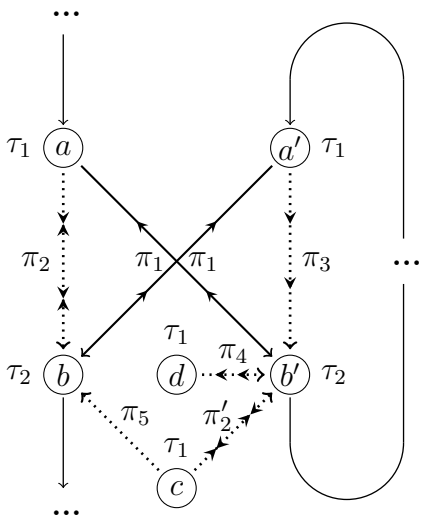


Figure A.28: Case 11 (b) after rewiring

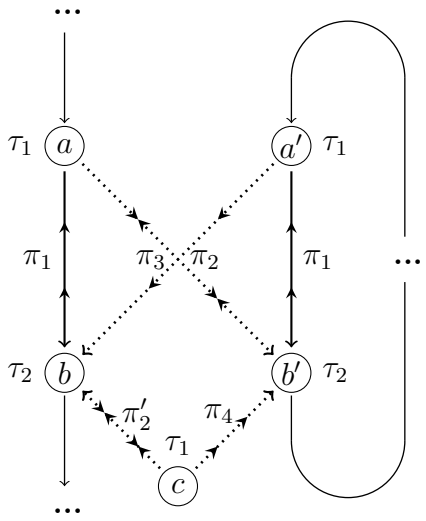


Figure A.29: Case 11 (c) before rewiring

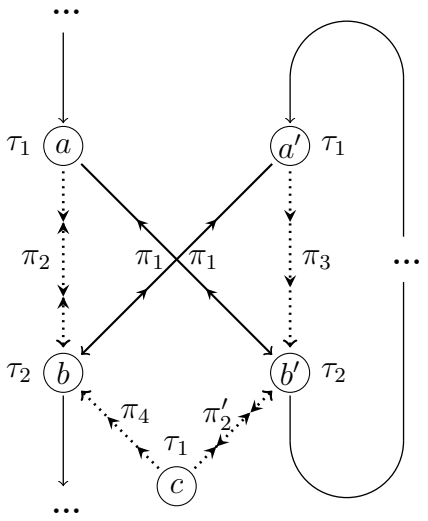


Figure A.30: Case 11 (c) after rewiring

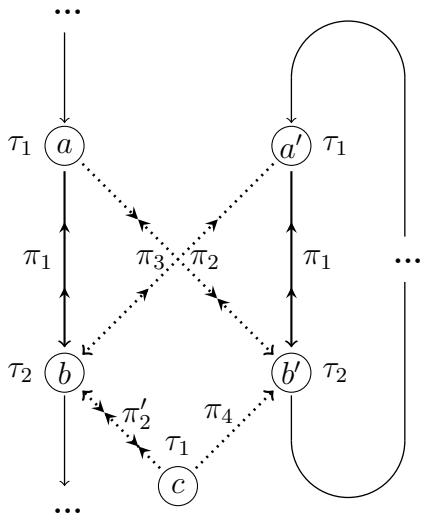


Figure A.31: Case 12 (a) before rewiring

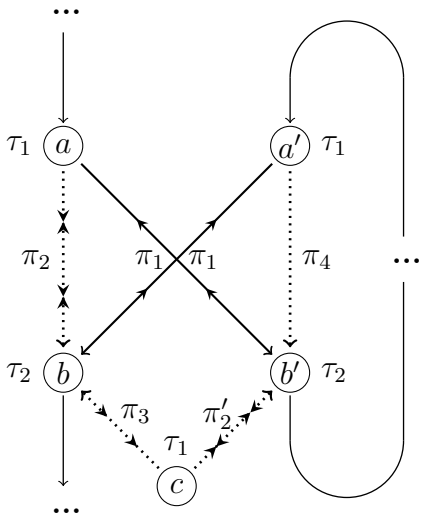


Figure A.32: Case 12 (a) after rewiring

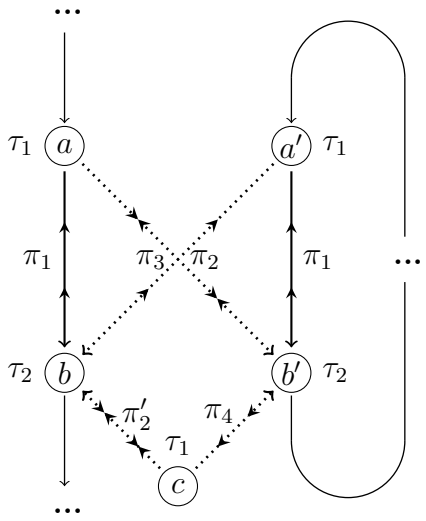


Figure A.33: Case 12 (b) before rewiring

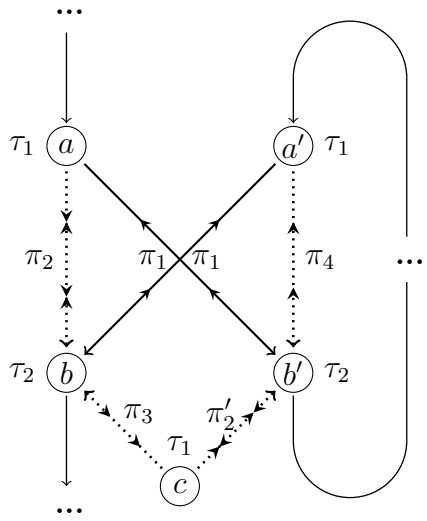


Figure A.34: Case 12 (b) after rewiring

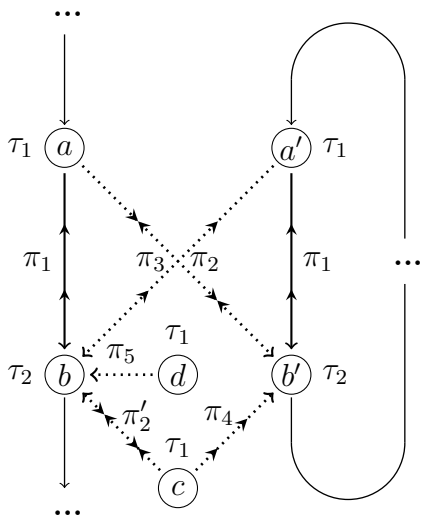


Figure A.35: Case 12 (c) before rewiring

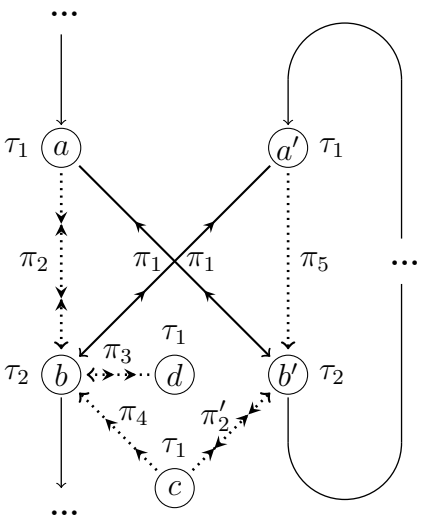


Figure A.36: Case 12 (c) after rewiring

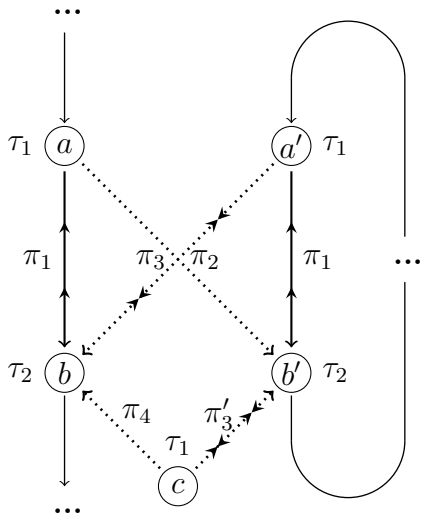


Figure A.37: Case 13 (a) before rewiring

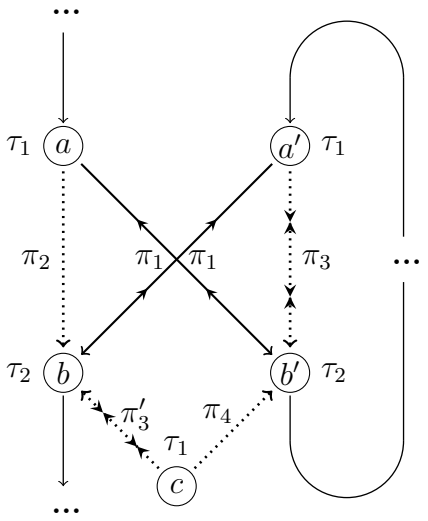


Figure A.38: Case 13 (a) after rewiring

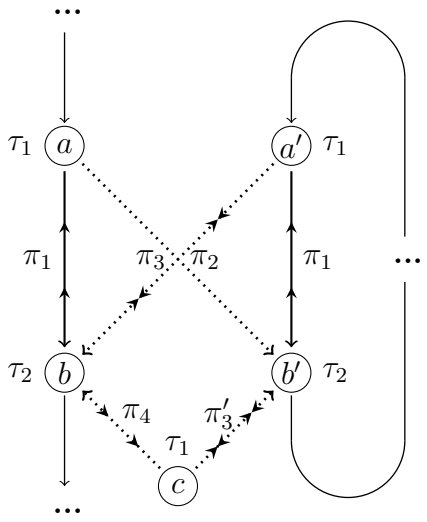


Figure A.39: Case 13 (b) before rewiring

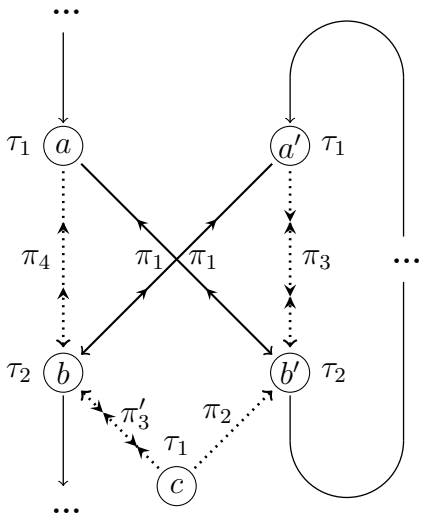


Figure A.40: Case 13 (b) after rewiring

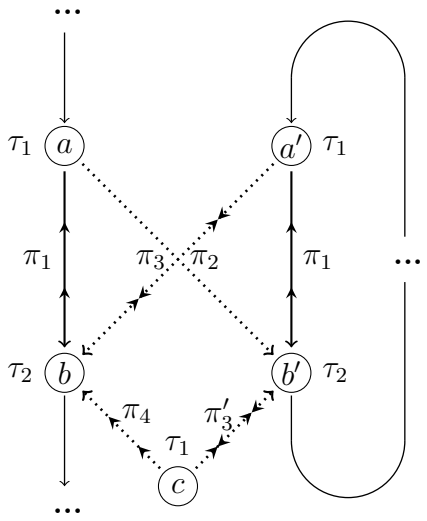


Figure A.41: Case 13 (c) before rewiring

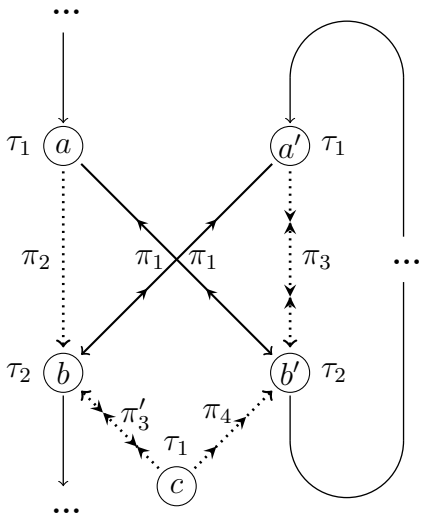


Figure A.42: Case 13 (c) after rewiring

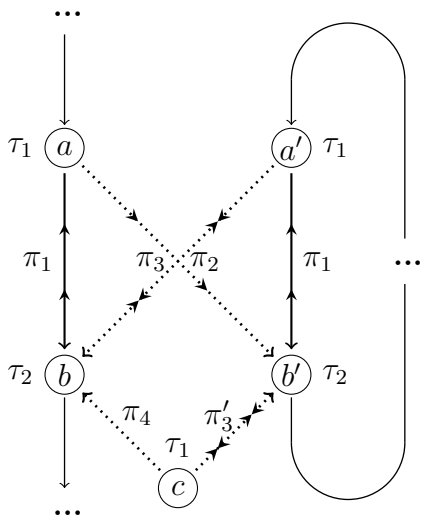


Figure A.43: Case 14 (a) before rewiring

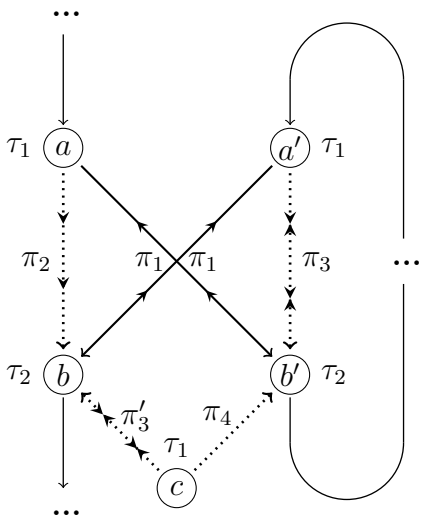


Figure A.44: Case 14 (a) after rewiring

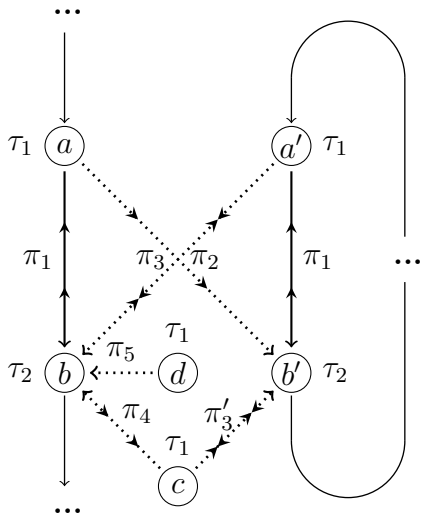


Figure A.45: Case 14 (b) before rewiring

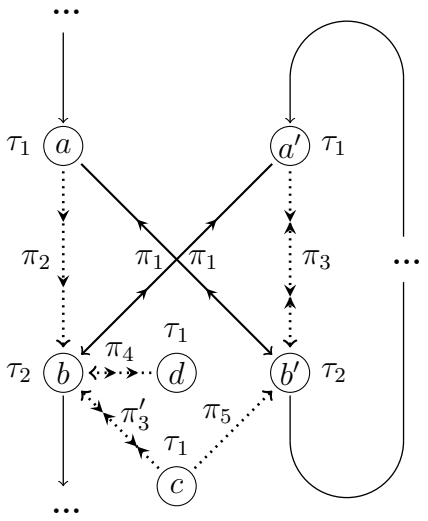


Figure A.46: Case 14 (b) after rewiring

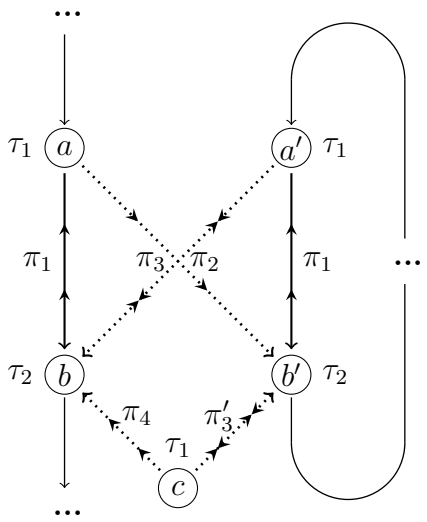


Figure A.47: Case 14 (c) before rewiring

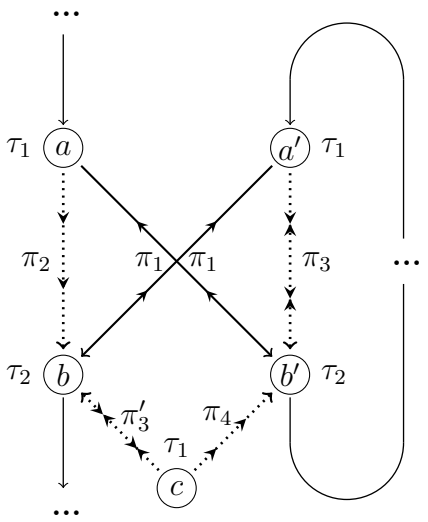


Figure A.48: Case 14 (c) after rewiring

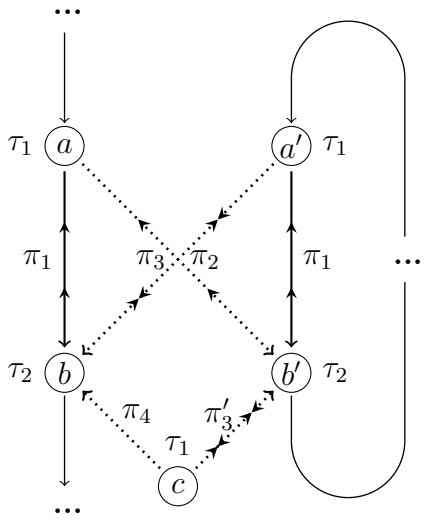


Figure A.49: Case 15 (a) before rewiring

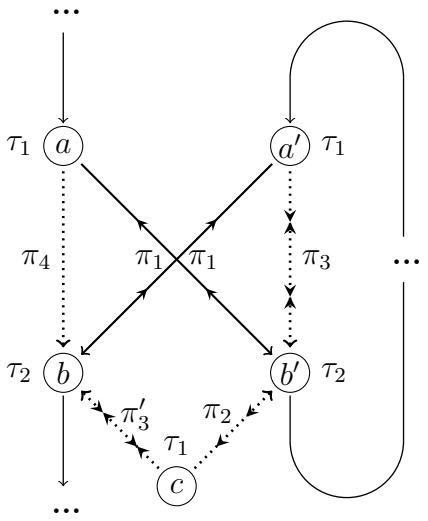


Figure A.50: Case 15 (a) after rewiring

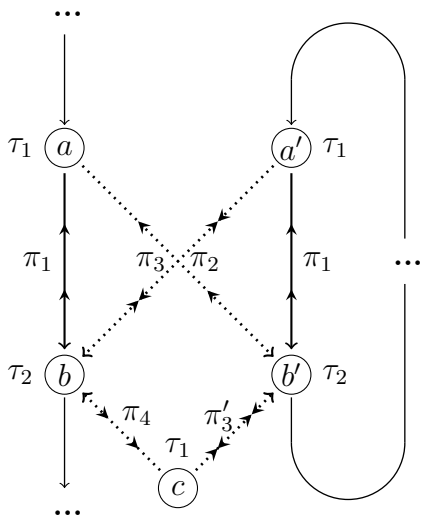


Figure A.51: Case 15 (b) rewiring

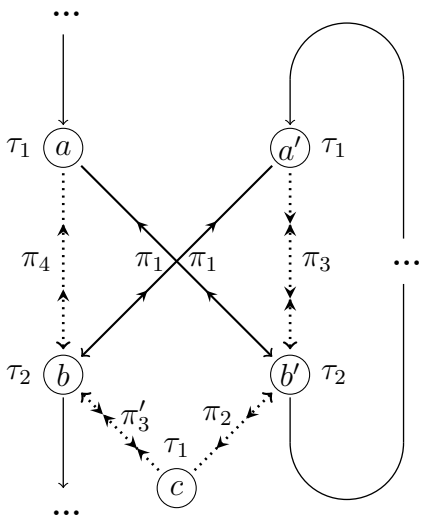


Figure A.52: Case 15 (b) after rewiring

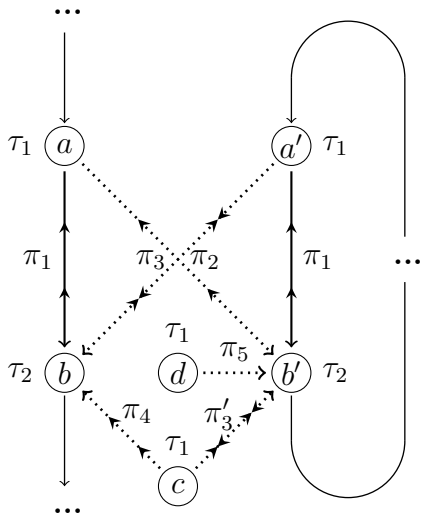


Figure A.53: Case 15 (c) before rewiring

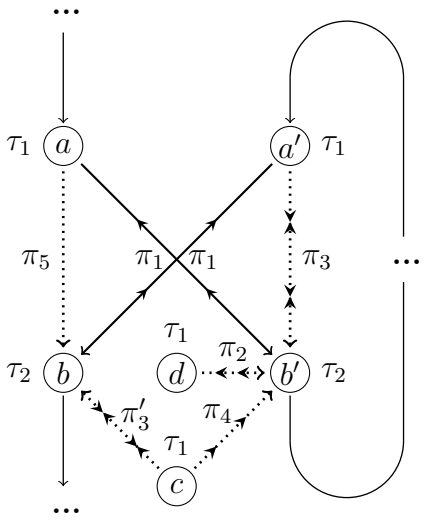


Figure A.54: Case 15 (c) after rewiring

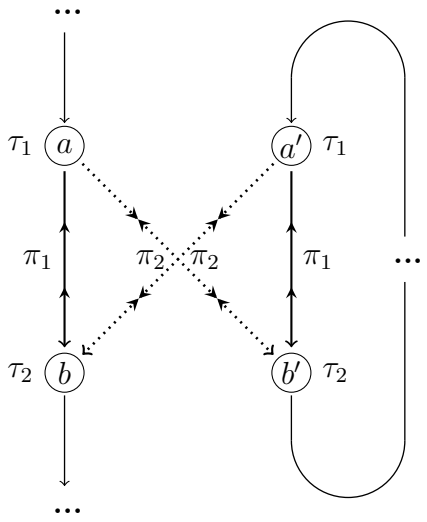


Figure A.55: Case 16 before rewiring

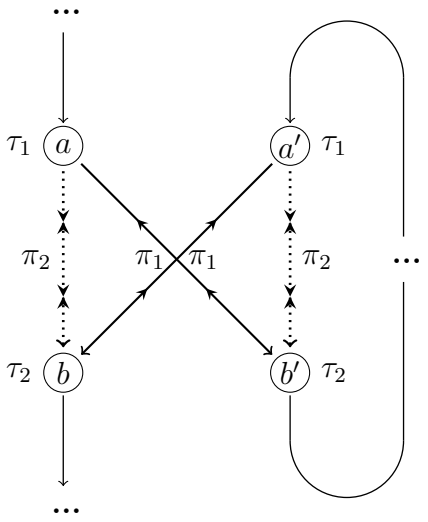


Figure A.56: Case 16 after rewiring

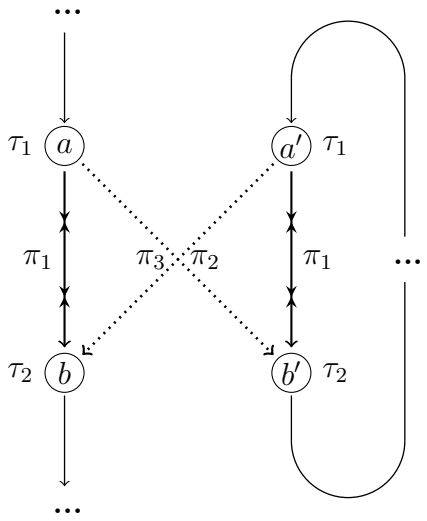


Figure A.57: Case 17 before rewiring

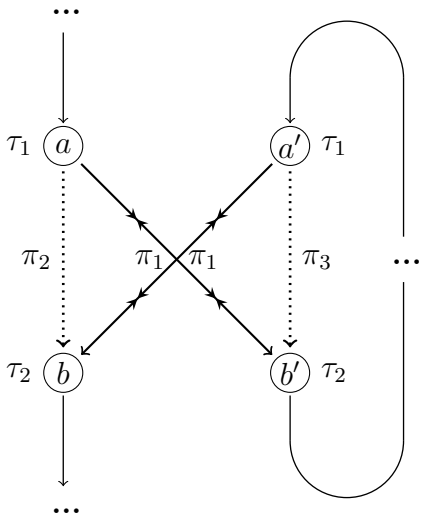


Figure A.58: Case 17 after rewiring

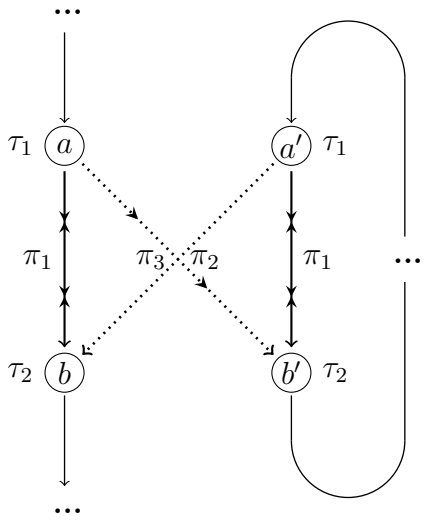


Figure A.59: Case 18 before rewiring

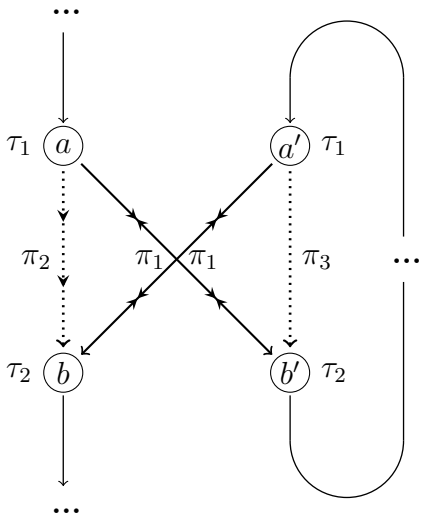


Figure A.60: Case 18 after rewiring

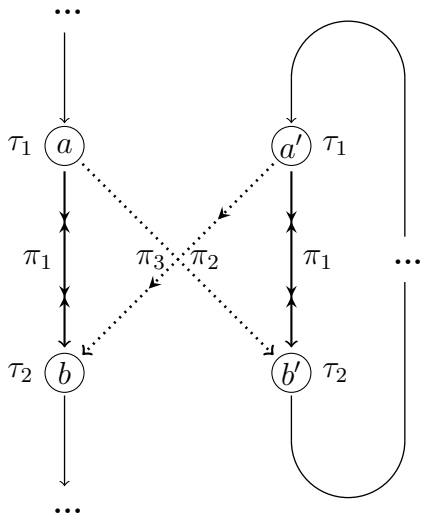


Figure A.61: Case 19 before rewiring

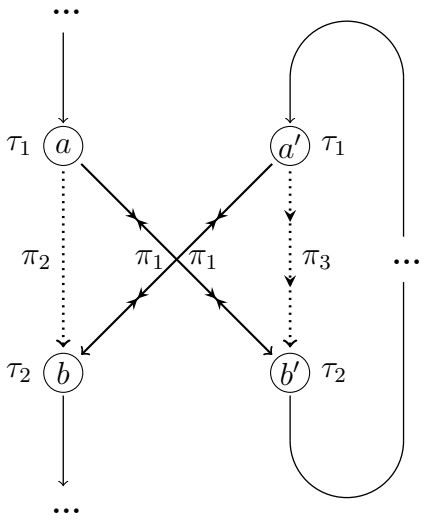


Figure A.62: Case 19 after rewiring

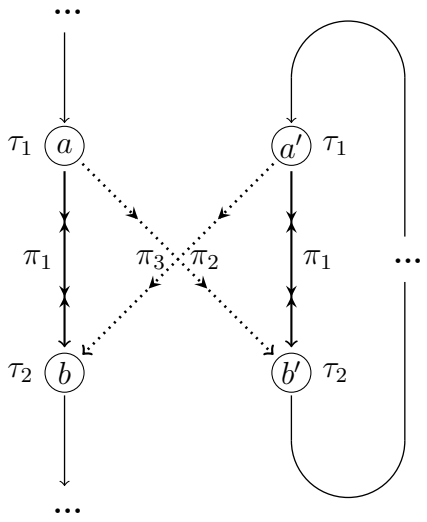


Figure A.63: Case 20 before rewiring

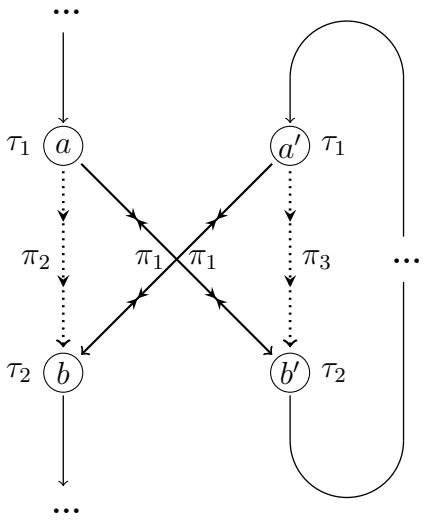


Figure A.64: Case 20 after rewiring

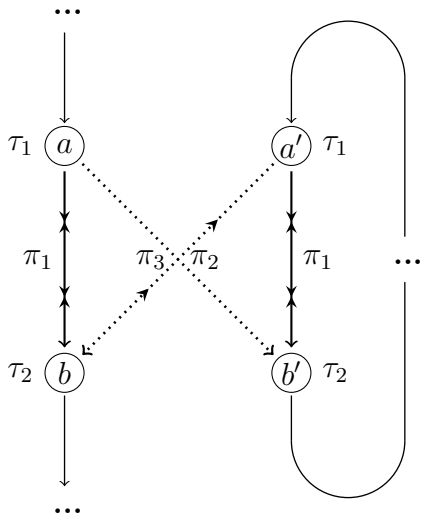


Figure A.65: Case 21 before rewiring

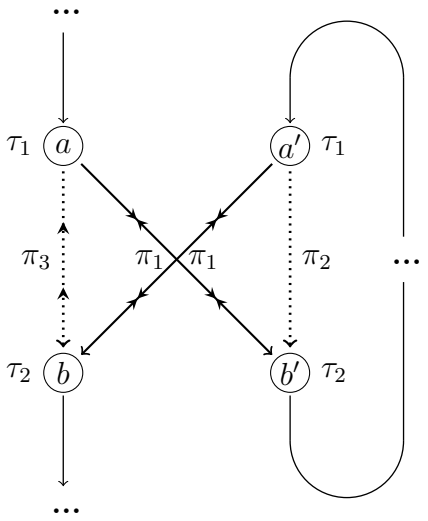


Figure A.66: Case 21 after rewiring

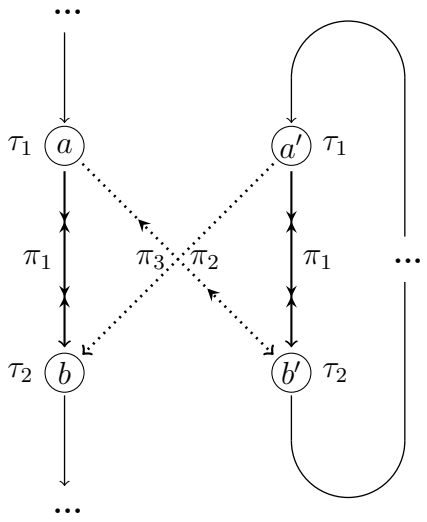


Figure A.67: Case 22 before rewiring

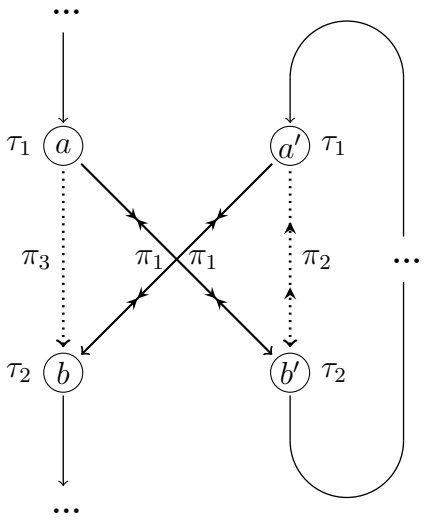


Figure A.68: Case 22 after rewiring

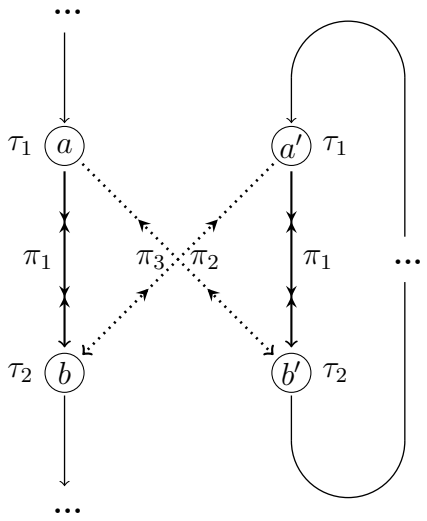


Figure A.69: Case 23 before rewiring

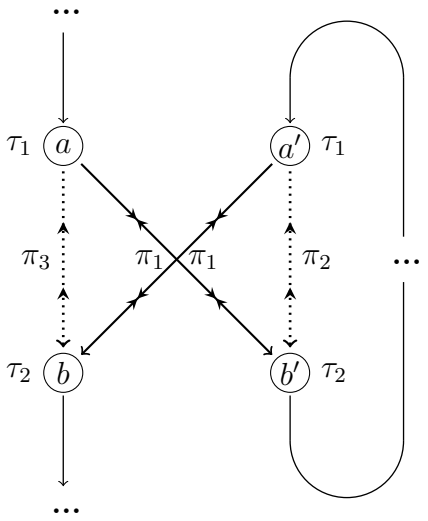


Figure A.70: Case 23 after rewiring

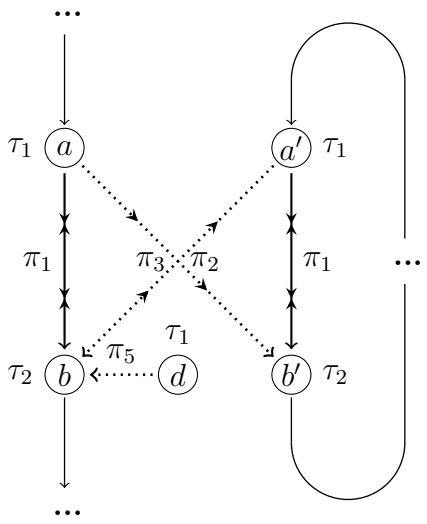


Figure A.71: Case 24 before rewiring

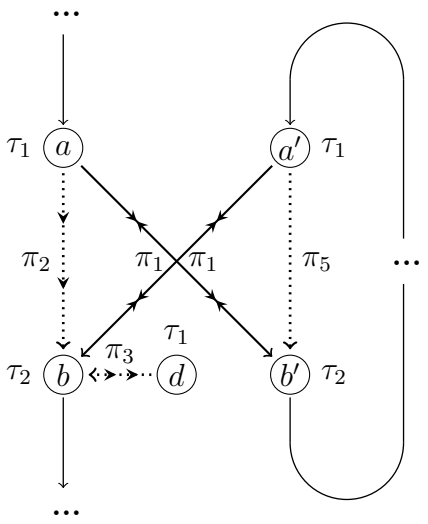


Figure A.72: Case 24 after rewiring

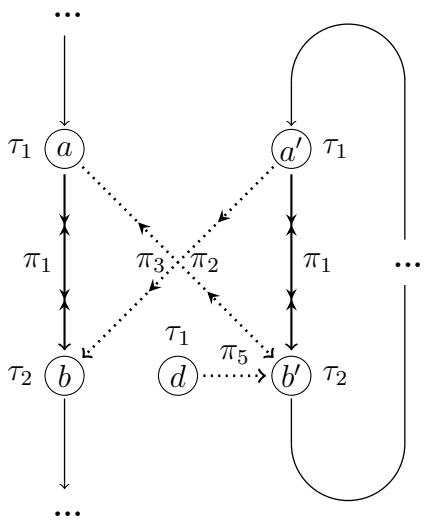


Figure A.73: Case 25 before rewiring

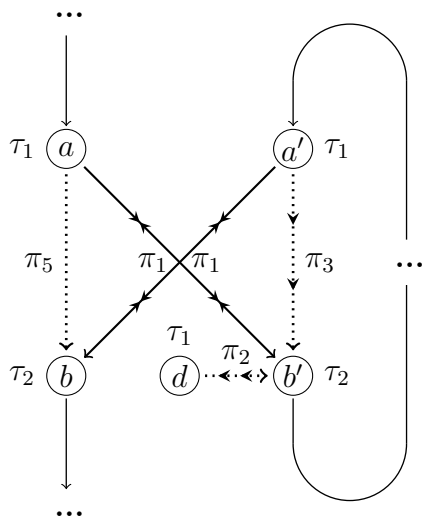


Figure A.74: Case 25 after rewiring