# Applications of algorithmic meta theorems to knowledge representation and reasoning

Diplomarbeit
von
Irena Schindler

Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

Institut für Theoretische Informatik

## Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe.

Irena Schindler

# Contents

# Abstract

In this thesis we will investigate circumscription logic, which was firstly introduced in [McC80]. We investigate the logic from a new point of view. Circumscription is a logic that is used to formalize common sense assumptions. The basic idea in that is to falsify everything – e.g. in a propositional formula – that is possible to falsify. From this, one builds a knowledge base that cirumscibes the investigated problem. The new approach of this thesis is to show that the reasoning in this logic is fixed parameter tractable (FPT). For this reason we apply Courcelle's theorem on it. In our case Courcelle's theorem is basically used as a tool in order to show that if a Circumscription is MSO definable it is fixed parameter tractable. We will recapitulate the property of FPT problems. In order to show that circumscription is MSO definable, we introduce new variables to express the knowledge base, the formula, and the input variables.

# Acknowledgements

Sincere thanks to Professor Vollmer for the support and supervision of this diploma thesis. Then I would like to thank all staff of the institute for theoretical computer science, especially Arne Meier for a excellent mentoring and incitementes. Furthermore I thank warmly Johannes Ebbing for the motivation, support and proofreading. Also I thank both my parents and my parents in-law for their support in taking care of my daughter Nicole, without finish my studies would be almost unfeasible. Finally goes my special thanks to my husband Andreas Schindler for the motivation in the years of study, for his support and in particular for his love.

*Keine Schuld ist dringender, als die, Dank zu sagen.*

Marcus Tullius Cicero

# CHAPTER 1

# Introduction

## 1.1 Circumscription

Circumscription is a non-monotone logic and it was firstly introduced by McCarthy in [McC80]. Initially this logic was invented as a first order variant and later formalized in other variants. In this thesis we consider a propositional variant of circumscription. It is used to model the idea of general assumptions, i.e., when making a statement everything that is not stated explicitly is not true. Originally McCarthy used this logic to formalize, e.g., the missionaries dilemma. It is an example, where on the two sides of a river are three missionaries and three cannibals, respectively. There is a boat that can carry at most two persons and in case the missionaries are outnumbered in any place, they will be killed by the cannibals. The goal here is to make them change the riverside without getting one of the missionaries killed. The circumscription is a way to fix the rules: everything that is not explicitly stated is not true, which is called the closed world assumption. One might get an idea to cross the river over the bridge but it is not the solution to the problem, because of the closed world assumption, since the bridge is not explicitly stated in the rules it does not exist.

In order to illustrate how circumscription works we show one further example. Consider you have three stackable glasses and four coins. The task is to put the coins into the glasses such that in every glass there are at least two coins. One would firstly think that this is impossible. But here is where circumscription comes in play. While explaining those rules to another person, one might get questions considering solutions

using more than four coins because of the assumed impossibility. But in the sense of Circumscription using, e.g., additional coins this is not a solution and in fact this is not allowed in the game. Note, that circumscription does not give a solution to the game, but formalizes the exclusiveness of the allowed rules in the game. Here the solution to the game is rather putting one coin into the first glass. Then stacking the second glass into the first and putting another coin into glass number two. The second coin now is inside glass number two and at the same time in glass number one. At least stacking the last glass into the other two and put the remaining two coins inside.

This example shows that the basic idea behind circumscribing a propositional formula is to set every literal false that does not have to be set to true necessarily. From this we gain a set of literals that are set true. That means that circumscribing models can be expressed by the sets of propositional literals.

For example let us consider the formula

$$\varphi = (x_1 \rightarrow x_2) \wedge (x_3 \vee x_4).$$

One set of variables that can be set to true in order to satisfy $\varphi$ is $M_1 = \{x_1, x_2, x_3, x_4\}$. For Circumscription we try to find the minimal ones. $M_2 = \{x_1, x_2, x_3\}$ and $M_3 = \{x_1, x_2, x_4\}$ are subsets of $M_1$ but the sets $M_4 = \{x_3\}$ and $M_5 = \{x_4\}$ both are minimal sets. From this we can construct a formula $\psi = \neg(x_3 \leftrightarrow x_4)$ that is satisfied by all models that also models $\varphi$, thus the formula $\psi$ is a circumscribing formula for $\varphi$.

In nowadays Circumscription logic has become one of the most developed and well studied formalisms for non-monotone reasoning.

## 1.2 Application of Courcelle's Theorem onto Circumscription

In order to show the property of fixed parameter tractability, we use Courcelle's theorem that has initially been introduced in [Cou90]. Courcelle's theorems based upon Bodleander's theorem. A more detailed insight and a historical outline is given in Chapter 3.2.2.1.

The theorem states that when given a graph $G$ and $\varphi$ is an MSO formula (that means a formula that describes graph properties while quantifying over nodes, not over edges) then checking whether the condition expressed in $\varphi$ holds or not is fixed parameter tractable (FPT), whereas the parameter is the tree width.

In this thesis we investigate circumscription logic, thus in order to apply Courcelle's theorem on it, we show in Chapter 4 how Circumscription logic is encoded in order to

apply Courcelle's theorem on it.

The thesis is organized as follows. In Chapter 2 we will introduce all the preliminaries needed in logic that are used in this thesis. We also give a short overview on tree decomposition and further give some examples to illustrate logical preliminaries, FPT, and Circumscription. In Chapter 3 we give the basic notion of parameterized complexity. Chapter 3.2.2.1 the Courcelle, Elberfeld et. al. theorem is introduced. Finally, Chapter 4 contents our main result, namely the application of Courcelle's theorem in order to show that the reasoning problem of Circumscription is fixed parameter tractable.

# CHAPTER 2

# Preliminaries in Logic

## 2.1 Propositional Logic

Before we get to the main scope of the diploma thesis we need to learn, respectively, repeat some basics.
First we start to define the term "language" with respect to logic, therefore it is required to know about the alphabet $\Sigma$. [Bey09]

**Definition 2.1** (Alphabet, word, and language)**.** A *alphabet* $\Sigma$ is a set of symbols. A finite string of the symbols $s_1, \ldots, s_k \in \Sigma$ is called a *word*, where $k \in \mathbb{N}$. The *language* $\mathcal{L}$ is a subset of all the words over $\Sigma$, called $\Sigma^*$, i.e., $\mathcal{L} \subseteq \Sigma^*$.

**Example 2.2.** Let $\Sigma = \{0, \ldots, 9, =, +, -, \cdot, /\}$, so now we able to describe the language of arithmetic $\mathcal{L}_{arithm}$, with the words like $3 + 4 = 7$. But it should be noted that the words like $1 = 7$ and $+- = 12/7$ are still elements of $\Sigma$ but not of $\mathcal{L}_{arithm}$. Therefore it holds $\mathcal{L}_{arithm} \subset \Sigma^*$.

Now we are going to define $\Sigma$ for propositional logic.

**Definition 2.3** (Alphabet for $\mathcal{PL}$)**.** The *alphabet for the propositional logic* $\Sigma_{\mathcal{PL}}$ consists of

(i) set of propositional variables $Var = \{x_1, x_2, x_3, \ldots\}$,

(ii) set of constants $\{0, 1\}$,

(iii) set of connectives $\{\vee, \wedge, \neg, \rightarrow, \leftrightarrow\}$ and

(iv) brace symbols (,),

i.e., $\Sigma_{\mathcal{PL}} = \Big(Var, \{0,1\}, \{\vee, \wedge, \neg, \rightarrow, \leftrightarrow\}, (,)\Big)$.

In the propositional logic holds the principal of bivalence. That means we have only two truth values namely $\{true,\ false\}$ or rather $\{0,1\}$. This represents an idealized wold-view, but offers a good starting basis. Propositional logic belongs to monotone logic, i.e., a proved valid argument remains valid, even if we add additional premise. But before we consider nonmonotonic logic, we need to learn the semantics of the propositional logic first.

**Definition 2.4** (Set of propositional formulae)**.** The set of propositional formulae over $\Sigma_{\mathcal{PL}}$ we will denote as $\mathcal{PL}$, it holds $\mathcal{PL} \subseteq \Sigma^*_{\mathcal{PL}}$ and is constructed recursively as follows

(i) each $x \in Var$ and the constants $\{0,1\}$ are elements in $\mathcal{PL}$,

(ii) are $\theta, \phi \in \mathcal{PL}$ so holds

- $\neg\phi \in \mathcal{PL}$,
- $(\theta \wedge \phi) \in \mathcal{PL}$,
- $(\theta \vee \phi) \in \mathcal{PL}$,
- $(\theta \rightarrow \phi) \in \mathcal{PL}$,
- $(\theta \leftrightarrow \phi) \in \mathcal{PL}$

(iii) the following priority order (from strong to weak) holds

(1) $\neg$, then

(2) $\wedge$, then

(3) $\vee$, then

(4) $\rightarrow$ and $\leftrightarrow$.

Heretofore we are able to express formulae $\phi \in \mathcal{PL}$. Often it is required to investigate only some parts of the formula, which is defined as follows.

**Definition 2.5** (Subset of propositional formulae)**.** The set $sub(\phi)$ of all parts of the propositional formula $\phi$ is recursively defined as follows

(i) $sub(0) = \{0\}$ and $sub(1) = \{1\}$,

(ii) for $x \in Var$ holds $sub(x) = \{x\}$,

(iii) for $\phi \in \mathcal{PL}$ holds $sub(\neg\phi) = \{\neg\phi\} \cup sub(\phi)$,

(iv) for $\theta, \phi \in \mathcal{PL}$ and $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ holds

$$sub(\theta \circ \phi) = \{\theta \circ \phi\} \cup sub(\theta) \cup sub(\phi).$$

So we learned the syntax of the propositional logic, now we need to deal with the semantics, thus the interpretation of the symbols, respectively the syntax.

**Definition 2.6** (Assignment)**.** A propositional *assignment* $\alpha$ is a mapping

$$\alpha\colon Var \rightarrow \{0, 1\}.$$

A assignment $\alpha$ is extendable to a mapping

$$\alpha' : \mathcal{PL} \rightarrow \{0, 1\}$$

as follows

(i) $\alpha'(x) = \alpha(x)$ for $x \in Var$.

(ii) $\alpha'(0) = 0$ and $\alpha'(1) = 1$.

(iii) $\alpha'(\neg\phi) = 1 - \alpha'(\phi)$ for $\phi \in \mathcal{PL}$.

(iv) $\alpha'(\phi \wedge \theta) = \alpha'(\phi)\alpha'(\theta)$ for $\phi, \theta \in \mathcal{PL}$.

(v) $\alpha'(\phi \vee \theta) = \max\{\alpha'(\phi), \alpha'(\theta)\}$ for $\phi, \theta \in \mathcal{PL}$.

(vi) are $\phi, \theta \in \mathcal{PL}$ and the connector $\circ \in \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ so we get the following truth-table for $\alpha'(\phi \circ \theta)$ :

| $\alpha'(\phi)$ | $\alpha'(\theta)$ | $\alpha'(\neg\phi)$ | $\alpha'(\phi \wedge \theta)$ | $\alpha'(\phi \vee \theta)$ | $\alpha'(\phi \rightarrow \theta)$ | $\alpha'(\phi \leftrightarrow \theta)$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |

**Definition 2.7** (Assignment for a formula)**.** We call $\alpha$ *assignment for a formula* $\phi$, if $\alpha$ is a mapping

$$\alpha\colon Var(\phi) \rightarrow \{0, 1\},$$

where $\alpha$ is expandable to a common assignment, by defining $\alpha$ for $Var \backslash Var(\phi)$ arbitrarily.

**Definition 2.8** (Satisfying assignment)**.** For the formula $\phi$ is $\alpha$ a *satisfying assignment*, if it holds that $\alpha^{'} = 1$, so we will write

$$\alpha \models \phi,$$

and call $\alpha$ a *model* for the formula $\phi$. Is $\Phi$ a set of formulae, we will also write

$$\alpha \models \Phi,$$

if $\alpha \models \phi, \forall \phi \in \Phi$ holds.

Monadic Second Order (MSO) Logic plays a central role in this diploma thesis. Therefore it is required to specify the syntax for First Order Logic as well as for Second Order Logic.

## 2.2 First Order Logic

**Definition 2.9** (Alphabet of First Order Logic)**.** The *alphabet for first order logic* $\Sigma_{\mathcal{FO}}$ consists of

(i) set of variables $Var_{\mathcal{FO}} = \{x_1, x_2, x_3, \dots\}$,

(ii) set of constants $C_{\mathcal{FO}} = \{c_1, c_2, c_3, \dots\}$,

(iii) set of functions $F_{\mathcal{FO}} = \{f_1^{a_1}, f_2^{a_2}, f_3^{a_3}, \dots\}$, where $a_i$ is the arty of the function $f_i$,

(iv) set of relations (predicates) $\mathcal{R}_{\mathcal{FO}} = \{R_1^{a_1}, R_2^{a_2}, R_3^{a_3}, \dots\}$, where $a_i$ is the arty of the function $R_i$,

(v) set of connectives $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$,

(vi) set of quantifiers $\{\exists, \forall\}$,

(vii) set of punctuation $\{(,)\}$,

i.e.,

$$\Sigma_{\mathcal{FO}} = \Big(Var_{\mathcal{FO}}, C_{\mathcal{FO}}, F_{\mathcal{FO}}, \mathcal{R}_{\mathcal{FO}}, \{\neg, \vee, \wedge, \rightarrow\}, \{\exists, \forall\}, \{(,)\}\Big).$$

**Definition 2.10** (Term)**.** We define the *set of Terms* $\mathcal{T}_{\mathcal{FO}}$ inductively as follows

(i) $C_{\mathcal{FO}} \subseteq \mathcal{T}_{\mathcal{FO}}$,

(ii) $Var_{\mathcal{FO}} \subseteq \mathcal{T}_{\mathcal{FO}}$, and

(iii) if $f \in F_{\mathcal{FO}}$ and $t_1, \ldots, t_n \in \mathcal{T}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{FO}}$, where $n$ is the arity of $f$, $n = \text{arity}(f)$.

No other strings are terms.

Hence we will follow the notation of Immermann [Imm99].

**Definition 2.11** (Syntax of $\mathcal{FO}$)**.** We define the set of formulae of first-order logic $\mathcal{FO}$ inductively as follows.

(i) $R^n \in \mathcal{R}_{\mathcal{FO}}$ and $t_1, \ldots, t_n \in \mathcal{T}$, then $R(t_1, \ldots, t_n) \in \mathcal{FO}$,

(ii) if $\phi \in \mathcal{FO}$, then $(\neg\phi) \in \mathcal{FO}$,

(iii) if $\phi, \theta \in \mathcal{FO}$, then $\phi \circ \theta \in \mathcal{FO}$ for each $\circ \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ and

(iv) if $x \in Var_{\mathcal{FO}}$ and $\phi \in \mathcal{FO}$, then $(\forall\phi(x)) \in \mathcal{FO}$ and $(\exists\phi(x)) \in \mathcal{FO}$.

And no other strings are elements of $\mathcal{FO}$.

So now we are in the position to built some strings of formulae. But in order to construe such strings, we need to have the semantics considering that we required two more definitions.

**Definition 2.12** (Vocabulary)**.** *Vocabulary* is a tuple of relation symbols $R_i$ of arity $a_i$, constant symbols $c_i$, and function symbols $f_i$ of arity $k_i$.

$$\tau = \langle R_1^{a_1}, \ldots, R_k^{a_k}, c_1, \ldots, c_s, f_1^{k_1}, \ldots, f_t^{k_t} \rangle$$

**Definition 2.13** (Structure)**.** A structure with vocabulary $\tau$ is a tuple,

$$\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}}, f_1^{\mathcal{A}}, \ldots, f_t^{\mathcal{A}} \rangle$$

The universe $|\mathcal{A}|$ is a nonempty set. It holds $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$, which means for each relation $R_i$ of arty $a_i$ in vocabulary $\tau$, the structure $\mathcal{A}$ has a relation $R_i^{\mathcal{A}}$ of arty $a_i$ defined on the universe $\mathcal{A}$. Similar it holds for each constant $c_j$ in $\tau$, the structure $\mathcal{A}$ has a specified element of its universe $c_j^{\mathcal{A}} \in |\mathcal{A}|$ and each function $f_i^{\mathcal{A}}\colon\ |\mathcal{A}|^{k_i} \rightarrow |\mathcal{A}|$ is a total, where $f_i \in \tau$.

**Definition 2.14** (Semantics of $\mathcal{FO}$)**.** Let $\mathcal{A}$ be a structure of vocabulary $\tau$. For an arbitrary $\tau$-formula $\phi$ and a relation symbol $R^n \in \mathcal{R}_{\mathcal{FO}}$ we define the satisfaction relation $\models$ as follows

$$\begin{aligned}
&\mathcal{A} \models \top && \Leftrightarrow \text{always},\\
&\mathcal{A} \models \bot && \Leftrightarrow \text{never},\\
&\mathcal{A} \models \exists x \phi(x) && \Leftrightarrow \text{there is a value } a \in |\mathcal{A}| \text{ such that } \mathcal{A} \models \phi(x/a),\\
&\mathcal{A} \models \neg\phi && \Leftrightarrow \mathcal{A} \not\models \phi,\\
&\mathcal{A} \models \phi \wedge \theta && \Leftrightarrow \mathcal{A} \models \phi \text{ and } \mathcal{A} \models \theta,\\
&\mathcal{A} \models R(t_1, \ldots, t_n) && \Leftrightarrow R^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}}) \text{ is true},\\
&\mathcal{A} \models t_k = t_m && \Leftrightarrow t_k^{\mathcal{A}} \text{ equals } t_m^{\mathcal{A}},
\end{aligned}$$

where $k, m, n \in \mathbb{N}$ and $t_1, \ldots, t_n$ are $\tau$-terms.

**Definition 2.15** (Free and Bound Variables)**.** Let $\phi \in \mathcal{FO}$. We define the *set of of free variables* of $\phi$, denoted $Var_{\mathcal{FO},free}(\phi)$, as follows

(i) if $\phi = R(t_1, \ldots, t_{arity(R)})$, then $Var_{\mathcal{FO},free}(\phi) = \{x \mid x \text{ appears in } t_i \text{ for some } 0 < i \leqslant arity(R)\}$,

(ii) if $\phi = (\neg\theta)$, then $Var_{\mathcal{FO},free}(\phi) = Var_{\mathcal{FO},free}(\theta)$,

(iii) if $\phi = (\theta \to \varphi)$, then $Var_{\mathcal{FO},free}(\phi) = Var_{\mathcal{FO},free}(\theta) \cup Var_{\mathcal{FO},free}(\varphi)$ and

(iv) if $\phi = \forall\theta(x)$, then $Var_{\mathcal{FO},free}(\phi) = Var_{\mathcal{FO},free}(\theta) - \{x\}$.

Other variables that occur in $\phi$ are called *bound*.

**Example 2.16.** For $n \in \mathbb{N}$, consider the logical structure

$$\mathcal{A}_n = \langle \{0, 1, \ldots, n-1\}, \textsc{Times}^{\mathcal{A}_n}, 0, 1 \rangle$$

of vocabulary

$$\tau = \langle R^3, 0, 1 \rangle,$$

where Times is the arithmetical relation, that means for $i, j, k < n$ holds,

$$\mathcal{A}_n \models \textsc{Times}(i, j, k) \Leftrightarrow i \cdot j = k.$$

Now we want to construct $\mathcal{FO}$ formulae in $\mathcal{L}_{\mathcal{FO}}(\tau)$ that represent the following arithmetic relations,

- $\textsc{Div}(x, y)$, meaning that $y$ is a multiple of $x$.
- $\textsc{Prime}(x)$, meaning that $x$ is a prime number.

First we start with the division

$$\textsc{Div}_{\mathcal{FO}}(x, y) := \exists z\Big(\textsc{Times}(x, y, z)\Big).$$

And $\textsc{Prime}_{\mathcal{FO}}(x)$ we will define as follows

$$\textsc{Prime}_{\mathcal{FO}}(y) := \forall x\Big(\textsc{Div}_{\mathcal{FO}}(x, y) \to \big(x = 0 \vee x = 1 \vee x = y\big)\Big).$$

All quantified variables are called bound. In the case of $\textsc{Div}_{\mathcal{FO}}(x, y)$ are $x, y$ free i.e., $x, y \in \mathit{Var}_{\mathcal{FO},free}(\textsc{Div})$. The variable $z$ is quantified with an extensional quantor, so $z$ is a bound variable. In the case of $\textsc{Prime}_{\mathcal{FO}}(y)$ $x$ is bound and $y$ is free.

## 2.3 Second Order Logic

Compared to the languages in First Order Logic, we are able to quantify over subsets of the universe $|\mathcal{A}|$ of the structure $\mathcal{A}$. In the following we will define the syntax and the semantics of the Second Order Logic.

**Definition 2.17** (Syntax of $\mathcal{SO}$)**.** We define the set of formulae of Second Order Logic $\mathcal{SO}$ inductively as follows.

(i) $R^n \in \mathcal{R}_{\mathcal{SO}}$ and $t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{SO}}$, then $R(t_1, \ldots, t_n) \in \mathcal{SO}$,

(ii) if $\phi \in \mathcal{SO}$, then $(\neg\phi) \in \mathcal{SO}$,

(iii) if $\phi, \theta \in \mathcal{SO}$, then $\phi \circ \theta \in \mathcal{SO}$ for each $\circ \in \{\vee, \wedge, \to, \leftrightarrow\}$,

(iv) if $x \in \mathit{Var}_{\mathcal{SO}}$ and $\phi \in \mathcal{SO}$, then $(\forall\phi(x)) \in \mathcal{SO}$ and $(\exists\phi(x)) \in \mathcal{SO}$ and

(v) if $X \subseteq \mathit{Var}_{\mathcal{SO}}$ and $\phi \in \mathcal{SO}$, then $\forall X(\phi(X)) \in \mathcal{SO}$ and $\exists X(\phi(x)) \in \mathcal{SO}$.

And no other strings are elements of $\mathcal{SO}$.

**Definition 2.18** (Semantics of $\mathcal{SO}$)**.** Let $\mathcal{A}$ be a structure of vocabulary $\tau$. For an arbitrary $\tau$-formula $\phi$ and a relation symbol $R^n \in \mathcal{R}_{\mathcal{SO}}$ we extend the satisfaction relation $\models$ for $\mathcal{FO}$ as follows

$$\mathcal{A} \models \exists X\phi(X) \Leftrightarrow \text{ there is a } R \subseteq |\mathcal{A}|^n \text{ such that } \mathcal{A} \models \phi(R),\ n \in \mathbb{N}.$$

### 2.3.1 MSO

Monadic Second Order Logic is a restricted form of Second Order Logic. We only allow the quantification of unary relations, where all variables are either first order variables or second order variables (set of variables). A large class of graph problems can be expressed by MSO, such as colorability. For further information about MSO, one can consider any standard book in logic. Here we give an example how the MSO works.

**Example 2.19.** Given a vocabulary $\tau = (V^G, E^G)$ with the set of vertexes $V$ and a set of edges $E$ of the instance graph G. Now we can specify the decision problem 3COLORGRAPH in MSO as follows:

$$
\begin{aligned}
3\text{COLORGRAPH} := & \exists C_1 \exists C_2 \exists C_3 \\
& \forall x \forall y \Big( \big(C_1(x) \vee C_2(x) \vee C_3\big) \\
& \wedge \big(\neg(C_1(x) \wedge C_2(x)) \wedge \neg(C_1(x) \wedge C_3(x)) \\
& \wedge \neg(C_2(x) \wedge C_3(x))\big) \\
& \wedge E(x,y) \rightarrow \neg\big(\neg(C_1(x) \wedge C_1(y)) \\
& \wedge \neg(C_2(x) \wedge C_2(y)) \wedge \neg(C_3(x) \wedge C_3(y))\big)\Big).
\end{aligned}
$$

That means the instance of the decision problem 3COLORGRAPH is positive iff the graph can be colored in three colors $C_1, C_2, C_3$ and each vertex of the graph is necessary colored in one and only one of three colors further the adjacent vertexes $x$ and $y$ do not have the same color.

# CHAPTER 3

# Parametrized Complexity

In this section we introduce the basics of parametrized complexity. The idea of parameterized complexity is to make use of the structure of the input to get some practical tractability. We will restrict a parameter from the input which effects the complexity at most. We will follow the notation of Rod Downey [Dow12] and [FG06] for our investigations.

## 3.1 FPT

Fixed Parameter Tractable problems deal with the issue on NP-complete problems, for which instance the NP-complete problem is still efficient solvable and which parameter constrains the runtime most. For example several graph problems can be solved with small tree width quickly. For further consideration we require the following essential definitions.

**Definition 3.1** (Parameterization)**.** Let $\Sigma$ be a finite alphabet. A *parameterization* of $\Sigma^*$ is a mapping

$$\kappa : \Sigma^* \to \mathbb{N}$$

that is polynomial time computable.

**Definition 3.2** (Parametrized Problem)**.** Let $\Sigma$ be a finite alphabet. A *parameterized problem* over $\Sigma$ is a pair $(Q, \kappa)$ consisting of a set $Q \subseteq \Sigma^*$ of strings over $\Sigma$ and parameterization $\kappa$ of $\Sigma^*$.

**Definition 3.3** (FPT-Algorithm)**.** Let $\Sigma$ be a finite alphabet and $\kappa : \Sigma^* \to \mathbb{N}$ a parameterization. An algorithm $A$ with input alphabet $\Sigma$ is an *fpt-algorithm with respect to* $\kappa$ if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial $p \in \mathbb{N}_0[X]$ such that for every $x \in \Sigma^*$, the running time of $A$ on input $x$ is at most

$$f\big(\kappa(x)\big) \cdot p(|x|)$$

where $|x|$ is the length of a string $x \in \Sigma^*$.

**Definition 3.4** (Fixed-Parameter Tractable)**.** Let $\Sigma$ be a finite alphabet and $\kappa : \Sigma^* \to \mathbb{N}$ a parameterization. A parametrized problem $(Q, \kappa)$ is *fixed-parameter tractable* it there is a fpt-algorithm with respect to $\kappa$ that decides $Q$.
FPT denotes the class of all fixed-parameter tractable problems.

In order to find an FPT-algorithm there exist technical tools like

- Kernelization, and
- Tree Width, and the outcome of this is
- Courcelle's theorem

These technical tools will be discussed in the following sections. The most interesting tool will be the extended Courcelle's theorem in the section of tree width. As the main task of this diploma thesis we use the Courcelle's theorem to show the fixed parameter tractability of reasoning in Circumscription.

## 3.2 Technical Tools for FPT-Algorithms

Given a NP-complete problem, the following technical tools are used to show the fixed parameter tractability of this problem. As an example we consider a known issue VertexCover, which one is known to be NP-complete.
Let $G = (V, E)$ be a graph with the set of vertices $V$ and the set of edges $E$. The subset of vertices $V' \subseteq V$ is called a*vertex cover*, if for each edge $(u, v) \in E$ holds $v \in V'$ or $u \in V'$. So we define the VertexCover problem as follows

$$\text{VertexCover} := \{\langle G, k\rangle \mid G \text{ has a vertex cover} \leqslant k \text{ vertices}\}.$$

We will show that VertexCover $\in$ NP. Therefore we give a verification algorithm. The instance of the verification algorithm is an undirected graph $G = (V, E)$ with the set of

vertices $V$ and $E$ the set of edges and the maximum value $k$ of vertices in the subset $V' \subseteq V$ of vertex cover. In the first step we check whether $V'$ is a subset of $V$ if it is not the case then we reject as the vertex cover can not be involved of the instant graph $G$. Subsequently we test if the subset of vertices exceeds the maximum value $k$. Then we verify every edge of the graph $G$, whether the adjacent vertices $u$ or $v$ are in the subset $V'$.

**Algorithm 1:** VERIFICATION of VERTEXCOVER

```
   Input: G, k, V'
 1 if V' ⊈ V then
 2 |   reject
 3 end
 4 if k < |V'| then
 5 |   reject
 6 end
 7 for (u, v) ∈ E do
 8 |   if not (u ∈ V or v ∈ V') then
 9 |   |   reject
10 |   end
11 end
12 accept
```

The runtime of this verification algorithm is $O(|E| \cdot |V|)$ and thus polynomial. Therefore it holds VERTEXCOVER $\in$ NP.

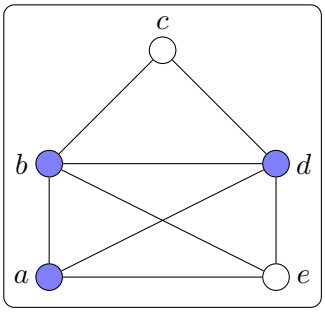



Figure 3.1: Example of vertex cover

In figure 3.6 we see a graph with the vertex cover subset $V' = \{a, b, d\}$. Please note that even the minimal vertex cover is not distinct. We can choose $V' = \{b, d, e\}$ for the vertex cover as well.

Unfortunately not all NP-complete problems are fixed parameter tractable, like the CLIQUE problem, which is known to be W[1]-complete [FG06].

### 3.2.1 Kernelization

Kernelization is a polynomial time many one reduction. It is used to design an efficient algorithm to get fixed parameter tractability. The input is modified in the preprocessing stage by mapping the original input to a smaller parameterized input so called "kernel". Certainly the output of the algorithm should be the same as of the original problem.

**Definition 3.5** (Kernalization)**.** A *kernalization* $K$ of a parameterized problem $(Q, \kappa)$ over the alphabet $\Sigma$ is polynomial if there is a polynomial $p(X) \in \mathbb{N}_0[X]$ such that $|K(x)| \leqslant p\big(\kappa(x)\big)$, $\forall x \in \Sigma^*$.

But not every problem in FPT has a polynomial time kernelization. J. Flum and M. Grohe [FG06] give the following nice example for it.

**Example 3.6.** Let $Q \in \Sigma^*$ be a problem that is EXPTIME-complete and hence not in PTIME and let

$$\kappa : \Sigma^* \to \Sigma^*$$

be defined by

$$\kappa(x) := \lceil \log \log |x| \rceil.$$

Then $(Q, \kappa)$ is fixed parameter tractable, but if $(Q, \kappa)$ had a polynomial kernelization, then $Q$ would be decidable in polynomial time.

Above we discussed the original problem of VERTEX-COVER. In 1993 S. Buss gave the first kernelization of the parameterized p-VERTEX-COVER problem that is known as "Buss' Kernalization".

**Proposition 3.7.** p-VERTEX-COVER has a polynomial kernelization. More precisely, there is a kernalization of p-VERTEX-COVER that, given an instance $(G, k)$ of p-VERTEX-COVER with $G = (V, E)$, computes in time $O(k + \|G\|)$ an instance $(G', k')$ such that

$$k' \leqslant k,$$

and either $(G', k') = (G^-, 1)$ is the trivial "no"-instance where the graph $G^-$ consists of two disjoint edges, or $G' = (V', E')$ such that

$$|V'| \leqslant 2(k')^2 \text{ and } |E'| \leqslant (k')^2.$$

*Proof.* Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. The kernelizetion is based on the following two observations

(i) If $v \in V$ with $\deg(v) > k$, then $v$ is contained in in every vertex cover of $G$ of at most $k$ elements.

(ii) If $\deg(G) \leqslant k$ and $G$ has a vertex cover of $k$ elements, then $G$ has at most $k^2$ edges.

For $v \in V$, let $G - v$ denote the graph obtained from graph $G$ by deleting $v$ and all edges incident to $v$.
Note that if $\deg(v) > k$, then $G$ has a vertex cover of $k$ elements if and only if $G - v$ has a vertex cover of $k - 1$ elements.

**Algorithm 2:** Buss' Kernelization

```
1  Reduce(G, k)
2  if k = 0 then
3  |  if G has no edges then
4  |  |  return (G^+, 1)
5  |  end
6  |  else
7  |  |  return (G^-, 1)
8  |  end
9  end
10 else
11 |  if G has a vertex v with deg(v) > k then
12 |  |  return Reduce(G - v, k - 1)
13 |  end
14 |  else
15 |  |  if G has at most k^2 edges then
16 |  |  |  return (G, k)
17 |  |  end
18 |  |  else
19 |  |  |  return(G^-, 1).
20 |  |  end
21 |  end
22 end
```

Starting from the instance $(G, k)$ of p-Vertex-Cover the two observations made above are ensured in the Buss algorithm at line 11 and 15. So we get with the subfunction Reduce$(G - v, k - 1)$ at line 12 the equivalent instance $(G', k')$ to the original instance $(G, k)$. It is either a graph $G^-$ consisting of two disjoint edges or a graph $G'$ which has

at most $(k^{'})^2$ edges and therefore at most $2 \cdot (k^{'})^2$ connected vertices. In view of the strict cardinality of $k'$ we can remove at most $n^{'} - k^{'}$ isolated vertices from $G^{'}$, where $n^{'}$ is the number of vertices in $G^{'}$.

Now we consider the implementation of REDUCE. for this purpose, the degree of all vertices must be calculated first. In the new created list $L_i$, with $i \in [0, |V| - 1]$ we save all vertices of degree $i$. Here we have a linear running time. After removing one vertex of degree $> k$ the list $L_i$ has to be updated. The runtime here is depend on the degree of the removed vertex, thus $O(\deg(v))$. Every vertex can be deleted only once, so we get the over all running time $O(|E|)$ for the update operation. □

The following corollary shows that the running time of proposition 3.7 can be reduced, if the kernelization is used as a preprocessing algorithm for a bound search tree algorithm.

**Corollary 3.8.** p-VERTEX-COVER can be solved in time $O(n + m + 2^k \cdot k^2)$, where $k$ denotes the parameter, $n$ the number of vertices of the input graph and $m$ the number of edges of the input graph.

In this subsection we learned a well known technique to create a fixed parameter tractable algorithm, the kernelization. We showed how to use the kernelization on example of VERTEXCOVER.

### 3.2.2 Tree Width

Rudolf Halin was the first one who introduced tree width in 1976. Tree width is a parameter to measure the graphs, trees or similar structures. It is known that many NP-hard tree-likeliness of decision and optimization problems are fixed parameter tractable if they are parametrized by the tree width. In 1990 Bruno Courcelle set a milestone in this area. He gives a powerful theorem, which says that every NP-hard MSO definable problem that is parametrized by tree width is fixed parameter tractable. Courcelle's theorem will be discussed in Subsection 3.2.2.1.

We start with three formal definitions of "Tree Decomposition", "Width of a Tree Decomposition" and "Tree Width" and subsequently we outline these definitions with an example.

**Definition 3.9** (Tree Decomposition)**.** A tree decomposition of a graph $G = (V, E)$ is a tree $\mathcal{T}$ together with a collection of subset $T_x$ (called bags) of $V$ labeled by the vertices $x$ of $\mathcal{T}$ such that following holds

(i) $\bigcup_{x \in \mathcal{T}} T_x = V$,

(ii) $\forall(u, v) \in E$ there is some $x$ such that $\{u, v\} \subseteq T_x$, and

(iii) if $y$ is a vertex on the unique path in $\mathcal{T}$ from $x$ to $z$ then $T_x \cap T_z \subseteq T_y$.

**Definition 3.10** (Width of a Tree Decomposition)**.** The width of a tree decomposition is the maximum value of $|T_x| - 1$ taken over all vertices $x$ of the tree $\mathcal{T}$'s decompositions.

**Definition 3.11** (Tree Width)**.** The tree width of a graph $G$ is the minimum tree width of all tree decompositions of G.

Below we consider an example of tree decomposition from a graph $G$ and a tree width of 2.

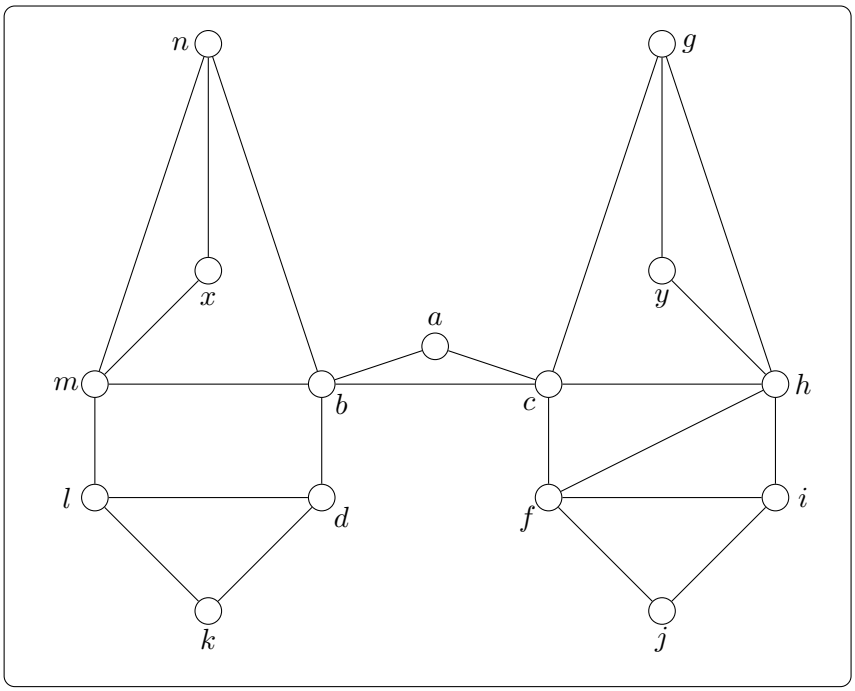



Figure 3.2: Graph $G$

Given a graph of the figure 3.2. We make use of the definition 3.9 with the rules

(i) $\bigcup_{x \in \mathcal{T}} T_x = V$,

(ii) $\forall(u, v) \in E$ there is some $x$ such that $\{u, v\} \subseteq T_x$, and

(iii) if $y$ is a vertex on the unique path in $\mathcal{T}$ from $x$ to $z$ then $T_x \cap T_z \subseteq T_y$,

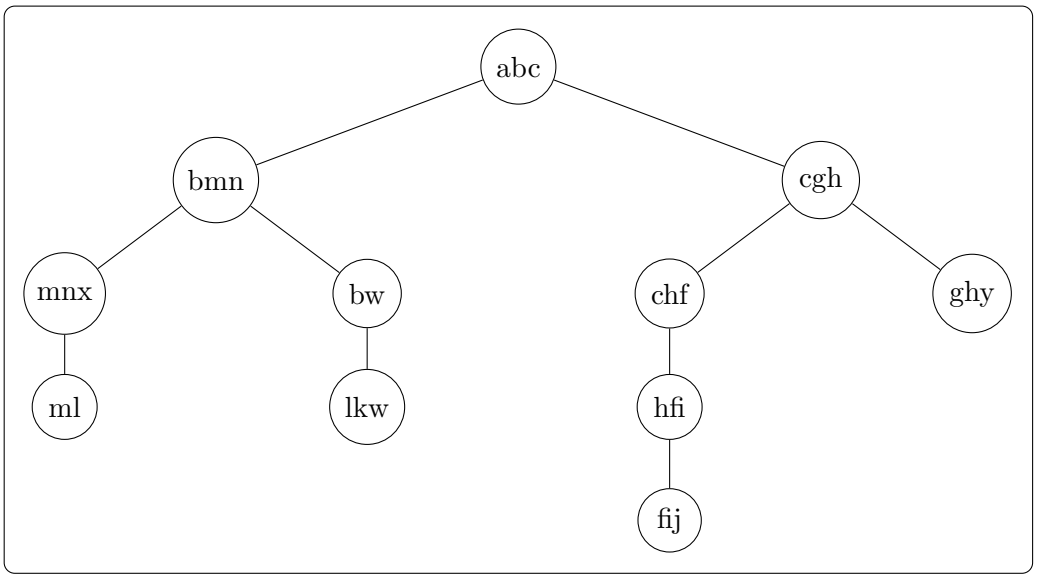



Figure 3.3: Tree decomposition of graph $G$.

so we get following tree of the Figure 3.3.

Note that the negative tree width of $-1$ is also possible, if we have an empty graph. Analogously, we get the tree width 0 of a graph with only one vertex and no edges. The minimal tree width 1 we can get with acyclic graphs only. The cyclic graph has a minimal tree width 2. Therefore in our example above we reached the minimal tree width.

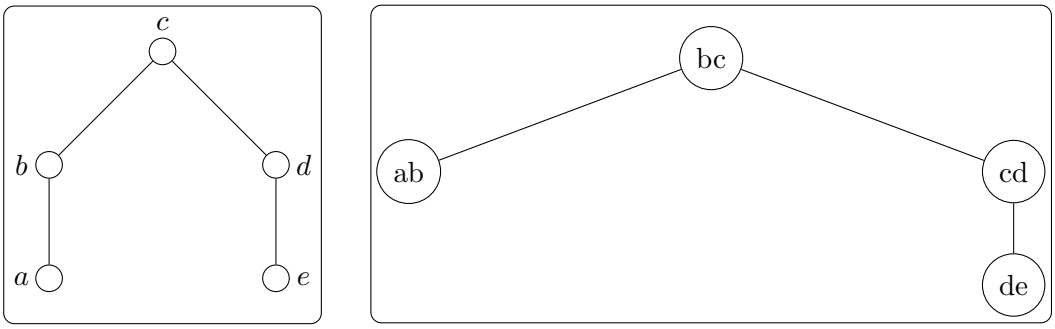
To illustrate that the cycle graph can not have a tree width 1, consider following example.

**Example 3.12.** We start with an acyclic graph $G$ and the minimal width of 1. Afterwards we will add an additional edge to make the graph $G$ cyclic and show that the tree width 1 is no longer sufficient. This is shown in Figure 3.4.

So now we modify our graph by adding an edge between the vertex $b$ and $d$ shown in Figure 3.5.

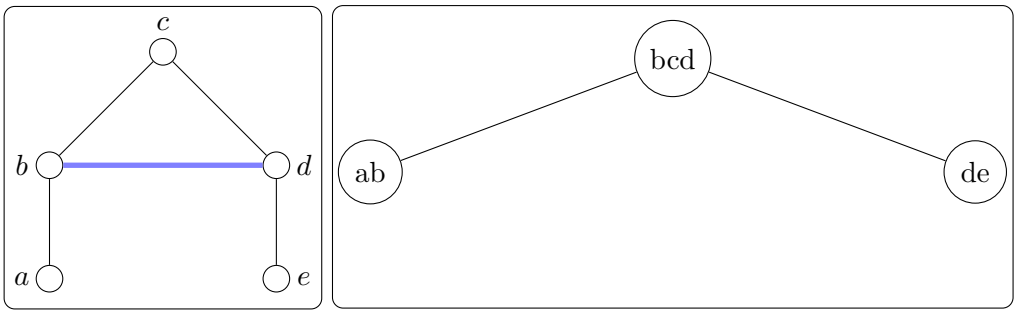
The modified graph is cycled now at vertices $b$, $c$ and $d$. So now we require a bigger bag at the root in the tree decomposition, to ensure that all tree vertices $b$, $c$ and $d$ are connected with each other.

Now it is straightforward to generalize the definitions of tree decomposition and tree width for arbitrary relational structures, in order to get finally know the Courcelle's




$(a)$ $(b)$

Figure 3.4: $(a)$ Graph $G$, $(b)$ Tree decomposition of $G$.




$(a)$ $(b)$

Figure 3.5: $(a)$ Cyclic graph $G'$, $(b)$ Tree decomposition of $G'$.

theorem and the application of it. In the following we will denote a tree as $\mathcal{T} = (T, F)$ with a set of nodes $T$ and a set of edges $F$.

**Definition 3.13** (Tree Decomposition)**.** A *tree decomposition* of a $\tau$-structure $\mathcal{A}$ is a pair

$$(\mathcal{T}, (B_t)_{t \in T}),$$

where $\mathcal{T} = (T, F)$ is a tree and $(B_t)_{t \in T}$ is a family of subsets of the universe $A$ of the $\tau$-structure $\mathcal{A}$ such that following holds:

(i) $\forall a \in A$, the set $\{x \in T \mid a \in B_t\}$ is non empty and connected in $\mathcal{T}$.

(ii) For every relation symbol $R \in \tau$ and every tuple $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$, where $r := \text{arty}(R)$, there is a $t \in T$ such that $a_1, \ldots, a_r \in B_t$.

Similarly, we adjust the terms of decomposition width and tree width for arbitrary relational structures.

**Definition 3.14** (Decomposition Width)**.** The *width* of the *decomposition* $(\mathcal{T}, (B_t)_{t \in T})$ is the number

$$\max\{|B| \mid t \in T\} - 1.$$

**Definition 3.15** (Tree Width)**.** The *tree width* $\text{tw}(\mathcal{A})$ of the $\tau$-structure $\mathcal{A}$ is the minimum of the tree decomposition of $\tau$-structure $\mathcal{A}$.

In order to get more sense for the definitions listed above for arbitrary relational structures, we consider following example.

**Example 3.16.** First we define the vocabulary $\tau$ as follows

$$\tau := \{X, Y\},$$

where $X$ is binary and $Y$ ternary. Now we define the $\tau$-structure $\mathcal{A}$ as

$$\mathcal{A} := \{A, X^{\mathcal{A}}, Y^{\mathcal{A}}\},$$

where

$$A := \{a, b, c, d\},$$

$$X^{\mathcal{A}} := \{(a, b), (b, c), (c, d), (d, e)\}, \text{ and}$$

$$Y^{\mathcal{A}} := \{(b, c, d), (b, d, c), (c, b, d), (d, b, c), (c, d, b), (d, c, b)\}.$$

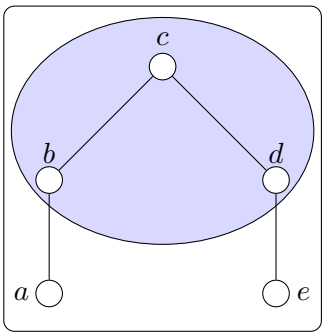



Figure 3.6: Graph $G_{\mathcal{A}}$ of a structure $\mathcal{A}$.

So now we get a known figure, namely the following

Note, that the tree decomposition is not unique. So the decomposition of Example 3.12 is valid as well as the following one

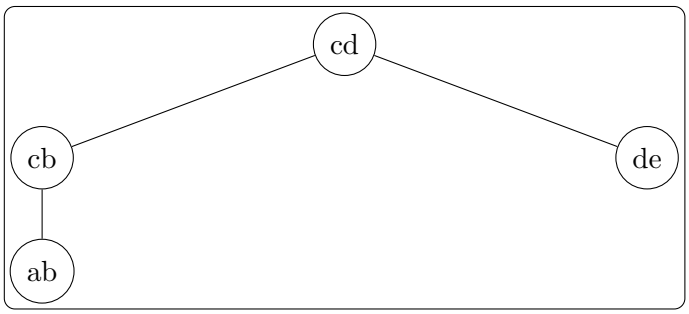



Figure 3.7: Tree Decomposition of the graph $G_{\mathcal{A}}$.

So now all of the preconditions are met to deal with Courcelle's theorem. We know the syntax and the semantics of monadic second order logic, we can deal with the tree width of arbitrary prepositional structure. Therefore in the next chapter we will discuss Courcelle's theorem, which shows that "monadic second order" definable problems are in linear time fixed parameter tractable for graphs of a fixed tree width.

#### 3.2.2.1 Theorem by Courcelle Elberfeld et al.

Courcelle's theorem is based upon Bodleander's theorem which states that for all $k$ there exists a linear time decision-algorithm which checks whether the input graph $G$ has a tree width of $k$ and in this case computes the tree decomposition of tree width $k$. Courcelle advanced the statement of Bodleander, he showed that for all MSO-formula $\phi$ and for all $k$ there exists an linear time decision-algorithm, which checks whether a

given logical structure $\mathcal{A}$ of tree width at most $k$ satisfies $\phi$. In the year 2010 Michael Eberfeld, Andreas Jakoby and Till Tantau proved that Courcelle's theorem works with logarithmic space as well as in linear time. In this chapter we follow the notation of [EJT10] and [FG06].

**Theorem 3.17** (Bodleander's Theorem)**.** There is a polynomial $p$ and an algorithm that, given a graph $G = (V, E)$, computes a tree decomposition of $G$ of width $k :=\text{kw}(G)$ in time at most

$$2^{p(k)\cdot n},$$

where $n := |V|$.

**Theorem 3.18.** Let $\mathcal{Q}$ be a decision problem, and $x$ be an input. $\mathcal{Q}$ is MSO-definable iff there exists an MSO-formula $\phi_{\mathcal{Q}}$ and a function $x \mapsto \mathcal{A}_x$ s.t. it holds that $x \in \mathcal{Q}$ if and only if $\mathcal{A}_x \models \phi_{\mathcal{Q}}$.

**Theorem 3.19** (Courcelle 1990, Elberfeld Jakoby Tantau 2010)**.** Let $\mathcal{Q}$ be an MSO-definable decision problem, and let $\mathcal{A}_x$ be a structure associated with an instance $x$. Further let $k \in \mathbb{N}$ s.t. the tree width of $\mathcal{A}_x$ is bounded by $k$. Then $\mathcal{Q}$ is solvable in time $O\Big(f(k) \cdot |x|\Big)$ and space $O\Big(\log\big(f(k)\big) + \log|x|\Big)$.

To discuss the whole proof of the theorem above is beyond the scope of this diploma thesis. In the following we give the proof idea of Courcelle's theorem. For more information we refer the reader to [EJT10].

Even for the proof idea we require some further information. First we need to define a labeling of a tree and afterwards we will get know an additional lemma.

**Definition 3.20** (Labeling of a Tree)**.** Let $\tau = \{R_1, \ldots, R_m\}$, where, for $i \in [m]$, the relation $R_i$ is $r_i$-ary. Let $\mathcal{A}$ be a $\tau$-structure and $\mathcal{D} = (\mathcal{T}, (\overline{b}^t)_{t\in T})$ an ordered tree decomposition (every tree decomposition whose bags are all nonempty immediately yields an ordered tree decomposition) of $\mathcal{A}$ of width k. *The labeling $\lambda$ of the tree $\mathcal{T} = (T, F)$* is defined then as follows.
We want the label $\lambda(t)$ to encode the isomorphism type of the substructure of $\mathcal{A}$ induced by the bag $\overline{b}^t$ and the interaction of $\overline{b}^t$ with the bag of the parent of $t$. We let

$$\lambda(t) := (\lambda_1(t), \ldots, \lambda_{m+2}(t)),$$

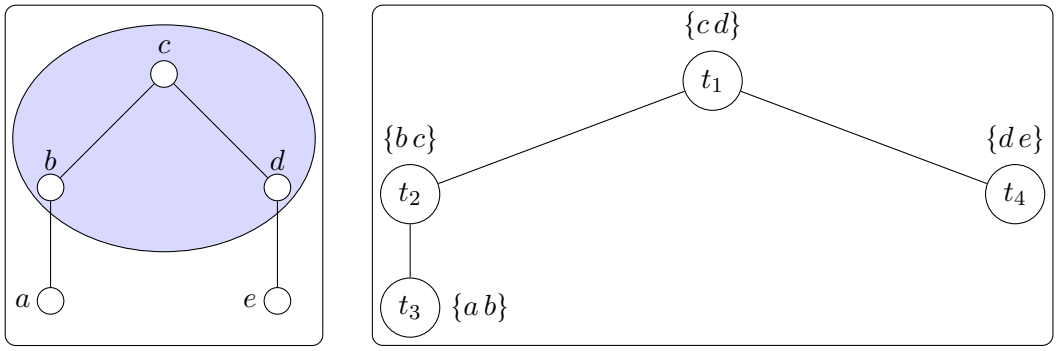



Figure 3.8: $(a)$ graph $G$, $(b)$ tree decomposition of $G$.

where

$$\lambda_i(t) := \{(j_1, \ldots, j_{r_i}) \in [k+1]^{r_i} \mid (b^t_{j_1}, \ldots, b^t_{j_{r_i}}) \in R_i^{\mathcal{A}}\}, \text{ for } 1 \leqslant i \leqslant m,$$

$$\lambda_{m+1} := \{(i,j) \in [k+1]^2 \mid b^t_i = b^t_j\},$$

$$\lambda_{m+2}(t) := \begin{cases} \{(i,j) \in [k+1]^2 \mid b^t_i = b^s_j\}, & \text{for the parants of } t \text{ if } t \text{ is not the root of } \mathcal{T}, \\ \varnothing, & \text{if } t \text{ is a root of } \mathcal{T}. \end{cases}$$

We let

$$\mathcal{T}(\mathcal{A}, \mathcal{D}) := (T, F, \lambda).$$

Note that the alphabet of $\mathcal{T}(\mathcal{A}, \mathcal{D})$ is

$$\Sigma(\tau, k) := \mathrm{Pow}([k+1]^{r_1}) \times \cdots \times \mathrm{Pow}([k+1]^{r_m}) \times \mathrm{Pow}([k+1]^2) \times \mathrm{Pow}([k+1]^2),$$

where $\mathrm{Pow}(X)$ denotes the power set of $X$.

**Example 3.21.** Consider Example 3.16 again with vocabulary $\tau = (X, Y)$, where $X$ is binary and $Y$ ternary, and the structure $\mathcal{A} := \{A, X^{\mathcal{A}}, Y^{\mathcal{A}}\}$ where

$$\begin{aligned} A :=& \{a, b, c, d\}, \\ X^{\mathcal{A}} :=& \{(a,b), (b,c), (c,d), (d,e)\}, \\ Y^{\mathcal{A}} :=& \{(b,c,d), (b,d,c), (c,b,d), (d,b,c), (c,d,b), (d,c,b)\}. \end{aligned}$$

Recall the tree decomposition has a tree width $k := \mathrm{tw}(\mathcal{A}) = 1$. The alphabet of the

tree $\mathcal{T}(\mathcal{A}, \mathcal{D})$ is defined as

$$\Sigma(\tau, k) := \mathrm{Pow}([k+1]^{r_1}) \times \cdots \times \mathrm{Pow}([k+1]^{r_m}) \times \mathrm{Pow}([k+1]^2) \times \mathrm{Pow}([k+1]^2).$$

In our case is the first relation $X$ of arity 2 and second relation $Y$ of arity 3. Thus the alphabet is

$$\Sigma(\{X, Y\}, 2) := \mathrm{Pow}([2]^2) \times \mathrm{Pow}([2]^3) \times \mathrm{Pow}([2]^2) \times \mathrm{Pow}([2]^2).$$

To give an example of a label $\lambda(t)$ in $\mathcal{T}(\mathcal{A}, \mathcal{D})$ to encode the decomposition tree $\mathcal{D}$, we consider the leaf $t_4$ with a bag $\bar{b}^{t_4} = \{d, e\}$ and get

$$\lambda(t_4) = \Big(\{(1,2)\}, \varnothing, \{(1,1),(2,2)\}, \{(1,2)\}\Big).$$

**Lemma 3.22.** Given a $\tau$-structure $\mathcal{A}$ of width $\leqslant k$, an ordered small tree decomposition $\mathcal{D}$ of $\mathcal{A}$ of width $\leqslant k$ the corresponding $\Sigma(\tau, k)$-labeled tree $\mathcal{T}(\mathcal{A}, \mathcal{D})$ can be computed in time

$$f(k, \tau) \cdot |A|$$

for a suitable computable function $f$.

**Algorithm 3:** COURCELLE

**Input**: $(\mathcal{A}, \phi)$

**if** *vocabulary* $\tau_{phi}$ *of* $\phi$ *is not contained in the vocabulary of* $\mathcal{A}$ **then**
  reject
**end**
**begin**
  Let $\mathcal{A}'$ be the $\tau_\phi$-reduct of $\mathcal{A}$.
  Compute an ordered tree decomposition $\mathcal{D} = (\mathcal{T}, (\bar{b}^t)_{t \in T})$ of $\mathcal{A}'$
  of width $\mathrm{tw}(\mathcal{A}')$ and the labeled tree $\mathcal{T}(\mathcal{A}', \mathcal{D})$.
  Compute the formula $\phi^*$.
**end**
**if** $\mathcal{T}(\mathcal{A}', \mathcal{D}) \models \phi^*$ **then**
  accept
**end**
**else**
  reject
**end**

Algorithm 3 then decides $\mathcal{A} \models \phi$. The algorithm is divided into two essential steps.

In the first step we consider $\mathcal{T}(\mathcal{A}, \mathcal{D})$ a labeled tree and every tree decomposition $\mathcal{D} = (\mathcal{T}, (B_t)_{t \in T})$ of a structure $\mathcal{A}$. Recall, tree decomposition is not unique, therefore we need to follow all alternatives. The labeled tree $\mathcal{T}(\mathcal{A}, \mathcal{D})$ must have all the information to reconstruct the structure $\mathcal{A}$. The substructure of $\mathcal{A}$ with the set of vertices $B_t$ is encoded by labeling a node $t$, additionally it gives the information about the intersection of the bags of the decomposition.
In the second part the COURCELLE algorithm translates the MSO-formulae over a structure $\mathcal{A}$ into MSO-formulae over the labeled tree $\mathcal{T}(\mathcal{A}, \mathcal{D})$ and decides whether $\mathcal{A}^{'} \models \phi^*$.

In the following we will investigate the running time of the COURCELLE algorithm. The running time of the instruction at the line 1 and 5 is $O(\|\mathcal{A}\|)$. For the further analyzation we will denote

$$
\begin{aligned}
&n := |\mathcal{A}| = |\mathcal{A}^{'}|, \\
&k := \mathrm{tw}(\mathcal{A}) \text{ the tree width of the structure } \mathcal{A}, \text{ and} \\
&\ell := |\phi|
\end{aligned}
$$

Line 3 is computable by Bodleander's theorem 3.17 and Lemma 3.22 in time

$$f_1^{'}(k^{'}, \tau_\phi) \cdot n \text{ for a suitable computable function } f_1, \text{ where } k' = \mathrm{kw}(\mathcal{A}').$$

It holds

$$f_1^{'}(k^{'}, \tau_\phi) \cdot n \leqslant f_1(k, \tau_\phi) \cdot n$$

with a suitable function $f_1$, while the size of $\tau_\phi$ is bounded by $k^{'} \leqslant k$ and $\ell$. At line 8 $\phi^*$ is computable in time $f_2(k, \ell)$, as is depends only on $\phi$ and $k^{'}$. Analogous we denote $\ell^* := |\phi^*|$. It can be shown [FG06] that the check at line 10 is computable in time $f(\ell^*) \cdot \|\mathcal{T}(\mathcal{A}^{'}, \mathcal{D})\|$. Overall we obtain the required time bound of $O(\|\mathcal{A}\|) + f(k, \ell) \cdot |A|$.

Before we get to the complex application of Courcelle's theorem in the next chapter, we consider one simple example of the application first.

**Example 3.23.** In the section of monadic second order logic we learned that the the problem 3COLORGRAPH is MSO-definable. It is easy to expand the problem 3COLORGRAPH to any number of colors , thus to problem $\alpha$-COLORGRAPH.

$$\alpha\text{-Colorgraph} := \exists C_1 \dots \exists C_\alpha \quad \forall x \forall y \Big( \bigvee_{1 \leqslant i \leqslant \alpha} C_i(x) \wedge \bigwedge_{1 \leqslant i < j \leqslant \alpha} \neg\big(C_i(x) \wedge C_j(x)\big) \wedge \bigwedge_{1 \leqslant i < j \leqslant \alpha} E(x,y) \rightarrow \neg\big(C_i(x) \wedge C_i(y)\big)\Big).$$

Now we want to show the parametrized problem of $\alpha$-Colorgraph is fixed parameter tractable. So let the instance be a graph $G$ and number of colors $\alpha \in \mathbb{N}$. We parameterize the problem by the tree width of the graph $G$. The decision problem is now the $\alpha$ colorability of the graph $G$.

Note that every not empty graph $G$ with a tree width at most $w$, has a vertex degree at most $w$ as well. It follows that the graph $G$ with $\mathrm{tw}(G) = k$ is $\alpha$-colorable $\forall k+1 \leqslant \alpha$. So the fpt-algorithm for our parameterized problem is:

**Algorithm 4:** FPT algorithm for parametrized $\alpha$-Colorgraph problem

```
   Input: (G, α)
 1 begin
 2 |  Compute the tree width of the graph k :=tw(G)
 3 |  (for example with Bodleander's Algorithm).
 4 end
 5 if k + 1 ⩽ α then
 6 |  accept
 7 end
 8 else
 9 |  if G does not satisfies the MSO-sentence of α-Colorgraph
10 |  (use Courcelle's algorithm 3) then
11 |  |  reject
12 |  end
13 end
```

So line 9 immediately decides with Courcelle's algorithm whether the instant graph $G$ is $\alpha$-colorable.

In this chapter we learned the class FPT and the technical tools to get a fpt-algorithm. In the next chapter we will learn more complex application of the Courcelle's theorem. We will show that the problems given in propositional Circumscription are fixed parameter tractable by using Courcelle's theorem.

# CHAPTER 4

# Application of Courcelle's Theorem

## 4.1 Circumscription

### 4.1.1 Basic Propositional Circumscription

The guideline in propositional Circumscription is the compliance of the minimal models to obtain as few exceptions as possible. A minimal model is a set $M_{min}$, that contains variables which are necessariely assigned to true. The central idea is that the variables which can be falsified must be assigned to false. In basic propositional Circumscription the positive assignment of propositional formulae are partially ordered according to the coordinate wise partial order $\leqslant$ on Boolean vectors, which obeys the order $0 \leqslant 1$ on $\{0, 1\}$. In the following we follow the notation of McCarthy [McC80] and [Tho10].

**Definition 4.1** (Ordering)**.** Let $M_\alpha$ and $M_\beta$ be two models of a given formula $\varphi$, if $M_\alpha$ and $M_\beta$ assign the same value to the variables $x \in \varphi$, then we write $M_\alpha(x) \leqslant M_\beta(x)$, respectively, $M_\alpha \leqslant M_\beta$ (if there exists a variable $x \in \varphi$ such that $M_\alpha(x) \neq M_\beta(x)$, we write $M_\alpha < M_\beta$).

**Definition 4.2** (Minimal model $M_{min}$)**.** Let $M_\alpha$ and $M_\beta$ be two models of a formula $\varphi$, then a *minimal model* $M_{min}$ of $\varphi$ is a satisfying model $M_\alpha$ such that there exists no satisfying model $M_\beta$, where $M_\beta < M_\alpha$ holds.

**Definition 4.3** (Circumscriptive models and inference)**.** Let $B$ and $B^{'}$ be be finite sets of Boolean functions. An assignment $\sigma$ is a *circumscriptive model* of the $B$-formula $\varphi$

(written: $\sigma \models^{circ} \varphi$) if $\sigma$ is a minimal model of $\varphi$.
A $B'$-formula $\psi$ can be *circumscriptively inferred* from $\varphi$ (written: $\varphi \models^{circ} \psi$) if $\psi$ holds in all minimal models of $\varphi$.

Lets consider the decision probleme CircInf(B), with a given formula $\phi$ and a set $\Gamma \subseteq \mathcal{L}(B)$. Now we need to decide whether a formula can be inferred from the Circumscription of a given knowledge base. We define this problem as the inference problem for $B$-formulae in propositional Circumscription.

| | |
|---|---|
| *Problem:* | CircInf(B) |
| *Input:* | A $B$-formula $\phi$, a finite set $\Gamma \subseteq \mathcal{L}(B)$ |
| *Question:* | Does $\Gamma \models^{circ} \phi$ hold? |

Michael Thomas has shown in [Tho10] the following theorem

**Theorem 4.4.** Let $B$ be a finite set of Boolean functions such that $[B] = \mathrm{BF}$, where the clone BF is based on $\{x \wedge y, \neg x\}$, thus defines all Boolean functions. Then CircInf($B$) is $\Pi_2^p$-complete.

**Example 4.5.**

$$\phi = x_1 \wedge (x_2 \vee x_3) \vee (x_2 \wedge x_4) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_4)$$

Trivially the model $M_0 = \{x_1, \ldots, x_4\}$ satisfies the function $\phi$. That means if we assign all variables $x_1, \ldots, x_4$ to true, $\phi$ gets true as well. But the model $M_0$ is not minimal. We take a closer look at $\phi$. Variables that are not possible to be falsified are connected with a Boolean conjunction. The variables which are connected with a Boolean disjunction must be assigned to false. Therefore we have three minimal models: $M_1 = \{x_1, x_2\}$, $M_2 = \{x_1, x_3\}$, $M_3 = \{x_2, x_4\}$.
Now the formula $\phi$ can be circumscribed as follows:

$$\chi = \neg\Big((x_1 \wedge x_2) \leftrightarrow (x_1 \wedge x_3)\Big) \wedge \neg\Big((x_1 \wedge x_2) \leftrightarrow (x_2 \wedge x_4)\Big) \wedge \neg\Big((x_1 \wedge x_3) \leftrightarrow (x_2 \wedge x_4)\Big).$$

Hence, overall $\phi \models^{circ} \chi$ holds.

## 4.2 MSO Encoding of basic propositional Circumscription

In this section we will express the propositional circumscriptive inference problem in monadic second order logic. We follow as antetype the paper “On the parameterized

complexity of default logic and autoepistemic logic" [MSTV12] for this chapter.

Let $B$ be a fix and finite set of Boolean functions. We define the vocabulary $\tau_B$ as follows

$$\tau_B := \{const_f^1 \mid f \in B, \text{arity}(f) = 0\} \cup \{conn_{f,i}^2 \mid f \in B, 1 \leqslant i \leqslant \text{arity}(f)\}.$$

Let $\Gamma$ be a set of propositional $B$ formulae and

$$\tau_{B,\mathcal{PL}} := \tau_B \cup \{\text{var}^1, \text{repr}_\Gamma^1, \text{repr}_\phi^1\}.$$

The $\tau_{B,\mathcal{PL}}$ structure $\mathcal{A}_{\Gamma,\phi}$ is associated with the set $\Gamma$ and a formula $\phi$ such that the universe of $A_{\Gamma,\phi}$ contains subformulae of $\Gamma$ and $\phi$ and the following holds:

(i) $\text{var}^1(x)$ holds iff $x$ represents a variable,

(ii) $\text{repr}_\Gamma^1(x)$ holds iff $x$ represents a formula in $\Gamma$,

(iii) $\text{repr}_\phi^1(x)$ holds iff $x$ represents a formula $\phi$,

(iv) $const(x)_f^1$ holds iff $x$ represents the constant $f$, and

(v) $conn_{f,i}^2(x,y)$ holds iff $x$ represents the $i$-th argument of $f$ at the root of the formula tree $y$.

**Lemma 4.6.** Let $B$ be set of Boolean functions. Then there exists an MSO-formula $\theta_{sat}$ over $\tau_{B,\mathcal{PL}}$ such that for any set of formulae $\Gamma \subseteq \mathcal{L}(B)$ over connectives in $B$ there exists a structure $\mathcal{A}_{\Gamma,\phi}$ such that

$$\Gamma \text{ is satisfiable iff } \mathcal{A}_{\Gamma,\phi} \models \theta_{circ}.$$

*Proof.* To proof the lemma we need to specify the MSO formulae.

$$\theta_{pre} := \forall x \Big( \neg\text{var}(x) \rightarrow \bigvee_{\substack{f \in B, \\ \text{arity}(f)=0}} const_f(x) \oplus \bigvee_{f \in B} \bigwedge_{1 \leqslant i \leqslant \text{arity}(f)} \exists y (conn_{f,i}(y,x) \wedge \forall z (conn_{f,i}(z,x) \rightarrow z = y)) \Big).$$

$$\theta_{struc_\Gamma} := \theta_{pre} \wedge \forall x \Big( \neg \mathrm{repr}_\Gamma(x) \rightarrow \exists y \big( \neg \mathrm{var}(y) \wedge \bigvee_{\substack{f \in B, \\ 1 \leqslant i \leqslant \mathrm{arity}(f)}} conn_{f,i}(x,y) \big) \Big).$$

$$\theta_{struc_\phi} := \theta_{pre} \wedge \forall x \Big( \neg \mathrm{repr}_\phi(x) \rightarrow \exists y \big( \neg \mathrm{var}(y) \wedge \bigvee_{\substack{f \in B, \\ 1 \leqslant i \leqslant \mathrm{arity}(f)}} conn_{f,i}(x,y) \big) \Big).$$

The structure is satisfiable, iff for all $x$ holds, if $x$ is not a variable then it is a function with well defined successor, where $x$ is either a formula or a subformula of $f$.

Let $n$ be defined as the maximum value of the arity of $f$, $n := \max\{\mathrm{arity}(f) \mid f \in B\}$. In the following we build the MSO formula for the assignment of any given modell $M$.

$$\theta_{assign}(M) := \forall x \forall y_1 \cdots \forall y_n \bigwedge_{f \in B} \Big( \bigwedge_{\mathrm{arity}(f)=0} const_f(x) \rightarrow \big(M(x) \leftrightarrow f\big) \wedge$$
$$\bigwedge_{1 \leqslant i \leqslant \mathrm{arity}(f)} conn_{f,i}(y_i, x) \rightarrow \big(M(x) \leftrightarrow f([\![y_1 \in M]\!], \ldots, [\![y_{\mathrm{arity}(f)} \in M]\!])\big)\Big),$$

where $[\![x \in M]\!]$ is true iff $x \in M$ holds and false otherwise.
The assignment of a given formula $M$ must holds that all constanst and arguments have a satisfiable assignment for the model $M$.

Finally for the satisfiability formulae of a given model $M$ holds that there is a valid structure and a valid assignment of the model $M$, where all individuals represent formulae from $\Gamma$ and respectively formula $\phi$ , and are defined as

$$\theta_{sat_\Gamma}(M) := \theta_{struc_\Gamma} \wedge \Big( \theta_{assign}(M) \wedge \forall x \big( repr_\Gamma(x) \rightarrow M(x) \big) \Big),$$

$$\theta_{sat_\phi}(M) := \theta_{struc_\phi} \wedge \Big( \theta_{assign}(M) \wedge \forall x \big( repr_\phi(x) \rightarrow M(x) \big) \Big).$$

Now we describe the minimal model in the MSO language, which states that the set $M_{min}$ is irreducible. We cannot find and remove one variable $x$ in the minimal model of

$\phi$, such that $M_{min}$ still satisfies $\phi$

$$\phi_{minimal}(M_{min}) := \forall M_{min} \neg \exists x \Big( M_{min}(x) \wedge \phi_{removed}(x) \Big)$$

where

$$\phi_{removed}(M_{min}) := \exists M'_{min} \Big( \neg M'_{min}(x) \wedge \forall y \big( M_{min}(y) \wedge x \neq y \rightarrow M'_{min}(y) \big) \wedge \phi_{sat_\Gamma}(M'_{min}) \Big)$$

All of this points to the fact that $\Gamma \models^{circ} \phi$ iff there exist a structure $\mathcal{A}_{\Gamma,\phi}$ such that $\mathcal{A}_{\Gamma,\phi} \models \theta_{circ}$, where

$$\theta_{circ} := \forall M \Big( \big( \theta_{sat_\Gamma}(M) \wedge \phi_{minimal}(M) \big) \rightarrow \theta_{sat_\phi}(M) \Big)$$

and satisfies the lemma.

□

## 4.3 MSO Encoding of a Trichotomy of Propositional Circumscription

Compared to basic Circumscription the variables of a propositional formula $\varphi$ are partitioned in three disjoined subsets $(P, Q, Z)$ now. In this chapter we follow the notation of Gustav Nordh [Nor04] as well as the notation of Arnaud Durand, Miki Herman, and Gustav Nordh [DHN12]. The partitioned subsets are defined as follows

- $P$ is the set of variables that are to minimize,
- $Q$ is the set of variables that must maintain a fixed value, in order to let the minimal models be comparable, and
- $Z$ is the set of variables allowed to vary,

The partial order on satisfying models will be defined as follows.

**Definition 4.7** ($\leqslant_{(P,Q,Z)}$ Ordering)**.** Let $M_\alpha$ and $M_\beta$ be two models of a given formula $\varphi$, then $M_\alpha \leqslant_{(P,Q,Z)} M_\beta$ if $M_\alpha$ and $M_\beta$ assign the same value to the variables in $Q$ and for every $p \in P$, $M_\alpha(p) \leqslant M_\beta(p)$ (if there exists a variable $p \in P$ such that $M_\alpha(p) \neq M_\beta(p)$, we write $M_\alpha <_{(P,Q,Z)} M_\beta$).

**Definition 4.8** (Minimal model $M_{(P,Q,Z)_{min}}$)**.** Let $M_\alpha$ and $M_\beta$ be two models of a formula $\varphi$, then a minimal model $M_{(P,Q,Z)_{min}}$ of $\varphi$ is a satisfying model $M_\alpha$ such that there exists no satisfying model $M_\beta$, where $M_\beta <_{(P,Q,Z)} M_\alpha$ holds.

| | |
|---|---|
| *Problem*: | $\textsc{CircInf}(B)_{(P,Q,Z)}$ |
| *Input*: | A $B$-formula $\phi$, a set $\Gamma \subseteq \mathcal{L}(B)$, and a partition $(P, Q, Z)$ of the propositions |
| *Question*: | Does $\Gamma \models^{circ}_{(P,Q,Z)} \phi$ hold? |

As well as for basic propositional Circumscription the complexity of $\textsc{CircInf}(B)_{(P,Q,Z)}$ is given in [Tho10].

**Theorem 4.9.** Let $B$ be a finite set of Boolean functions such that $[B] = \mathrm{BF}$, where the clone BF is based on $\{x \wedge y, \neg x\}$, thus defines all Boolean functions. Then $\textsc{CircInf}(B)_{(P,Q,Z)}$ is $\Pi^p_2$-complete.

Now we give the MSO-Encoding of the disjoined subsets partition $(P, Q, Z)$ of formula $\varphi$:

$$\begin{aligned}\varphi_{part} := \exists P \exists Q \exists Z \forall x \Big( & P(x) \rightarrow \neg\big(Z(x) \vee Q(x)\big) \wedge \\ & Q(x) \rightarrow \neg\big(P(x) \vee Z(x)\big)\Big) \wedge \\ & Z(x) \rightarrow \neg\big(P(x) \vee Q(x)\big) \wedge \\ & \forall x\Big(\mathrm{var}(x) \rightarrow \big(P(x) \vee Q(x) \vee Z(x)\big)\Big).\end{aligned}$$

Additionally MSO-encodings are required for the set of variables that are to minimize as well as for the set of variables that must maintain a fixed value. We start with the set $P$. It must hold that $M_{min} \subseteq P$ and $M_{min}$ is a valid minimal model, so it follows

$$\varphi_P := \forall M_{min}\Big(\forall x\big(M_{min}(x) \wedge \phi_{minimal}(M_{min}) \rightarrow P(x)\big)\Big).$$

The variables in $Q$ must maintain a fix value $\in \{0, 1\}$, so we define

$$\begin{aligned}\varphi_Q := \forall M_{min_0} \forall M_{min_1} \Big(\forall x\big(Q(x) \rightarrow \big(M_{min_0}(x) \leftrightarrow M_{min_1}(x)\big)\big) \wedge \\ \phi_{minimal}(M_{min_0}) \wedge \phi_{minimal}(M_{min_1})\Big).\end{aligned}$$

The set $Z$ needs no further consideration, as the variables are allows to vary here.

Compared to basic Circumscription we need to add $\varphi_P$, $\varphi_Q$ and $\varphi_{part}$ in the implementation now

$$\theta_{sat_{(P,Q,Z)}} := \theta_{sat} \wedge \varphi_P \wedge \varphi_Q \wedge \varphi_{part}.$$

**Example 4.10.** In order to compare we take the same formula as in basic Circumscription. So consider the input

$$\phi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_4),$$

with

$$P = \{x_1, x_3\},\ Q = \{x_2\},\ Z = \{x_4\}.$$

In this case we can falsify $x_1$ as well as $x_3$, hence the first and the second Boolean and-operator we conclude to be false. Overall we get only one minimal model $M_1 = \{x_2, x_4\}$ and $\phi$ can be circumscribed as follows

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_4) \models^{circ}_{(P,Q,Z)} x_2 \wedge x_4.$$

**Example 4.11.** Now we extend our formula $\phi$ with an additional clause

$$\phi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_2 \wedge x_5),$$

with

$$P = \{x_1, x_3\},\ Q = \{x_2\},\ Z = \{x_4, x_5\}.$$

We can falsify $x_1$ as well as $x_3$ in this case again, hence the first and the second Boolean conjunction are going to be false. But now we get two minimal models $M_1 = \{x_2, x_4\}$ and $M_2 = \{x_2, x_5\}$. Note $x_2$ should be fixed, therefore $M_1$ contains $x_2$ as well as $M_2$, so $\phi$ can be circumscribed as follows

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_4) \models^{circ}_{(P,Q,Z)} \neg\Big((x_2 \wedge x_4) \leftrightarrow (x_2 \wedge x_5)\Big).$$

Using Courcelle's theorem we can show the following result now.

**Theorem 4.12.** Let $B$ be a set of Boolean functions. Then it holds $\textsc{CircInf}(B)_{(P,Q,Z)}$ is fixed parameter tractable.

*Proof.* With Lemma 4.6 we have shown that we have a monadic second order definable satisfiability formula over the structure $\mathcal{A}_{\Gamma,\phi}$ for the $\textsc{CircInf}(B)_{(P,Q,Z)}$ problem. Thus, with Courcelle's theorem 3.19 holds $\textsc{CircInf}(B)_{(P,Q,Z)}$ is fixed parameter tractable. □

Note that the theorem 4.12 especially holds for the CircInf($B$) problem, as in this case the sets $Q$ and $Z$ are empty.

# CHAPTER 5

# Conclusion

In this thesis we have shown that the inference problem of Circumscription is fixed parameter tractable. One main challenge here was to formalize the inference problem of Circumscription in a way that Courcelle's theorem can be applied. We used the fact shown by Courcelle that a given problem is fixed parameter tractable in case it is MSO definable and parametrized by the tree width.

The problem by using Courcelle's theorem is to find the tree decomposition. The tree decomposition algorithm runs in $O(2^{35 \cdot k^3} \cdot |x|)$ for fixed tree width $k$ and for an instance $x$. Thus by the large exponent constant factor the algorithm is not practical, as shown in the tutorial by Downey [Dow12]. It remains open whether there exists an efficiently computable tree decomposition. However, the running time in this diploma thesis is of a minor importance, as we are interested in the theoretical aspect only.

For the inference problem of Circumscription we give a complete result for arbitrary sets of Boolean functions similar to [MSTV12] for Default logic and Autoepistemic logic. We leave it as an open question whether there are lower bounds analogous for the Circumscription inference problem, like they were given in [MSTV12] for Default logic and Autoepistemic logic.

# Bibliography


[Bey09] Dr. Olaf Beyersdorff, *Logik*, Institut für Theoretische Informatik, Leibniz Universität Hannover, 2009.

[Cou90] Bruno Courcelle, *The monadic second-order logic of graphs. i. recognizable sets of finite graphs*, Inf. Comput. **85** (1990), no. 1, 12–75.

[DHN12] Arnaud Durand, Miki Herman, and Gustav Nordh, *Trichotomies in the complexity of minimal inference*, Theory Comput. Syst. **50** (2012), no. 3, 446–491.

[Dow12] Rod Downey, *A parameterized complexity tutorial*, Language and Automata Theory and Applications, Springer Berlin Heidelberg, 2012.

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau, *Logspace versions of the theorems of bodlaender and courcelle*, Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 2010.

[FG06] J. Flum and M. Grohe, *Parameterized complexity theory*, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Imm99] Neil Immerman, *Descriptive complexity*, Springer, 1999.

[McC80] John McCarthy, *Circumscription - a form of non-monotonic reasoning*, Artificial Intelligence **13** (1980), no. 1-2, 27–39.

[MSTV12] Arne Meier, Johannes Schmidt, Michael Thomas, and Heribert Vollmer, *On the parameterized complexity of default logic and autoepistemic logic*, Proceedings of the 6th International Conference on Language and Automata Theory and Applications, 2012, pp. 389–400.

[Nor04] Gustav Nordh, *A trichotomy in the complexity of propositional circumscription*, LPAR (Franz Baader and Andrei Voronkov, eds.), Lecture Notes in Computer Science, vol. 3452, Springer, 2004, pp. 257–269.

[Tho10] Michael Thomas, *On the complexity of fragments of nonmonotonic logics*, vol. 1, Cuvillier Verlag, 2010.