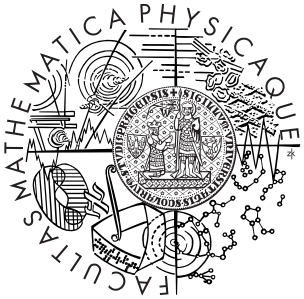
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Vendula Uchytilová

Algebraický přístup k CSP (Algebraic Approach to CSP)

Katedra algebry

Vedoucí bakalářské práce: Mgr. Pavel Růžička, Ph.D.
Studijní program: Matematika, obecná matematika

2009

Děkuji Mgr. Pavlu Růžičkovi, Ph.D za odborné vedení této práce. Dále bych chtěla poděkovat Abigail Smith za jazykovou korekturu a hlavně Tomáši Jirotkovi za faktické připomínky k této práci.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 7.8.2009 Vendula Uchytilová

Contents

Název práce: Algebraický přístup k CSP
Autor: Vendula Uchytilová
Katedra (ústav): Katedra algebry
Vedoucí bakalářské práce: Mgr. Pavel Růžička, Ph.D.
e-mail vedoucího: Pavel.Ruzicka@mff.cuni.cz

Abstrakt: V této práci sledujeme složitost CSP vzhledem k operacím v množině všech polymorfismů. Postupně procházíme jednotlivé operace a rozhodujeme, které z nich zaručují polynomiální složitost a které **NP**-úplnost. Dále se zabýváme algebrami a varietami. Pomocí CSP zavádíme algebry s polynomiální časovou složitostí a **NP**-úplné algebry. Navíc pomocí (l, k)-algoritmu definujeme algebry omezené šířky. V poslední řadě přikračujeme ke složitosti surjektivních algeber.
CSP definujeme jako třídu rozhodovacích problémů, zda existuje homomorfismus z dané konečné relační struktury do fixní konečné relační struktury. V některých částech textu, především v sekcích věnovaných konzistenci, přecházíme k názvosloví proměnné a hodnoty.

Klíčová slova: CSP, polymorfismus, algebra omezené šířky

Title: Algebraic Approach to CSP
Author: Vendula Uchytilová
Department: Department of Algebra
Supervisor: Mgr. Pavel Růžička, Ph.D.
Supervisor's e-mail address: Pavel.Ruzicka@mff.cuni.cz

Abstract: In the present work we study the complexity of CSP towards operations in the set of all polymorphisms. Gradually we go through every operation and we decide which one ensures tractability and which one ensures **NP**-completeness. Further we concern with algebras and varieties. Using CSP we define algebras with polynomial-time complexity and **NP**-complete algebras. In addition using the (l, k)-algorithm we define bounded width algebras. After all we proceed to the complexity of surjective algebras. We define CSP as a class of decision problems whether any homomorphism from a given finite relational structure to a fixed finite relational structure exists. In some parts of the text, particulary in sections devoted to the consistency theory, we convert the terminology to variables and values.

Keywords: CSP, polymorphism, bounded width algebra

Introduction

Constraint satisfaction problems have a wide range of applications in real life. Nowadays, a well known example is the Sudoku game which became popular among people. However it is not just Sudoku or a graph coloring problem which make CSPs useful. Typical representatives of CSPs are timetables and scheduling in different areas from job tasks to airline timetables. As a such example we can assume a school timetable.

We receive subjects with their hour dotation per week and a list of teachers. As constraints we can take time when a subject can be teached, that pupils can have only one subject at the same time, pause for lunch and that one teacher can teach only one subject at the same time. This is a well known problem which has to be solved every year before the first school day. We can also look for optimal solutions.

Applications of CSPs are also in areas which seems to have nothing in common with mathematics and programming such as in biomedicine, telecommunications, syntactic analysis of natural-language sentence and in other areas. We can also find different modifications of constraint satisfaction problems.

Another task is how we can solve such problems. Basic methods are testing every combination and backtracking. Advaced methods are constraint-propagation or consistency methods. However in this text we will not look at these methods but at the complexity of such problems. We will fix one structure and say which operation from the set of all polymorphisms ensures tractability and which one ensures **NP**-completeness.

Chapter 1

Polymorphisms

In the first chapter there are definitions of relational structures, constraint satisfaction problems and polymorphisms. We show that a set of all polymorphisms forms a clone.

Later we give an introduction to the complexity theory where we define a Turing machine and time and space complexity. Especially we put our attention to tractable problems, i.e. to problems in a polynomial-time class, and to **NP**-complete problems. In conclusion of this section we state a few **NP**-complete problems which we will use in the next chapter.

In the last section of the first chapter we state lemmas about reductions between two CSPs. For proof we will need notion of the Gallois correspondence. We will use these lemmas in the next chapters.

1.1 Relational structures and constraint satisfaction problems

The first section includes key definitions for the whole paper. In particular definitions of relational structures, homomorphisms between structures, constraint satisfaction problems, polymorphisms and invariants.

We will also state an important lemma about a set of all polymorphisms. The set of all polymorphisms constitutes a clone and therefore from properties of clones only some operations can be polymorphisms. We will use this fact in the second chapter where we will look properly at such operations.

Definitions are from [2] Larose, Zadori: Bounded width problems and

algebras or from [3] Jeavons: On the algebraic structure of combinatorial problems from where the lemma about operations in a clone is. Its proof is based on the paper [9] Rosenberg: Minimal clones I., the five types.

Definition 1.1.1. An *n-ary relation* over a set D is a subset of D^n. *Relational symbols* denote relations and a *vocabulary* is a set of relational symbols.

Definition 1.1.2. A *Relational structure* is a pair of a non-empty set, called a *universe*, and of a system of finitary relations over the universe.

A relational structure is *finite* if its universe and also its system of finitary relations are both finite.

Definition 1.1.3. Two relational structures are *similar* if they have the same vocabulary.

Remark. In the whole text we will use a letter in a caligraphic font to denote a relational structure and the same letter in an italic font for its universe. Relations from a structure will have a symbol of the structure as an upper index to emphasize where they belong.

We will turn our attention to a definition of a constraint satisfaction problem on finite structures. At first we will give a definition of a non-uniform problem and later also of an uniform problem.

Definition 1.1.4. A *homomorphism* from a relational structure $\mathcal{G}$ to a similar relational structure $\mathcal{H}$ is a mapping $\varphi : G \to H$ between its universes, such that for every n-tuple from an n-ary relation from the structure $\mathcal{G}$ its image is in a corresponding n-ary relation from the structure $\mathcal{H}$, i.e.

$$\forall(a_1, a_2, \ldots, a_n) \in R^{\mathcal{G}} : (\varphi(a_1), \varphi(a_2), \ldots, \varphi(a_n)) \in R^{\mathcal{H}}.$$

Definition 1.1.5. Let us have a finite relational structure $\mathcal{H}$. A *non-uniform constraint satisfaction problem*, $CSP(\mathcal{H})$, is a decision problem whether any homomorphism from a given similar relational structure $\mathcal{G}$ to the relational structure $\mathcal{H}$ exists.

The structure $\mathcal{G}$ is called an *instance* of $CSP(\mathcal{H})$, the structure $\mathcal{H}$ is called a *template* of $CSP(\mathcal{H})$. A *solution* of a constraint satisfaction problem is any homomorphism from the instance to the template.

In the next sections we will look at complexity of constraint satisfaction problems. For this purpose we define a special operation – a polymorphism and a special relation – an invariant, which play the main role in our text.

Definition 1.1.6. An *n-ary operation* on a set D is a function $f : D^n \to D$.

Definition 1.1.7. Let R be an n-ary relation over a set D. Let f be a k-ary operation on the same set D. The operation f on n-tuples from the relation R is defined :

$$f(a_1, a_2, \ldots, a_k) = f((a_{11}, \ldots, a_{1n}), (a_{21}, \ldots, a_{2n}), \ldots, (a_{k1}, \ldots, a_{kn})) =$$

$$f(a_{11}, a_{21}, \ldots, a_{k1}), \ldots, f(a_{1n}, a_{2n}, \ldots, a_{kn}))\forall a_i \in R.$$

In a natural way we define an image of the relation R under the operation f as $f(R) = \{f(a_1, a_2, \ldots, a_k); a_1, a_2, \ldots, a_k \in R\}$.

Definition 1.1.8. A *polymorphism* of relation R is an operation f such that $f(R) \subseteq R$.
On the other hand the relation R is an *invariant* of the operation f.

Definition 1.1.9. A *polymorphism* of a relational structure is an operation which is a polymorphism of every relation in the structure. An *invariant* is defined in a similar way.

A set of all polymorphisms of relational structure $\mathcal{H}$ is denoted by $Pol(\mathcal{H})$. A set of all invariants of every operation from set ϕ is denoted by $Inv(\phi)$.

We have mentioned major definitions so it is time for an example to get better notion of them.

Example. *We will assume a relational structure* $\mathcal{H} = (H, R_1^{\mathcal{H}}, R_2^{\mathcal{H}}, R_3^{\mathcal{H}})$ *where* $H = \{0, 1, 2\}$ *is the universe of the structure with relations:*
$R_1^{\mathcal{H}} = \{(021), (112), (120), (121), (122)\}$,
$R_2^{\mathcal{H}} = \{(00), (01), (02), (21)\}$,
$R_3^{\mathcal{H}} = \{(0210), (0212), (1210), (2202), (2210)\}$.
We will also get a similar relational structure $\mathcal{G} = (G, R_1^{\mathcal{G}}, R_2^{\mathcal{G}}, R_3^{\mathcal{G}})$ *where the universe is* $G = (a, b, c, d)$ *and relations are*
$R_1^{\mathcal{G}} = \{(aab), (abc), (cda)\}$,
$R_2^{\mathcal{G}} = \{(ba), (cc), (cd), (da)\}$,
$R_3^{\mathcal{G}} = \{(abac), (cdab), (dbcd)\}$.
A solution of a problem with the instance $\mathcal{G}$ *and the template* $\mathcal{H}$ *can be for example mapping*

$$a \mapsto 1, b \mapsto 2, c \mapsto 0, d \mapsto 2.$$

It is a homomorphism since $(aab) \mapsto (112)$ *is a triple from relation* $R_1^{\mathcal{H}}$, $(abc) \mapsto (120)$ *and* $(cda) \mapsto (021)$ *are triples from relation* $R_1^{\mathcal{H}}$. *We can similarly show for the rest relations* $R_2^{\mathcal{G}}$ *and* $R_3^{\mathcal{G}}$.

We will look for a polymorphism of structure $\mathcal{H}$. *We will assume a symetric operation* f *defined:*

f	0	1	2
0	0	1	0
1	1	1	2
2	0	2	2

.

We will not show for every relation that the operation f *is a polymorphism. We will show it only on relation* $R_1^{\mathcal{H}}$.

From the table we can see that the operation f *is a symetric binary operation. Let us assume for example triples* $(021), (122)$. *If we apply the operation* f *on these triples we receive triple* $(f(01), f(22), f(12)) = (122)$. *We will state these results in a table.*

triple a	*triple b*	*f(a,b)*	*triple* $c = f(a,b)$
(021)	(021)	(f(00),f(22),f(11))	(021)
(021)	(112)	(f(01),f(21),f(12))	(122)
(021)	(120)	(f(01),f(22),f(10))	(121)
(021)	(121)	(f(01),f(22),f(11))	(121)
(021)	(122)	(f(01),f(22),f(12))	(122)
(112)	(112)	(f(11),f(11),f(22))	(112)
(112)	(120)	(f(11),f(12),f(20))	(120)
(112)	(121)	(f(11),f(12),f(21))	(122)
(112)	(122)	(f(11),f(12),f(22))	(122)
(120)	(120)	(f(11),f(22),f(00))	(120)
(120)	(121)	(f(11),f(22),f(01))	(121)
(120)	(122)	(f(11),f(22),f(02))	(120)
(121)	(121)	(f(11),f(22),f(11))	(121)
(121)	(122)	(f(11),f(22),f(12))	(122)
(122)	(122)	(f(11),f(22),f(22))	(122)

.

On the right side are triples $(021), (122), (121), (112), (120)$ *which are all from relation* $R_1^{\mathcal{H}}$ *and thus the operation* f *is a polymorphism of the relation. In a similar manner we can show the same thing for relations* $R_2^{\mathcal{H}}, R_3^{\mathcal{H}}$. *Thus* $f \in Pol(\mathcal{H})$.

Later we will find out that f is an example of a binary idempotent operation.

Now we will show that the set $Pol(\mathcal{H})$ forms a *clone*. A clone is a set of operations on a finite set, which contains all the projections and is closed under composition. A formal definition follows.

Definition 1.1.10. Let ϕ be a set of operations on a finite set D. The set ϕ is a *clone* if:

- A projection $\pi_i^k : D^k \to D$, where $\pi_i^k(x_1, x_2, \ldots, x_k) = x_i$ is in the set ϕ.
- A k-ary operation ψ such that

  $$\psi(x_1, x_2, \ldots, x_k) = \varphi(\varphi_1(x_1, x_2, \ldots, x_k), \ldots, \varphi_m(x_1, x_2, \ldots, x_k))$$

  is in the set ϕ for any k-ary operation φ_i from the set ϕ and for any m-ary operation φ from the set ϕ.

Lemma. *A set of all polymorphisms, $Pol(\mathcal{H})$, constitutes a clone.*

Proof. Suppose that $R^{\mathcal{H}}$ is an n-ary relation and $a_1, a_2, \ldots, a_k$ are n-tuples from $R^{\mathcal{H}}$.

1. Applying a projection π_i^k on $a_1, a_2, \ldots, a_k$ yields n-tuple a_i from $R^{\mathcal{H}}$.
2. Let us take k-ary operations $\varphi_1, \varphi_2, \ldots, \varphi_m$ and an m-ary operation φ from the set $Pol(\mathcal{H})$. Let us define a new n-tuple $b_i := \varphi_i(a_1, a_2, \ldots, a_k)$. Since the operation φ_i is a polymorphism, the tuple b_i is in the relation $R^{\mathcal{H}}$. Similary $\varphi(b_1, b_2, \ldots, b_m)$ is in the relation $R^{\mathcal{H}}$, as the operation φ is a polymorphism.

□

Proposition 1.1.11. *A set of all polymorphisms, $Pol(\mathcal{H})$, contains operations of a specific type:*

- *An essentially unary operation only, i.e. an operation φ such that $\varphi(a_1, a_2, \ldots, a_k) = f(a_i)$, where f is a non-constant unary operation.*
- *Or at least one from the following list*

1. *A constant operation, i.e. an operation φ such that $\varphi(a_1, \ldots, a_k) = c$ where c is a constant.*
2. *An idempotent binary operation which is not a projection, i.e. an operation φ such that $\varphi(a, a) = a$ and the operation φ is not a projection.*
3. *A majority operation, i.e. an operation φ such that $\varphi(a_1, a_1, a_2) = \varphi(a_1, a_2, a_1) = \varphi(a_2, a_1, a_1) = a_1$.*
4. *An affine operation, i.e. an operation φ on a set H such that $\varphi(a_1, a_2, a_3) = a_1 - a_2 + a_3$ where $(H, +, -)$ is an Abelian group and $a_1, a_2, a_3 \in H$.*
5. *A semiprojection, i.e. an n-ary operation φ such that $\varphi(a_1, a_2, \ldots, a_n) = a_i$, $\forall(a_1, a_2, \ldots, a_n)$ where $|\{a_1, a_2, \ldots, a_n\}| < n$, $n \geq 3$ and φ is not a projection.*

Proof. Proof of the proposition is based on the clone theory. We will not give a complete proof here as it is long and more complicated. Our aim is to give an idea how it can be done. We will start with knowledge that for structure $\mathcal{H}$ the set of all polymorphisms $Pol(\mathcal{H})$ is a clone.

Clones are ordered by an inclusion : $P \subseteq C_1 \subseteq \ldots$ where P is a set of all projections and C_1 is a *minimal* clone. The minimal clone C_1 can be expressed as $\langle f \rangle$, where f is an operation from C_1 which is not a projection, and $\langle f \rangle$ means the least clone which contains f. From ordering of clones follows that every clone contains some minimal clone.

I. E. Rosenberg showed in 1983 that minimal clones contain only specific operations:

- a constant operation
- an idempotent binary operation which is not a projection
- a majority operation
- an affine operation
- a semiprojection
- a non-identity unary operation

Suppose an operation g from the set $Pol(\mathcal{H})$ which is not an essentially unary operation and has the smallest arity among such operations in $Pol(\mathcal{H})$. Every minimal clone can be viewed as a union of projections and minimal element from a block of equivalence given by $f \leftrightarrow g$ iff f is in $\langle g \rangle$ and g is in $\langle f \rangle$.

Thus if we consider g, it constitutes a minimal clone and thus it can be either a constant operation, an idempotent binary operation which is not a projection, a majority operation, a generalised parity operation or a semiprojection. □

Remark. Any projection is just a special type of an essentially unary operation. Thus it is contained in $Pol(\mathcal{H})$.

1.2 Time and Space Complexity

This section is devoted to the complexity theory. Cornerstones are Turing machines. We distinguish between two types - deterministic one and non - deterministic one. We use them for defining two important complexity classes - polynomial one and non-deterministic polynomial one. We also define logarithmic and complete classes.

From now we can work with tractable and **NP**-complete problems. In conclusion we state few **NP**-complete problems which we will need in the next chapter. Key text for this section is [1] Garey, Johnson: Computers and Intractability: A guide to the theory of NP-completeness.

For our purpose we need to define a notion of time and space complexity. Ussually we look for a worst possible case of time and space among the same size of input.

An *algorithm* is a step-by-step procedure, *time* is the number of steps of an algorithm solving a problem with a certain input. Polynomial time algorithms are those whose time can be bounded by a polynomial.

For a formal definition we need to fix a computational model.

Definition 1.2.1. Assume an infinite tape divided into squares and a read-write head which looks on one square of the tape. Let us have two finite sets Σ and Q where Σ is a set of tape symbols containing one unique blank symbol b and Q is a set of states containing a special start-state q_s. A foursome $T = (Q, \Sigma, q_s, M)$, where $M = \{\leftarrow, \rightarrow, -\}$ is a set of directions, is a *one-tape Turing machine*, TM.

Definition 1.2.2. A *Program* is a subset of $Q \times \Sigma \times Q \times \Sigma \times M$ consisting of instructions (q, s, q', s', m). If there is at most only one instruction for q, s then we call T a *deterministic one-tape Turing maching*. If there is for some q, s more than one instruction we call T a *nondeterministic one-tape Turing machine*.

An instruction (q, s, q', s', m) tells a read-write head to erase s and write s' in its place. The read-write head moves one square to the left if $m = \leftarrow$, one square to the right if $m = \rightarrow$ and does not move if $m = -$. At the same time, state q changes to q'.

An *input* to TM is a string from $\Sigma^\star$ where $\Sigma^\star$ is a set of finite sequences of symbols from Σ. A program halts if there is no instruction for a current symbol s and a current state q. A sequence of non-empty symbols is in this case an *output* of the program.

Now we can define complexity classes.

Definition 1.2.3. A *time complexity function*, T_T, for a deterministic program T is a maximum of time among inputs of the same size, i.e. $T_T(n) = max_{|x|=n}\{t\text{: }\textit{computation of } T \textit{ on a input } x \textit{ takes time } t\}$ where time is a number of steps in the computation up until the program halts.

In a similar way a *space complexity function*, S_T, for a deterministic program T is a maximum of space among inputs of the same size, i.e. $S_T(n) = max_{|x|=n}\{s\text{: }\textit{computation of } T \textit{ on an input } x \textit{ needs space } s\}$ where space is a number of diferent squares visited during computation without assuming squares with input.

These are the cases of deterministic complexity functions. If we want a nondeterministic case we have to take time as a minimum of steps required up until the program halts. Similarly with space as there can be more than one way for which the program halts.

Definition 1.2.4. A *non-deterministic time complexity function*, T_T, for a non-deterministic program T is $T_T(n) = max_{|x|=n} min_t\{t\text{: }\textit{computation of } T \textit{ on an input } x \textit{ takes time } t\}$ where time is a number of steps in the computation up until the program halts.

Definition 1.2.5. The *polynomial* class **P** is a class of problems which can be solved by a program T with a polynomial time complexity function. Similarly a problem is in the *logarithmic-space* class **L** if it can be solved by a program T with a logarithmic space complexity funciton.

Definition 1.2.6. A *non-deterministic polynomial* class **NP** is a class of problems which can be solved by a non-deterministic program T with a polynomial non-deterministic time complexity function.

We can define class **NP** in a diferent way as a class of problems where we can verifie in polynomial time if something is or is not a solution to the problem. Both ways define a class of the same problems. Good picture for ilustrating non-deterministic machine is a picture of a tree.

Now we will state reductions between problems.

Definition 1.2.7. Problem A is *polynomial time reducible* to problem B if there is a function f computable by a polynomial time program such that x is a solution to A if and only if $f(x)$ is a solution to B.

A *log-space reduction* is the same as polynomial time reduction except, insted of polynomial time is a logarithmic space computability.

Any reduction is a transitive relation, i.e if A is reducible to B and B to C then also A is reducible to C where all reductions are of the same type. Now we will look at another type of problem.

Definition 1.2.8. Problem A is **NP**-complete if it is in the class **NP** and every problem B from the class **NP** is polynomial time reducible to the problem A.

Now we can look on a relation between complexity classes and reductions. If problem A is polynomial time reducible to problem B from the polynomial time class then also A is from the polynomial time class. If **NP**-complete problem A is polynomial time reducible to problem B from the class **NP** then also B is an **NP**-complete problem. If problem A is log-space reducible to problem B from the logarithmic space class then also A is in the logarithmic space class.

These conditions give us an idea how to show that something is **NP**-complete or in **P**. At first we will use the word *tractable* when we want to tell that something is in the class **P**. If we want to show that problem A is **NP**-complete we have can use the following strategy:

- A is an **NP** problem
- find proper **NP**-complete problem B
- find a polynomial time reduction from B to A

If we want to show that problem A is tractable we can :

- find proper tractable problem B
- find a polynomial time reduction from A to B

In the text we will need two **NP**-complete problems. For their definition we will need some new terms. *Variables* and negations of variables makes *literals*. In fact first one is a positive literal and the second one is a negative literal. A *clause* is a disjunction of literals and thus a formula is in a *conjunctive normal form* if it is expressed as a conjuction of clauses.

First of all we should say what a satisfiability problem, abreviated to SAT, is. A *SATISFIABILITY* problem is a decision problem whether for a boolean formula in a conjunctive normal form exists an assignment to the variables true or false in a way that the whole formula is true. Its **NP**-completness was shown in 1971 by Cook. Nowadays **NP**-completness of SAT is stated as Cook's theorem.

One modificiation of a famous SAT problem is 3-SAT. In general problem *k-SAT* is a satisfiability problem with exactly k literals per every clause. This means that 3-SAT problem has exactly 3 literals per every clause of a boolean formula in a conjunctive normal form. This problem is also known to be **NP**-complete. A way of showing it is via reduction from SAT.

A *NOT-ALL-EQUAL SATISFIABILITY* problem is a problem where every clause contains exactly 3 literals and question is whether we can find a satisfying assignment such that whole formula is true and each clause has at least one true literal and at least one false literal. Usually it is abreviated to NAESAT.

This problem was shown to be **NP**-complete and a general proof can be find in work The complexity of Satisfiability problems by Schaefer. It can be shown via reduction from SATISFIABILITY to 3SAT and then to our NOT-ALL-EQUAL.

The second problems is a *GRAPH K-COLORABILITY* problem which is a problem with a given graph and the positive integer K. The question is whether there exists a function f from the set of vertices to the set $\{1, 2, \ldots, K\}$ such that every two vertices joined by an edge have a different number under f, i.e. for graph $G = (V, E)$: for every $\{u, v\} \in E$ holds $f(u) \neq f(v)$.

Proof of **NP**-completness can be found in Reducibility among combinatorial problems by Karp. It can be shown via reduction from SATISFIABILITY to 3SAT and then to our GRAPH K-COLORABILITY.

The last problem we will need is a *HORN-CLAUSE SATISFIABILITY* with at most 3 literals per clause which is known to be **P**-complete. A horn clause is a clause with at most one positive literal in every clause.

We will give a few examples of diferent satisfiability problems.

Example. *We can now show a few examples of different satisfiability problems. For example formula* $f = (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$ *is in a conjunctive normal form and it is an example of SAT. If we for example assign* $x_1 := 1, x_2 := 1, x_3 := 1$, *first clause has value* 1 *and the second clause has also value* 1 *and thus the whole formula is true. It is also an example of 2-SAT as it has exactly two literals in both clauses.*

Another example is a formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$. *For example assignment* $x_1 := 1, x_2 := 1, x_3 := 0, x_4 := 0$ *yields that the whole formula is true. It is an example of SAT. As it has exactly three literals per clause it is also an example of true 3-SAT. Similar as in the first clause is one true literal* x_1 *and two false literals* $\neg x_2, x_3$ *and as in the second clause is one true literal* x_2 *and two false literals* $\neg x_1, x_4$ *it is also an example of true NAESAT if we use the former assignment.*

What if we look for an example of a problem which has true assignment as 3-SAT and has false assignment as NAESAT? A formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$ *is such an example. An assignment* $x_1 := 1, x_2 := 1, x_3 := 0$ *makes formula true and thus the formula is true as 3-SAT problem.*

If we look for a true assignment as a NAESAT problem, we fail. At first we should realize that the formula is symetric in variables. At first we will look at the last clause. If we take an assignment $x_1 := 0, x_2 := 1, x_3 := 1$, *then the third clause is* $1 \vee 1 \vee 1$ *and thus there is no negative literal in this clause. If we assume an assignment* $x_1 := 0, x_2 := 0, x_3 := 1$, *the the first clause is* $0 \vee 0 \vee 0$ *and there is no positive literal in the clause. Now because of the symetricity we have assumed all posibilities how to make the problem true as NAESAT from perspective of the last clause. Thus the formula is never true as NAESAT.*

A former examples have never fullfilled the definition of HORN-SAT as there were two or more positive literals. An example of HORN-SAT is a formula $(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$. *An assignment* $x_1 := 1, x_2 :=$

$1, x_3 := 1$ *makes the formula true. As it has 3 literals per every clause it is also an example of 3-SAT and also an example of NAESAT as there is always one true literal and two false literals per claues.*

1.3 Reductions between CSPs

In this section we state few reductions between CSPs. As the first step we define a new wider structure. Then we show that sets Pol and Inv form a Gallois connection which we use in proof of Lemma B.

The main part of this section form two lemmas about reduction between two CSPs. One lemma is with the new wider structure and the second one is with sets of all polymorphisms.

In the next chapter we will often use these reductions in a way that we will reduce a well known **NP**-complete problem to the given problem. A key text for this section is [3] Jeavons: On the algebraic structure of combinatorial problems.

Definition 1.3.1. Assume a structure $\mathcal{H}$. A new structure $\mathcal{H}^\star$, over the same set H, is defined to be the smallest such that

- $\mathcal{H} \subseteq \mathcal{H}^\star$
- If we permutate elements of a tuple from a relation $R^{\mathcal{H}^\star}$ we must get a tuple from a relation $R^{\mathcal{H}^\star}$. We call this procedure a *permutation*.
- If we put to every n-tuple from a relation $R^{\mathcal{H}^\star}$ any element from the set H to extend them to an $(n+1)$-tuple we will get a relation from $\mathcal{H}^\star$. We call this procedure an *extension*.
- If we delete in every n-tuple from a relation $R^{\mathcal{H}^\star}$ the last element, we will get a relation of $(n-1)$-tuples which is again in $\mathcal{H}^\star$. We call this procedure a *truncation*.
- The itersection of two relations from $\mathcal{H}^\star$ gives a relation from $\mathcal{H}^\star$. We call this procedure an *intersection*.

Remark. For example a projection is also in $\mathcal{H}^\star$ as a combination of truncations and permutations makes a projection.

Lemma. *Any operation from* $Pol(\mathcal{H})$ *is also in* $Pol(\mathcal{H}^\star)$.

Proof. Let us take an operation φ from $Pol(\mathcal{H})$ and a tuple $(a_1, \ldots, a_n)$ from an n-ary relation $R^{\mathcal{H}}$. Let us suppose that $f(a_1, \ldots, a_n) = (b_1, \ldots, b_n)$. If we impose the operation f on a permutated tuple we will get a permutated image which has to be in $R^{\mathcal{H}^\star}$ from the definition. In a similar way if we impose the operation on an extension we will get an extension of the image which has to be again from the definition in the relation $R^{\mathcal{H}^\star}$. We can make similar discussion for other possible situations. □

Lemma (A)**.** *Assume finite structures $\mathcal{H}_1$ and $\mathcal{H}_2$ with the same universe. If $\mathcal{H}_1 \subseteq \mathcal{H}_2^\star$, we can reduce $CSP(\mathcal{H}_1)$ to $CSP(\mathcal{H}_2)$ in logarithmic space.*

Proof. A structure $\mathcal{H}_2^\star$ was generated from $\mathcal{H}_2$. Since a structure $\mathcal{H}_1$ is finite, $\mathcal{H}_1$ was generated from $\mathcal{H}_2$ by a finite sequence of permutation, extension, truncation and intersection. Denoting such a minimal sequence by Σ and assuming a concrete problem $P = (\mathcal{G}_1, \mathcal{H}_1)$ from $CSP(\mathcal{H}_1)$.

We will use an induction to obtain reduction between problems.

1. At first suppose that the length of Σ is zero, i.e. a sequence Σ is empty. Any relation $R^{\mathcal{H}_1}$ is also a relation in $\mathcal{H}_2$ since $\mathcal{H}_1 \subseteq \mathcal{H}_2$. Thus we can surely reduce $CSP(\mathcal{H}_1)$ to $CSP(\mathcal{H}_2)$ in log-space.

2. Suppose that length of Σ is non-zero n. Define Σ' as a sequence of length $n-1$ obtained from a sequence Σ by deleting a final operation. Denote a new relational structure obtained from the structure $\mathcal{H}_2$ by the new sequence Σ' as $\mathcal{H}_1'$. An induction hypothesis gives a log-space reduction from $CSP(\mathcal{H}_1')$ to $CSP(\mathcal{H}_2)$.

   We will find a log-space reduction from $CSP(\mathcal{H}_1)$ to $CSP(\mathcal{H}_1')$. A transitivity property will then give log-space reduction from $CSP(\mathcal{H}_1)$ to $CSP(\mathcal{H}_2)$. To show the first reduction we need to find a problem $P' = (\mathcal{G}_1', \mathcal{H}_1')$ from $CSP(\mathcal{H}_1')$. We will look step by step at all possible final operations.

   - Assume a final operation φ is a permutation. We can write any tuple $(a_1, a_2, \ldots, a_n)$ from $R^{\mathcal{G}_1}$ as $(a_{\varphi(i_1)}, a_{\varphi(i_2)}, \ldots, a_{\varphi(i_n)})$. If we define $R^{\mathcal{G}_1'} = \varphi^{-1} R^{\mathcal{G}_1}$, a problem P' will be a problem from $CSP(\mathcal{H}_1')$.

     We have to show that problems P, P' have the same set of solutions. Let us assume tuples $(a_1, \ldots, a_n)$ from $R^{\mathcal{G}_1}$ which can be rewritten as $(a_{\varphi(i_1)}, \ldots, a_{\varphi(i_n)})$ where $(a_{i1}, \ldots, a_{in})$ is from $R^{\mathcal{G}_1'}$ and a tuple $(b_1, \ldots, b_n)$ from $R^{\mathcal{H}_1}$ which can be rewritten as

$(b_{\varphi(i_1)}, \ldots, b_{\varphi(i_n)})$ where tuple $(b_{i1}, \ldots, b_{in})$ is from $R^{\mathcal{H}'_1}$. Therefore the homomorphism g maps $(a_1, \ldots, a_n)$ to $(b_1, \ldots, b_n)$ if and only if g maps $(a_{i1}, \ldots, a_{in})$ to $(b_{i1}, \ldots, b_{in})$ and thus both problems have the same set of solutions.

- Assume a final operation is an extension. We define a structure $\mathcal{G}'_1$ whose $(n-1)$-ary relation $R^{\mathcal{G}'_1}$ is obtainded from an n-ary relation $R^{\mathcal{G}_1}$ by truncating the last element.

  Suppose that g is a solution to P and $g(a_1, \ldots, a_n)$ is a tuple $(b_1, \ldots, b_n)$ in $R^{\mathcal{H}_1}$. A homomorphism g also solves P' since $g(a_1, \ldots, a_{n-1})$ is a tuple $(b_1, \ldots, b_{n-1})$ in $R^{\mathcal{H}'_1}$. The same is for the other direction.

- Assume a final operation is a truncation. Define for every tuple $(a^i_1, \ldots, a^i_n)$ from $R^{\mathcal{G}_1}$ a new element x^i and define a tuple $(a^i_1, \ldots, a^i_n, x^i) \in R^{\mathcal{G}'_1}$. Problems P, P' have the same set of solutions since $g(a^i_1, \ldots, a^i_n) = (b_1, \ldots, b_n)$ is in $R^{\mathcal{H}_1}$ if and only if $g(a^i_1, \ldots, a^i_n, x^i) = (b_1, \ldots, b_n, y_{n+1})$ is in $R^{\mathcal{H}'_1}$.

- Finally assume an intersection as a final operation. Assume that $R^{\mathcal{H}_1} = R_1^{\mathcal{H}'_1} \cap R_2^{\mathcal{H}'_1}$. Define problem P' to be a problem with relations $R_1^{\mathcal{H}'_1}, R_2^{\mathcal{H}'_1}$ and a structure $\mathcal{G}'_1$ containing twice the relation $R^{\mathcal{G}_1}$.

  If g solves P then g is a homomorphism from any tuple in the relation $R^{\mathcal{G}'_1}$ to a tuple in the relation $R_1^{\mathcal{H}'_1}$ and also to a tuple in the relation $R_2^{\mathcal{H}'_1}$. Therefore a solution to the problem P is a solution to the problem P'.

  Assume that g is a solution to the problem P'. If for any tuple its homomorphic image to $R_1^{\mathcal{H}'_1}$ is different from a homomomorphic image to $R_2^{\mathcal{H}'_1}$, mapping g is not a homomorphism. Thus both homomorphic images have to be the same and lie in $R^{\mathcal{H}_1}$. Therefore a solution to the problem P' is also a solution to the problem P.

□

For other proofs we will use another attribute of operators Pol and Inv called the Galois connection and we will define it directly.

Definition 1.3.2. The *Galois connection* between nonempty sets X, Y is a pair of functions α, β where α maps from the set of all subsets of the set

X to the set of all subsets of the set Y and β maps conversely from the set of all subsets of the set Y to the set of all subsets of the set X and it must satisfy following three conditions:

1. $X_1 \subseteq X_2 \Rightarrow \alpha X_2 \subseteq \alpha X_1$
2. $Y_1 \subseteq Y_2 \Rightarrow \beta Y_2 \subseteq \beta Y_1$
3. $X_1 \subseteq \beta\alpha X_1$ and $Y_1 \subseteq \alpha\beta Y_1$

for all subsets X_1, X_2 of the set X and all subsets Y_1, Y_2 of the set Y.

Lemma. *Mapping Pol and Inv constitute a Galois connection.*

Proof. We will show that Pol and Inv fulfil three conditions from the definition of the Galois connection where as subsets of the set X we will assume structures and as subsets of the set Y we will assume sets of operations.

1. Let us take two relational structures such that $\mathcal{H}_1 \subseteq \mathcal{H}_2$ and assume a function φ from $Pol(\mathcal{H}_2)$. This means that for every relation $R^{\mathcal{H}_2}$ holds $\varphi(R^{\mathcal{H}_2}) \subseteq R^{\mathcal{H}_2}$. However if we apply the operation φ on a relation $R^{\mathcal{H}_1}$ from the structure $\mathcal{H}_1$, $\varphi(R^{\mathcal{H}_1}) \subseteq R^{\mathcal{H}_1}$ since the relation $R^{\mathcal{H}_1}$ is also in the structure $\mathcal{H}_2$. Our relation $R^{\mathcal{H}_1}$ was arbitrary and therefore from the definition of the set $Pol(\mathcal{H}_1)$ the operation φ is also in $Pol(\mathcal{H}_1)$.

2. Let us take two sets of operations such that $Y_1 \subseteq Y_2$. Assume a relation R from $Inv(Y_2)$. If we apply a function φ from the set Y_1 on the relation R we get $\varphi(R) \subseteq R$ since φ is also in the set Y_2 and R is from $Inv(Y_2)$. Therefore $Inv(Y_2) \subseteq Inv(Y_1)$.

3. - Let us take a structure and its relation R. If an operation φ is from the set of polymorphisms, i.e. $\varphi(R) \subseteq R$ then the relation R is in $Inv(Pol(R))$ since $\varphi(R) \subseteq R$ for any operation φ from the set of polymorphisms.
   - Let us take an operation φ and a structure $\mathcal{H}$ from $Inv(\varphi)$. Then the operation φ is also in $Pol(Inv(\varphi))$ since $\varphi(\mathcal{H}) \subseteq \mathcal{H}$ for any relation from $Inv(\varphi)$.

□

Definition 1.3.3. An equality relation on a set D is a binary relation E such that $(a, b) \in E$ if and only if $a = b$.

In the next lemma we will use following remark which we will take for granted. Its proof can be found in [3] Jeavons: On the algebraic structure of combinatorial problems.

Remark. If an equality relation is contained in $\mathcal{H}^\star$ then $Inv(Pol(\mathcal{H}))$ is equal to $\mathcal{H}^\star$.

Lemma (B)**.** *Assume finite structures $\mathcal{H}_1$ and $\mathcal{H}_2$ with the same universe. If $Pol(\mathcal{H}_2) \subseteq Pol(\mathcal{H}_1)$, any problem from $CSP(\mathcal{H}_1)$ is polynomial-time reducible to a problem in $CSP(\mathcal{H}_2)$. With an additional condition of $E \in \mathcal{H}_2^\star$ there is a logarithmic-space reduction from $CSP(\mathcal{H}_1)$ to $CSP(\mathcal{H}_2)$.*

Proof. We will divide our proof into two parts:

1.
   - A set $Pol(\mathcal{H}_2 \cup E)$ is a subset of $Pol(\mathcal{H}_1 \cup E)$ as $Pol(\mathcal{H}_2) \subseteq Pol(\mathcal{H}_1)$. Imposing an operation Inv changes inclusions. We will apply the above remark about equality and get

     $$(\mathcal{H}_2 \cup E)^\star = Inv(Pol(\mathcal{H}_2 \cup E)) \supseteq Inv(Pol(\mathcal{H}_1 \cup E)) = (\mathcal{H}_1 \cup E)^\star.$$

     Combination with sequence $\mathcal{H}_1 \subseteq (\mathcal{H}_1 \cup E) \subseteq (\mathcal{H}_1 \cup E)^\star$ yields $\mathcal{H}_1 \subseteq (\mathcal{H}_2 \cup E)^\star$. Now we can apply a Lemma A about log-space reduction to show that $CSP(\mathcal{H}_1)$ is reducible to $CSP(\mathcal{H}_2 \cup E)$ in log-space.

   - Suppose we have a concrete problem from $CSP(\mathcal{H}_2 \cup E)$ with a given structure $\mathcal{G}$. Look at all relations $R^{\mathcal{G}}, R^{(\mathcal{H}_2 \cup E)}$. If the relation $R^{(\mathcal{H}_2 \cup E)}$ is equal to the equality relation E, the relation $R^{\mathcal{G}}$ is a binary relation.

     Remove every pair (a_1, a_2) from the relation $R^{\mathcal{G}}$ and replace every occurence of the element a_1 in the structure $\mathcal{G}$ with the element a_2. Then remove relations $R^{\mathcal{G}}, R^{(\mathcal{H}_2 \cup E)}$ from structures.

     This polynomial-time procedure yields new structures where the second is equivalent to the structure $\mathcal{H}_2$. A problem with new structures has a solution if and only if the original problem has a solution.

   Impossing these two reductions together makes a polynomial-time reduction from $CSP(\mathcal{H}_1)$ to $CSP(\mathcal{H}_2)$.

2. From the first point $CSP(\mathcal{H}_1)$ is reducible to $CSP(\mathcal{H}_2 \cup E)$ in log-space. Since $\mathcal{H}_2$ is a substructure of $\mathcal{H}_2^\star$ and because of an additional condition, $(\mathcal{H}_2 \cup E) \subseteq \mathcal{H}_2^\star$, from Lemma A about log-space reduction, $CSP(\mathcal{H}_2 \cup E)$ is reducible in log-space to $CSP(\mathcal{H}_2)$.

□

Chapter 2

Tractability and a set of polymorphisms

In the second chapter we go through all posible operations in the set of all polymorphisms and using former lemmas about reduction we investigate if the operation is a sufficient condition for tractability. We use lemma B and well known **NP**-complete problems (NAESAT, GRAPH k-COLORABILITY) for showing **NP**-completness of a given problem via reduction. We also show what kind of operation can be in the set of all polymorphisms if a constraint satisfaction problem is tractable.

In the beginnig of sections we briefly remind definitions of used operations: an essentially unary operation, a constant operation, a majority operation, an idempotent binary operation, an affine operation and a semiprojection. We organize this section in a way that at first we state operations which are sufficient for tractability and later we state the rest of operations. However a semilattice operation will be among the second type of operations because it is a special case of an idempotent binary operation which does not ensure tractability in general.

As a result we get that sufficient operations in the set of all polymorphisms are constant operations, majority operations, semilattices which with additional condition yields to **P**-completeness and affine operations. On the other hand we show that essentially unary operations, binary idempotent operations and semiprojections gives **NP**-completeness.

2.1 Constant operations

At first we will look at constant operations. We will show that constant operations ensures tractability. Key idea is that every relation has to contain a tuple consisting of the same constants. A k-ary constant operation is an operation which maps every k elements to a constant.

This section is based on [3] Jeavons: On the algebraic structure of combinatorial problems and [4] Jeavons, Cohen, Gyssens: Closure properties of Constraints.

Theorem 2.1.1. *A constant operation in $Pol(\mathcal{H})$ is a sufficient condition for tractability of $CSP(\mathcal{H})$.*

Proof. We will divide proof into two sections.

1. If there is no relation in $\mathcal{H}$, any problem from $CSP(\mathcal{H})$ has no solution and thus we can decide about solutions immediatly.

2. If there is a non-empty relation in $\mathcal{H}$, denoting a k-ary constant operation from $Pol(\mathcal{H})$ by φ. Suppose that φ maps to a constant c. Since φ is in $Pol(\mathcal{H})$, a tuple consisting of constant c is in a relation of the structure $\mathcal{H}$. Thus a solution of $CSP(\mathcal{H})$ is a homomorphism which maps any tuple to a tuple of constants c. It can be find in constant-time.

□

2.2 Majority operations and basics from consistency theory

The second type we will look more deeply at is a type of majority operations which also ensures tractability of CSP. Proof of sufficienty is based on enforcing strong consistency and on the decomposition algorithm. You can find it in [4] Jeavons, Cohen, Gyssens: Closure Properties of Constraints. Necessary things from the consistency theory can be found in [8] Dechter: From local to global consistency.

In the beginning of this section we define local, global, i- and strong i-consistency. As a next step we state the decomposition algorithm.

Now we remind what majority operations are. A majority operation is a ternary operation which maps on an element which was twice in the triple.

Let us have a problem P with structures $\mathcal{G}, \mathcal{H}$. Suppose $G = \{g_1, g_2, \ldots, g_l\}$ (we can call g_i a *variable*) and $H = \{h_1, h_2, \ldots, h_k\}$ (we can call h_i a *value*). A number l is a *size* of the problem.

Definition 2.2.1. A *locally consistent* assignment is a partial assignment $(g_1 := h_1, \ldots, g_i := h_i)$ which satisfies restriction of the problem to the set $\{g_1, \ldots, g_i\}$. The restricted problem $P_{|\{g_1, g_2, \ldots, g_i\}}$ with locally consistent assignment is called *locally consistent problem*.

Remark. Above restriction can be done by projection. In that case we will get new relations of different arity.

To restrict the problem we will go through every tuple from every relation from $\mathcal{G}$ and project this tuple to tuple containing only elements from the required set. To get a corresponding relation we have to apply the same projection on the corresponding relation from $\mathcal{H}$. We will later inlustrate it on an example.

Definition 2.2.2. Suppose that $X \subseteq Y \subseteq G$. A *relative consistent* problem $P_{|X}$ to a problem $P_{|Y}$ is a locally consistent problem $P_{|X}$ whose solution can be extended to a solution of a locally consistent problem $P_{|Y}$.

Definition 2.2.3. Suppose that $X \subseteq G$. A problem $P_{|X}$ is *globally consistent* if $P_{|X}$ is relative consistent to P. A problem P is *globally consistent* if every subproblem $P_{|X}$ is globally consistent.

Definition 2.2.4. A problem is *i-consistent* if for every subproblem of size $i-1$ any solution can be consistently extended to a problem of size i by adding any ith variable. A problem is *strong i-consistent* if it is j-consistent for every $j = 1, 2, \ldots, i$.

Example. *We will ilustrate consistency on problem* $P = (\mathcal{G}, \mathcal{H})$ *where*
$G = \{a, b, c, d\}$,
$H = \{1, 2, 3\}$,
$R_1^{\mathcal{G}} = \{(aba), (baa), (cda)\}$,
$R_2^{\mathcal{G}} = \{(abab), (babc), (cdab)\}$,
$R_1^{\mathcal{H}} = \{(112), (123), (231), (233), (323)\}$,
$R_2^{\mathcal{H}} = \{(1212), (2131), (1132), (2321), (3232)\}$.
We would like to find a locally consistent assignment for set $\{a, b\}$.

We have two possibilities how to do it. First one is via projections. If we make a projection we will get new corresponding relations from relations R_1*:*

- $\{(aba), (baa)\}$ *with* $\{(112), (123), (231), (233), (323)\}$ *and*
- $\{(a)\}$ *with* $\{(2), (3), (1)\}$.

From relations R_2 *we get:*

- $\{(abab)\}$ *with corresponding* $\{(1212), (2132), (2132), (2321), (3232)\}$,
- $\{(bab)\}$ *with* $\{(121), (213), (113), (232), (323)\}$ *and*
- $\{(ab)\}$ *with corresponding relation* $\{(12), (31), (32), (21)\}$.

Now we have a new problem and its solutions are locally consistent to the original one.

Other way how to find a proper assignment is via guessing. If we assign 1 *to* a *we cannot find a corresponding triple to* (aba) *as in the relation* $R_1^{\mathcal{H}}$ *is no triple* $(1?1)$. *In a similar fashion we cannot assign* 2 *to* a. *If we assign* 3 *to* a *we can find to* (aba) *only triple* (323). *Thus* b *has to be* 2.

Now we have to check if this assignment satisfies all tuples. For (baa) *we need* (233) *which is in the relation. For* (cda) *we can take* $(123), (233)$ *or* (323), *for* $(abab)$ *we have to take* (3232) *which is again in the relation. For* $(babc)$ *we can only take* (2321), *for* $(cdab)$ *we can take* (1132) *or* (3232). *We do not have to take care of values of variables* c, d.

Thus the locally consistent assignment restricted to the set $\{a, b\}$ *is* $a := 3$, $b := 2$.

Now we will look at relative consistency of problem $P_{|\{a,b\}}$ *to problem* $P_{|\{a,b,c\}}$. *We know that first problem is locally consistent as* $a := 3, b := 2$ *is a locally consistent assignment. Now we would like to extend it. From the above observation we know that to the tuple* $(babc)$ *we can only assign* (2321) *and thus the only possibility is* $a := 3, b := 2$ *and* $c := 1$.

From the above observation we know that there are only three tuples containing variable c. *Exactly* (cda) *to which we can assign* (123), *to* $(babc)$ *we can assign* (2321) *and to* $(cdab)$ *we can assign* (1132). *Thus we have found a localy consistent assignment for variables* a, b, c *and thus problem* $P_{|\{a,b,c\}}$ *is locally consistent and therefore from the definition* $P_{|\{a,b\}}$ *is relative consistent to* $P_{|\{a,b,c\}}$.

What about global consistency of $P_{|\{a,b\}}$? *From the above observation we know that the only consistent possibility is* $a := 3, b := 2$ *extended with* $c := 1$

which is again the only one possibility. Tuples with variable d are (cda) *and* $(cdab)$*. From the above text we know that* (cda) *can be only* (123) *and* $(cdab)$ *can be only* (1132) *and thus from the first one* $d := 2$ *and from the second one* $d := 1$*. Thus we could not consistently extend locally consistent problem* $P_{|\{a,b\}}$ *to the whole problem* P*.*

Therefore $P_{|\{a,b\}}$ *is not globally consistent and also* P *is not globally consistent.*

Now we will look at i*-consistency, exactly at* 2*-consistency. All subproblems of size* 1 *are* $P_{|\{a\}}, P_{|\{b\}}, P_{|\{c\}}, P_{|\{d\}}$*. We would like to extend any solution of* $P_{|\{a\}}$ *to consistent solution of* $P_{|\{a,b\}}, P_{|\{a,c\}}, P_{|\{a,d\}}$*. From the triple* (aba) *we can see that only possibility for* a *is* $a := 3$ *which is a consistent assignment from the above observations. Consistent extensions are* $a := 3, b := 2$*;* $a := 3, c := 1$*;* $a := 3, d := 2$*.*

Now we will turn our attention to restriction to b*. The value of* b *can be* 1 *or* 2*. Consistent extensions to* $P_{|\{a,b\}}$ *are* $a := 3, b := 2$ *however for* $b := 1$ *there is no consistent extension.*

Therefore the problem is not 2*-consistent.*

Remark. If the problem P is strong $(k+1)$-consistent then it is globally consistent and thus tractable.

Lemma 2.2.5. *A problem* $P = (\mathcal{G}, \mathcal{H})$ *which has only binary relations and is strong* $|H| + 1$*-consistent is globally consistent.*

Proof. We will show that P is $(k+1+i)$-consistent for any $i > 0$. Assume we have a subset $S = \{g_1, g_2, \ldots, g_{k+i}\}$ of some $k+i$ variables from the set G. We have a non-empty set of locally consistent assignments $A = \{g_1 := h_1, g_2 := h_2, \ldots, g_{k+i} := h_{k+i}\}$ where all values are from the set H and they do not have to be distinct. To make it $(k+i+1)$-consistent we have to show that there is a consistent extension for any varibale g.

If h is a possible assignment to the variable g we will denote A_h a subset of all locally consistent assignements from the set A which can be consistently enxtended with assignment $g := h$. Maximum number of such subsets is $|H|$ as the variable g can takes values only from the set H.

For a contradiction let us assume that every subset A_h misses some member, i.e. denote the missing value h'. If we make a new tuple consisting of missing memebers from each subset A_h we will get a tuple which is not consistent with any assignment $g := h$ as something is missing.

However as the tuple of missing members is locally consistent subset, from the definition of strong consistency there should be a consistent extension with variable g. However we know that this is impossible and thus we received a contradiction.

Therefore at least one subset A_h has to contain our assignment to variables $g_1, \ldots, g_{k+i}$. In the last step we will assume without loss of generality that every asignment $g_1 := h_1, \ldots, g_{k+i} := h_{k+i}$ is consistent with assignment $g := h$ and therefore we have found a value consistent with $g_1 := h_1, \ldots, g_{k+i} := h_{k+i}$. □

However after enforcing consistency new non binary relations may occur. An algorithm for enforcing strong consistency is in the appendix.

Next algorithm states that a new problem with relations decomposed to binary ones has the same set of solutions.

Algorithm (decomposition)**.** *Let us have a particular problem P with relational structures $\mathcal{G}, \mathcal{H}$ where $Pol(\mathcal{H})$ contains a majority operation.*

Replace every relation $R^{\mathcal{G}}$ and $R^{\mathcal{H}}$ with binary relations made by binary projections π_{ij} for every $i \leq j$. A new problem P' has the same solution.

Proof. Denote a new instance by $\mathcal{G}'$ and a new template by $\mathcal{H}'$. At first we will show that a solution of P is also a solution to P'.

If f is a solution of P, a tuple $(a_1, a_2, \ldots, a_n)$ from $R^{\mathcal{G}}$ is mapped by f to a tuple $(b_1, b_2, \ldots, b_n)$ in $R^{\mathcal{H}}$. Consequently a couple (a_i, a_j) from $R^{\mathcal{G}'}$ is mapped by f to a couple (b_i, b_j) in $R^{\mathcal{H}'}$ and f is a solution of P'.

For a second direction suppose that φ is a solution of the problem P'. Let us take a tuple $(a_1, a_2, \ldots, a_n)$ from $R^{\mathcal{G}}$ and denote
$b := (\varphi(a_1), \varphi(a_2), \ldots, \varphi(a_n))$. We need to show that $b \in R^{\mathcal{H}}$. A usefull tool will be an induction.

1. $n = 2$: As $R^{\mathcal{G}} = R^{\mathcal{G}'}$ and $R^{\mathcal{H}} = R^{\mathcal{H}'}$, φ is a solution of P.

2. $n > 2$: Let $I = \{1, 2, \ldots, n\}$. Let us take three indices i_1, i_2, i_3 from I and apply a projection π_{I-i_j} on the relation $R^{\mathcal{H}}$. Results are $(n-1)$-ary relations R_j from $(R^{\mathcal{H}})^{\star}$. However $Pol(R_j)$ contains also a majority operation and thus by induction, the way of getting R_j there is a tuple d_j which has the same projection as b except position i_j. Applying the majority operation on d_1, d_2, d_3 yields b from $R^{\mathcal{H}}$.

□

Theorem 2.2.6. *A majority operation in $Pol(\mathcal{H})$ is a sufficient condition for tractability of $CSP(\mathcal{H})$.*

Proof. Let us have a particular problem $\mathcal{P}$ with an instance $\mathcal{G}$ and a template $\mathcal{H}$. Enforcing strong $|H|+1$ consistency (in polynomial time) does not change a set of solutions. New relations are elements of $\mathcal{H}^\star$. We know that $Pol(\mathcal{H}^\star)$ also contains the same majority operation. From the above algorithm the new problem can be decomposed in polynomial time into a problem with binary relations and with the same set of solutions. From the strong $|H|+1$ consistency we know that the problem is globally consistent and thus tractable according to the remark. □

2.3 Affine operations

The third type of operations we look at is affine one. The main result of this section is that affine operations are sufficient for tractability. Its proof is in [4] Jeavons, Cohen, Gyssens: Closure Properties of Constraints, [5] Feder, Vardi: The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study Through Datalog nad Group Theory, [11] Furst, Hopcroft, Luks: Polynomial-time algorithms for permutation groups, [12] Hoffmann: Group-Theoretic Algorithms and Graph Isomorphism.

At first we define a representation matrix, later we state the representatives algorithm which finds coset representatives for a chain of permutation groups. Actually we will show that it finds all coset representatives in polynomial time. As an interstep we use the Sift algorithm which inserts at most one new coset representative to the reprentation matrix which we use for storing coset representatives. These things builds sufficiency of affine operations for tractability of CSP.

Now it is time to remind what affine operations are. A ternary operation φ such that $\varphi(a,b,c) = a - b + c$ where $(D,+,-)$ is an Abelian group and $a,b,c \in D$ is an affine operation.

Let G be a permutation group on $\{1,2,\ldots,n\}$ and define groups G_i of permutations such that G_i fixes $1,2,\ldots,i$ for $i \leq n$. Thus we get a descending chain $G \supset G_1 \supset G_2 \supset \ldots \supset G_n = I$ where I is an identity. Below we will always assume G and G_i as has just been defined. A permutation group G can be viewed as G_0.

Definition 2.3.1. Table T of size $n \times n$ is called a *representation matrix* if the next conditions hold:

- rows are labelled $0, 1, \ldots, n-1$ and columns $1, 2, \ldots, n$
- in a row i are coset representatives for G_i/G_{i+1}
- in a position (i, j) is a representative which fixes $1, \ldots, i$ (except row 0 which does not fix anything) and moves $i+1$ to j
- on the main diagonal are identity permutations
- elsewhere are empty elements

For an easier orientation we will denote rows of the representation matrix as u_i.

Remark. From the definition of a representation matrix follows that a representation matrix is empty bellow the main diagonal. Let us look at a coset G_i/G_{i+1}. It moves $i+1$ which is a reason why $|G_i/G_{i+1}| \leq n$ and consequently the representation matrix has n columns.

Definition 2.3.2. A *canonical representation* of an element from a permutation group is an expression $g_0 g_1 g_2 \ldots g_{n-1}$ where g_i is from u_i in a corresponding representation matrix.

Lemma 2.3.3. *Let M be a set of all permutations expressible in canonical representation. The set M creates a group if and only if for every pair of permutations φ, ψ from the representation matrix of size $n \times n$, their product $\varphi\psi$ can be expressed in canonical representation, i.e. their product is in the set M.*

Proof. At first we will show that M is a group. Suppose that φ, ψ are two elements from M. We will show by induction that their product is also in M.

1. If $n = 1$, there is only one permutation in representation matrix. This permutation has to be an identity from the definition of a representation matrix and only identity permutation can be expressed in a canonical representation. Thus M contains only identity permutations. The product of identities is again the identity and lies in M.

2. Now let us take a subset M' of permutations from M which fixes 1 and let us transform the representation matrix by deleting row u_0 and first column to obtain a new matrix T'. The subset M' can be associated in a similar way with T' as the set M was with the representation matrix.

   Assume two permutations from T' and make their product. Since both are also from the representation matrix, their product is in M. However both of them fix 1 and therefore also their product fixes 1. Consequently the product of two permutations from T' is in M'. By induction hypothesis M' is a group.

   We will show that M is product closed. Let us take two permutations φ, π from M. They can be written in canonical representation:

   $$\varphi\pi = (a_0 a_1 \dots a_{n-1})(b_0 b_1 \dots b_{n-1}) = (a_0 \dots a_{n-2})(a_{n-1} b_0)(b_1 \dots b_{n-1}).$$

   Permutations a_{n-1} and b_0 are from the representation matrix and we can write their product in canonical representation:

   $$(a_0 \dots a_{n-2})(c_0 \dots c_{n-1})(b_1 \dots b_{n-1}) =$$

   $$(a_0 \dots a_{n-2})(c_0 \dots c_{n-2})c_{n-1}(b_1 \dots b_{n-1}).$$

   Now we will use the induction hypothesis to get:

   $$(d_0 \dots d_{n-2})c_{n-1}(b_1 \dots b_{n-1}).$$

   We can proceed in a similiar way until we get

   $$(y_0 \dots y_{n-2})z_{n-1} = (y_0 y_1 \dots y_{n-2} z_{n-1})$$

   which is in canonical representation and thus the product of two permutations from M is in M.

The identity is in M and the same is with an inversion. Therefore M creates a group.

For the second direction suppose that M creates a group and the permutation φ is from the representation matrix. The permutation φ can be expressed as $\varphi = I \circ I \circ \dots \circ \varphi \circ I \circ \dots \circ I$ which is canonical representation and thus φ is in M. However M creates a group. Thus the product of two representatives from the representation matrix lies in M. □

We will now define an algorithmus $Sift$.

Algorithm (Sift(x)). *Input: permutation x*

- $i := 0$
- *while* $i \neq n-1$ *and* $\exists y \in u_i$ *such that* $x(i+1) = y(i+1)$ *do*
  1. $i := i + 1$
  2. $x := y^{-1}x$
- *if* $x \notin u_i$ *insert* x *in* u_i *such that definition of the representation matrix us fullfilled.*

This algorithm inserts at most one new representative to a representation matrix. Note that x, y are permutations.

Lemma. *All of the coset representatives are in the representation matrix.*

Proof. Define a new matrix N which we obtain from the representation matrix by changing every element above the diagonal in a row u_0 to an empty element. We will show that N defines the subgroup G_1 which fixes 1. Denote H as a set of all permutations which can be formed as in canonical representation with matrix N. Our goal is to show that H is the same as G_1.

Let us take two representatives from N and make their product. Since they are also contained in the representation matrix and G is a group thus from the lemma 2.3.3 about group and representation matrix, their product is in G. However the product also fixes 1 and thus its canonical representation will have the identitity permutation in the place of g_0 which means its canonical representation is with elements from N. Thus its product is in H and H is a subgroup of G. We know that it fixes 1 thus it is in addition a subgroup of G_1.

Suppose we have a permutation $\varphi \in G_1 \subseteq G$ which can be expressed in a canonical representation $a_0a_1 \dots a_{n-1}$. Since it fixes 1, $a_0 = id$. Therefore this canonical representation is obtained from elements of N which means that φ is in H. Hence $H = G_1$.

Suppose we have an element g from G. Thus it can be written in canonical representation $a_0a_1 \dots a_{n-1}$ where $a_1 \dots a_{n-1}$ is a canonical representation of some element h from G_1. Thus $g = a_0h$ and we have all coset representatives. □

Lemma. *Let* G *be a permutation group generated by* $g_1, g_2, \dots, g_k$. *Coset representatives for* G_i/G_{i+1} *can be found in polynomial time.*

Proof. A cornerstone of the proof is the *Representatives algorithm*.

Algorithm (Representatives)**.**

1. *Initialize representative matrix with identities on the main diagonal and empty elsewhere.*

2. *Run Sift algorithm on every generator g_i.*

3. *Run Sift algorithm on a product xy for every pair x, y from a representative matrix to close it in a way that xy can be written in a canonical representation.*

A combination of the above lemmas shows that the Representatives algorithm finds all the coset representatives. Let us turn our attention to complexity of the Representatives algorithm.

Number of coset representatives is at most n^2 since the size of the representation matrix is n^2. To run Sift on a product of every pair we need at most $n^2 \cdot n^2$ calls of Sift. We have k generators so all together we have to run Sift at most $(n^4 + k)$ times.

Each Sift takes roughly n^2 since we can go through the whole matrix. Thus it takes $O(kn^2 + n^6)$ steps to find all the coset representatives which is polynomial time. □

Example. *We will show how the Representative algorithm works. We will assume a symetric group on* $\{1, 2, 3, 4\}$ *as* $G = G_0$*, i.e. group containing* $\{I, (12), (13), (14), (23), (24), (34), (12)(34), (13)(24), (14)(23), (123), (132), (124), (142), (134), (143), (234), (243), (1234), (1243), (1324), (1342), (1423), (1432)\}$*. Then from the definition* $G_1 = \{I, (23), (24), (34), (234), (243)\}$*,* $G_2 = \{I, (34)\}$*,* $G_3 = \{I\}$*,* $G_4 = \{I\}$*.*

At first we need generators of G *to start the algorithm. For a symetric group we can take as generators transposition* (12) *and a cyclic shift* (1234)*. We will not show it since it is long and not interesting for our example.*

Now we can start. At first we initialize the representation matrix:

	1	*2*	*3*	*4*
u_0	*id*			
u_1		*id*		
u_2			*id*	
u_3				*id*

We start with permutation (1234) *and continue with the second generator* (12).

Sift((1234))):
i=0, the permutation (1234) *moves* $i+1=1$ *to* 2. *As in the row* u_0 *is no permutation which moves* 1 *to* 2 *we skip the whole while-cyclus and look at if-condition. We insert the permutation to the representation matxic and get:*

	1	*2*	*3*	*4*
u_0	*id*	*(1234)*		
u_1		*id*		
u_2			*id*	
u_3				*id*

Sift((12)):
i=0, the permutation (12) *moves* 1 *to* 2. *In the row* u_o *there is permutation* (1234) *which also moves* 1 *to* 2 *thus we have to run the while-cyclus.*
$i := 1, x := (1234)^{-1} \circ (12) = (243)$. *Permutation* (243) *moves* 2 *to* 4 *and in the row* u_1 *there is no permutation which moves* 2 *to* 4 *and thus we skip off the while-cyclus. The if-condition tells us to insert* (243) *in the row* u_1. *We insert the permutation in column* 4 *to fullfill the definition of the representation matrix. We get:*

	1	*2*	*3*	*4*
u_0	*id*	*(1234)*		
u_1		*id*		*(243)*
u_2			*id*	
u_3				*id*

Now we have to run shift on every pair in the representation matrix. We will start with $x = (243) \circ (1234) = (14)$.

Sift((14))):
$i = 0$, *the permutation* (14) *moves* 1 *to* 4. *As the condition for the while-cyclus is not fullfilled we immediately skip to the if-condition and insert the permutation into the representation matrix and get:*

	1	*2*	*3*	*4*
u_0	*id*	*(1234)*		*(14)*
u_1		*id*		*(243)*
u_2			*id*	
u_3				*id*

As the next we take $x = (1234) \circ (14) = (234)$.

Sift((234)):
$i = 0$, *the permutation* (234) *fixes* 1. *In the row* u_0 *also the identity permutation fixes* 1 *and thus we have to run the while-cyclus:* $i = 1$, $x = id \circ (234) = (234)$. *In the row* u_1 *is no permutation moving* 2 *to* 3 *as the permutation* (234) *do. Thus we do not continue in while-cyclus and move to the if-condition which tells us to modify the representation table:*

	1	*2*	*3*	*4*
u_0	*id*	*(1234)*		*(14)*
u_1		*id*	*(234)*	*(243)*
u_2			*id*	
u_3				*id*

The next permutation is $x = (14) \circ (1234) = (123)$.

Sift((123)):
$i = 0$, (123) *moves* 1 *to* 2 *as in the row* u_0 *also the permutation* (1234) *does and thus we have to run the while-cyclus:* $i = 1$, $x = (1234)^{-1} \circ (123) = (34)$. *Permutation* (34) *fixes* 2 *as in the row* u_1 *the identity permutation does and therefore we continue in while-cyclus:* $i = 2$, $x = id \circ (34) = (34)$. *Now we skip off the cyclus and move to the if-condition and get:*

	1	*2*	*3*	*4*
u_0	*id*	*(1234)*		*(14)*
u_1		*id*	*(234)*	*(243)*
u_2			*id*	*(34)*
u_3				*id*

Now we miss just one permutation moving 1 *to* 3. *Such a permutation we can get as* $x = (1234) \circ (1234) = (13)(24)$.

Sift((13)(24)):
$i = 0$, *we skip the while-cyclus and from the if-condition we get:*

	1	*2*	*3*	*4*
u_0	*id*	*(1234)*	*(13)(24)*	*(14)*
u_1		*id*	*(234)*	*(243)*
u_2			*id*	*(34)*
u_3				*id*

The representation matrix is now finnished. We had not go through all pairs as the Representative algorithm tells but the matrix is full and thus we cannot put anything else into it and thus we do not have to run for the rest of pairs. We will show that these permutations are really coset representatives. We will start with the last line of the representation matrix and G_3*. The representation matrix tells us that* $G_3 = IG_4$ *which is true since* $G_4 = \{I\}$*.*

The representation matrix tells $G_2 = IG_3 \cup (34)G_3$*. We know that* $G_3 = \{I\}$ *and thus we get* $G_2 = \{I\} \cup \{(34)\} = \{I, (34)\}$ *which corresponds to* G_2 *from the beginning of the example.*

The representation matrix also tells that $G_1 = IG_2 \cup (234)G_2 \cup (243)G_2$*. As* $G_2 = \{I, (34)\}$ *we get* $G_1 = \{I, (34)\} \cup \{((234), (23)\} \cup \{(243), (24)\} = \{I, (34), (23), (24), (234), (243)\}$ *which corresponds to* G_1 *from the beginning of the example.*

Finally we look at $G = IG_1 \cup (1234)G_1 \cup (13)(24)G_1 \cup (14)G_1$*. As* $G_1 = \{I, (34), (23), (24), (234), (243)\}$ *we get:*
$G = \{I, (34), (23), (24), (234), (243)\} \cup \{(1234), (123), (124), (12)(34),$
$(1243), (12)\} \cup \{(13)(24), (1324), (1342), (13), (132), (134)\} \cup \{(14), (143),$
$(14)(23), (142), (1423), (1432)\}$ *which corresponds to* $S_4 = G$ *from the beginning of the example.*

We have found all the coset representatives and have showed how the algorithm works.

Lemma 2.3.4. *Let* $\mathcal{H}$ *be a structure whose relations are cosets of subgroups of* H^k*. Then* $CSP(\mathcal{H})$ *is polynomially solvable.*

Proof. Assume a concrete problem $P = (\mathcal{G}, \mathcal{H})$ where relations in $\mathcal{G}$ are defined on n elements. We are looking for a solution which will assign to an element from G an element from H.

At first let us order elements in G. A solution will be an element of $\bar{H} = H^n$. Relations form a coset $a_i J_i$ for some subgroup J_i of $\bar{H}$. Define $H_i := J_1 \cap J_2 \cap \ldots \cap J_i$ for $i \leq s$ where s is an amount of relations. Then fix components to 1 to obtain $H_r = H_{s+n} = \{1\}$.

Now we have a descending chain of groups and we can obtain coset representatives in polynomial time with the above Representatives algorithm.

A relation a_1J_1 is a coset of $\bar{H}/H_1 = \bar{H}/J_1$ whose representatives can be found in row u_0 in the representative matrix. We will denote it a. Suppose we have a solution $v \in \bar{H}$ where $\bar{H}$ can be viewed as a disjoint union of cosets a_iH_i. If our solution v is in aH_1, it has to be in the form ax where x is in H_1.

Condition $a_jx \in a_iJ_i$ is equivalent to $x \in a_j^{-1}a_iJ_i = a_lJ_i$. A solution has to hold for every relation.

Suppose our solution v is from a_2J_2. From the representation matrix we know representatives for H_1/H_2. Every $v \in \bar{H}$ can be expressed as abh where a is a representative of H/H_1, b of H_1/H_2 and h is from H_2. Thus bh is in $a^{-1}a_2J_2$ which gives bh is in b^2J^2. However $h \in H_2 \subseteq J_2$ and thus b is in b^2J_2. If there is no represantive b such that b is from b_2J_2 there is no solution for the problem.

Similar process can be taken for $H_3, H_4, \ldots, H_s$. This algorithm is polynomial since everything is polynomially bounded. □

Theorem 2.3.5. *An affine operation in $Pol(\mathcal{H})$ is a sufficient condition for tractability of $CSP(\mathcal{H})$.*

Proof. Suppose φ is an affine operation from $Pol(\mathcal{H})$ and assume a relation $R^{\mathcal{H}}$. Assume an abelian group with the following property : $(a_1 - a_2 + a_3)$ in the relation $R^{\mathcal{H}}$ for any a_1, a_2, a_3 from $R^{\mathcal{H}}$.

Define a set $S := \{a_1 - a_2; a_1, a_2 \in R^{\mathcal{H}}\}$. We will show that S is a subgroup of the relation $R^{\mathcal{H}}$. Take two elements $s_1 = a_1 - a_2$ and $s_2 = a_1' - a_2'$. A difference $s_1 - s_2 = a_1 - a_2 + a_2' - a_1'$ where $a_1 - a_2 + a_2'$ is an element in $R^{\mathcal{H}}$ and thus also $s_1 - s_2$ is an element of S.

Now assume an element s from S and a tuple a_3 from $R^{\mathcal{H}}$. Then $s + a_3 = (a_1 - a_2) + a_3$ which is a condition of an affine operation and thus $s + a_3 \in R^{\mathcal{H}}$. This is a reason why $R^{\mathcal{H}}$ is a coset of the direct product of an abelian group with an affine property.

We can now impose the lemma 2.3.4 about tractability for a structure with cosets of subgroups as relations. □

2.4 Idempotent binary operations

Another type of operations we investigate is a type of idempotent binary operations. At first we show that in general an idempotent operation does not ensure tractability. You can find it in [4] Jeavons, Cohen, Gyssens: Closure

Properties of Constraints. Key idea is to find an example which we can reduce in polynomial time to a well known **NP**-complete problem. We will use NAESAT which we know is **NP**-complete.

The second part of this section is about a special case of idempotent binary operations - semilattice operations. We will show that semilattice operations are sufficient for tractability. For proof we will need the pairwise consistency algorithm which works in polynomial time and does not change the set of solutions. As the first step of the proof we will enforce pairwise consistency then we will find all possible values for variables via projection and with use of projections we will find a solution to a new pairwise consistent problem and thus also to the given problem in polynomial time. Proof can is in [4] Jeavons, Cohen, Gyssens:Closure Properties of Constraints and [13] Jeavons, Cohen, Gyssens: A Unifying Framework for Tractable Constraints.

In the last part we will show that with an additional condition semillatice operation ensures **P**-completeness. In the proof we will define a new structure and a corresponding problem which will match to HORN-SAT. With use of Gallois connection and Lemma B about reduction we will reduce HORN-SAT to the given problem. As HORN-SAT is known to be **P**-complete the given problem will be also **P**-complete. You can find it in [3] Jeavons: On the algebraic structure of combinatorial problems.

Now we remind what an idempotent binary operation is. A binary operation which applied on two same elements gives again this element is called an idempotent binary operation if it is not a projection. We will later define a semilattice operation which is an asociative commutative idempotent binary operation.

Theorem 2.4.1. *An idempotent binary operation, different from a projection, in $Pol(\mathcal{H})$ yields* **NP***-completeness of $CSP(\mathcal{H})$.*

Proof. We need to find an **NP**-complete example. Assume a set $H = \{0, 1, 2, 3\}$ and an idempotent binary operation φ given by

φ	0	1	2	3
0	0	1	0	1
1	0	1	0	1
2	2	3	2	3
3	2	3	2	3

.

We can easily see that φ is not a projection because $\varphi(2, 1) = 3$. Now put our stress on a function b_1, respectively b_2, which returns the first, respectively

the second, bit in a binary expression of elements of a set H. Define a relation $R \subseteq H^3$ where exactly two elements in 3-tuple have the same first bit and exactly two elements have the same second bit in its binary expression.

We will show that then φ is in $Pol(H, R)$. Divide elements of the set H into two groups along first bit in the binary expression. The operation $\varphi(a_i, a_j)$ returns an element from a group which contains a_i. Denote this group by A_i. Now put the operation on two 3-tuples $((a_1, a_2, a_3), (b_1, b_2, b_3))$. Applying the operation φ yields a tuple (A_1, A_2, A_3). We know that exactly two elements of the tuple (a_1, a_2, a_3) are in the same group. Thus also exactly two elements from (A_1, A_2, A_3) are in the same group. For the second bit enforce the same procedure.

We can reduce $CSP(\mathcal{H})$ to the NOT-ALL-EQUAL SATISFIABILITY problem, which is known to be **NP**-complete, in polynomial time. □

Therefore we can see that in general an idempotent binary operation is not enough for the tractability of the constraint satisfaction problem. However there is a special type of idempotent binary operation which is sufficient for tractability.

Definition 2.4.2. A binary operation φ on a set D is called an *AC operation* if it is associative, i.e. $\varphi(\varphi(a_1, a_2), a_3) = \varphi(a_1, \varphi(a_2, a_3)) \forall a_1, a_2, a_3 \in D$, and commutative, i.e. $\varphi(a_1, a_2) = \varphi(a_2, a_1) \forall a_1, a_2 \in D$. An operation is called a *semilattice* if it is an AC operation and all together idempotent.

Remark. Every semilattice operation gives rive to a partiall ordering of a set. Supose we have a semilattice operation φ of a set H. Then H is ordered by $\leq$ such that $a \leq b \Leftrightarrow \varphi(a, b) = b$. It can be also viewed conversely, the partial ordering can induce a semilattice.

We will in addition define another type of consistency called pairwise consistency. Suppose we have a concrete problem $P = (\mathcal{G}, \mathcal{H})$.

In the next two definitions we assume an n-tuple a from a relation $R_n^{\mathcal{G}}$ and a k-tuple b from a relation $R_k^{\mathcal{G}}$. Suppose that $a[i] = b[1], a[i+1] = b[2], \ldots, a[n] = b[n-i+1]$ for some $i \leq n$ where $a[i]$ means the ith coordinate in a. We write $t = (\varphi(a[1]), \ldots, \varphi(a[i]), \varphi(b[2]), \ldots, \varphi(b[k]))$ where φ is an assignment which maps some variables from G to values in H.

Definition 2.4.3. Associated relations $R_n^{\mathcal{H}}, R_k^{\mathcal{H}}$ are *pairwise consistent* if assigning an element from $R_n^{\mathcal{H}}$ to any element from $R_n^{\mathcal{G}}$ can be extended to a tuple t in a way that $\pi_{i,i+1,\ldots,i+k-1}(t)$ is in $R_k^{\mathcal{H}}$. Same thing has to hold for an element from $R_k^{\mathcal{G}}$.

Definition 2.4.4. *Join* of two relations $R_n^{\mathcal{H}}$,$R_k^{\mathcal{H}}$, denoted by $R_n^{\mathcal{H}} \bowtie R_k^{\mathcal{H}}$, is a set of tuples t satisfying that $\pi_{1,\dots,n}(t) \in R_n^{\mathcal{H}}$ and $\pi_{i,\dots,i+k-1}(t) \in R_k^{\mathcal{H}}$ for some i.

Now we can define an algorithm for establishing pairwise consistency.

Algorithm (Pairwise consistency algorithm)**.**

- *Make a join of any two corresponding relations.*
- *Make a projection of the join to the original tuples.*

Repeat this procedure until there is no further change in relations.

Remark. A parwiese consistency algorithm works in polynomial time and does not change a set of solutions.

Example. *We will show an example to ilustrate how the pairwise consistency algorithm works.*

Assume relations

- $R_1^{\mathcal{G}} = \{(aba), (baa), (cda)\}$,
- $R_2^{\mathcal{G}} = \{(abab), (babc), (cdab)\}$,
- $R_1^{\mathcal{H}} = \{(123), (112), (231), (233)\}$,
- $R_2^{\mathcal{H}} = \{(1212), (2131), (2223), (1221)\}$.

At first we will look at relations $R_1^{\mathcal{G}}$ and $R_2^{\mathcal{G}}$ and make their join. We will get $(abab), (ababab), (ababc), (baabab), (cdabab), (cdab)$ which projecting down to the original scopes gives $(aba), (baa), (cda)$ and $(abab), (babc), (cdab)$ and thus relations did not change.

Secondly we make a join of $R_1^{\mathcal{H}}$ and $R_2^{\mathcal{H}}$. We get tuples: $(11212), (112131), (112223), (11221), (231212), (231221)$. *If we project them down to the original ones we get triples* $(112), (231)$ *and* 4*-tuples* $(1212), (2131), (2223)(1221)$ *and as we can see we missed tuples* $(123), (233)$. *The relation $R_1^{\mathcal{H}}$ changed to* $\{(112), (231)\}$. *Now we should again run the pairwise consistency algorithm.*

We will again make a join of relations $R_1^{\mathcal{H}}$ and $R_2^{\mathcal{H}}$. We get: $(11212), (112131), (112223), (11221), (231212), (231221)$ *which projecting down to the original ones give triples* $(112), (231)$ *and* 4*-tuples* $(1212), (2131), (2223), (1221)$ *and thus the relations did not change anymore.*

Now we have to make a join of $R_2^{\mathcal{H}}$ and $R_1^{\mathcal{H}}$. We get: $(121231), (213112), (22231), (122112)$ *which projecting down to the original ones gives* 4*-tuples* $(1212), (2131), (2223), (1221)$ *and triples* $(231), (112)$ *and thus the relations have not changed anymore and they are pairwise consistent.*

Theorem 2.4.5. *A semilattice operation φ in $Pol(\mathcal{H})$ is a sufficient condition for tractability of $CSP(\mathcal{H})$.*

Proof. Assume a concrete problem $P = (\mathcal{G}, \mathcal{H})$. Enforce a pairwise consistency algorithm and obtain a new problem $P' = (G', H')$. From the pairwise algorithm holds $H' \subseteq (\mathcal{H}^\star)$. We know that φ is in $Pol(\mathcal{H}^\star)$ and thus φ is in $Pol(\mathcal{H}')$.

Let us take a variable g from G and denote by $D(g)$ all possible values from H which does not violate relations in P'. A simple method for finding $D(g)$ is a projection of possible tuples in relations. Since relations obtained via projection are in $\mathcal{H}^\star$, φ is in $Pol(D(g))$.

We have to distinguish between two situations:

1. If $D(g)$ is empty for any g from G, the problem P' does not have any solution.

2. Suppose that $D(g)$ is non-empty for any variable g from G. Let us take an n-tuple a from $R^{\mathcal{G}'}$ and associated relation $R^{\mathcal{H}'}$. If we take any n-tuple b from $R^{\mathcal{H}'}$, a projection $\pi_i(b)$ is in $D(\pi_i(a))$. Since $D(\pi_i(a))$ is obtained via projection and $\varphi(D(\pi_i(a))) \subseteq D(\pi_i(a))$, there is a tuple b in $R^{\mathcal{H}'}$ such that $\pi_i(b) = \varphi(D(\pi_i(a)))$. For an easier orientation we will denote this tuple as b_i.

   Make a new tuple $t = \varphi(b_1, \varphi(b_2, \varphi(b_3, \ldots, \varphi(b_{n-1}, b_n)\ldots)))$. As φ is in $Pol(\mathcal{H}')$, also t is in $R^{\mathcal{H}'}$. Since $\varphi(g_1, g_2) = g_2$, $t[i] = \varphi(D(a[i]))$.

   Thus we have found an assignment for a tuple a which was arbitrary and therefore we have found a solution to the problem P' and also to the problem P.

□

In the next propositon we will need the equality relation. The equality relation E is a binary relation such that $(a, b) \in E$ if $a = b$.

Proposition 2.4.6. *Assume that φ from $Pol(\mathcal{H})$ is a semilattice operation, $Pol(\mathcal{H}) = Pol(Inv(\varphi))$ and $E \in \mathcal{H}^\star$. Then the problem $CSP(\mathcal{H})$ is* **P***-complete.*

Proof. Assume that $\{0,1\} \subseteq H$ and $\varphi(a_1, a_2) = a_1 \wedge a_2$ for any $a_1, a_2 \in \{0,1\}$. We can suppose that since φ is the only possible binary semilattice operation on $\{0,1\}$.

Define a structure $\mathcal{R}$ with relations $R = \left\{R_1^{\{0,1\}}, \ldots, R_9^{\{0,1\}}\right\}$ where
$R_1^{\{0,1\}} = \{(0)\}$
$R_2^{\{0,1\}} = \{(1)\}$
$R_3^{\{0,1\}} = \{(0,0),(0,1),(1,1)\}$
$R_4^{\{0,1\}} = \{(0,0),(1,0),(1,1)\}$
$R_5^{\{0,1\}} = \{(0,0),(1,0),(0,1)\}$
$R_6^{\{0,1\}} = \{(0,0,0),(0,0,1),(0,1,0),(1,0,0),(1,0,1),(0,1,1),(1,1,1)\}$
$R_7^{\{0,1\}} = \{(0,0,0),(0,0,1),(0,1,0),(1,0,0),(1,1,0),(0,1,1),(1,1,1)\}$
$R_8^{\{0,1\}} = \{(0,0,0),(0,0,1),(0,1,0),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$
$R_9^{\{0,1\}} = \{(0,0,0),(0,0,1),(0,1,0),(1,0,0),(1,0,1),(1,1,0),(0,1,1)\}$

Now we will show that φ is in $Pol(\mathcal{R})$. At first, look at a ternary relation and let us discuss whether we can obtain any tuple which is not in a relation. We need to discuss only for a tuple of exactly one zero.

To obtain a tuple with exactly one zero we have to take this tuple and a tuple $(1,1,1)$ or the second possibility is to take twice this tuple. So in both cases we need to apply the operation φ on a tuple which is not in the relation. Similar discussion can be enforced on binary and unary relations.

Galois connections give $Pol(Inv(\varphi)) \subseteq Pol(Inv(Pol(\mathcal{R}))) = Pol(\mathcal{R})$. Together with $Pol(\mathcal{H}) = Pol(Inv(\varphi))$ gives $Pol(\mathcal{H}) \subseteq Pol(\mathcal{R})$. Provided that E is in $\mathcal{H}^\star$, we can reduce $CSP(\mathcal{R})$, due to lemma B, to $CSP(\mathcal{H})$ in logarithmic-space.

The second part of the proof is based on expressing relations in the Boolean logic:
$R_1^{\{0,1\}}$ can be expressed as $\neg x$
$R_2^{\{0,1\}}$ can be expressed as x
$R_3^{\{0,1\}}$ can be expressed as $\neg x_1 \vee x_2$
$R_4^{\{0,1\}}$ can be expressed as $x_1 \vee \neg x_2$
$R_5^{\{0,1\}}$ can be expressed as $\neg x_1 \vee \neg x_2$
$R_6^{\{0,1\}}$ can be expressed as $\neg x_1 \vee \neg x_2 \vee x_3$
$R_7^{\{0,1\}}$ can be expressed as $\neg x_1 \vee x_2 \vee \neg x_3$
$R_8^{\{0,1\}}$ can be expressed as $x_1 \vee \neg x_2 \vee \neg x_3$
$R_9^{\{0,1\}}$ can be expressed as $\neg x_1 \vee \neg x_2 \vee \neg x_3$

Now we can see that they are expressed as a disjunctive Horn clause with at most 3 variables per clause. Therefore $CSP(\mathcal{R})$ corresponds to the HORN-CLAUSE-SATISFIABILTY with at the most 3 variables per clause, which is known to be **P**-complete. □

2.5 Semiprojections

This section is about semiprojections which ensure **NP**-completeness. We will show a stronger result. If all semiprojections are contained in the set of all polymorphisms it also do not ensure tractability. At first we discuss when there can be semiprojections in the set of all polymorphisms. Then we take a relation for which all semiprojections are polymorphisms and we show the isomorphism with NAESAT which is known to be **NP**-complete. Results can be find in [3]Jeavons: On the algebraic structure of combinatorial problems and [4] Jeavons, Cohen, Gyssens: Closure Properties of Constraints.

Now we remind what semiprojections are. A k-ary operation (at least ternary) which maps every k-tuple consisting of at least two same elements to the ith element is called a semiprojection if it is not a projection.

Theorem 2.5.1. *A semiprojection in $Pol(\mathcal{H})$ yields* **NP***-completeness of $CSP(\mathcal{H})$.*

Proof. We will show that even if $Pol(\mathcal{H})$ contains all semiprojections, it is not enough to ensure tractability. Assume two situations depending on the size of the set H.

- If $|H| = 2$, there is no semiprojection on H.
- Suppose $|H| > 2$. Take a relation $R^{\mathcal{H}}$ defined as follows:

  $$\{(a_1, a_1, a_2), (a_1, a_2, a_1), (a_2, a_1, a_1), (a_1, a_2, a_2), (a_2, a_1, a_2), (a_2, a_2, a_1)\}.$$

  Apply a semiprojection φ on elements from $R^{\mathcal{H}}$. The result will be an element from $R^{\mathcal{H}}$ and thus any semiprojection is in $Pol(\mathcal{H})$, where $\mathcal{H}$ contains the relation $R^{\mathcal{H}}$. Assigning values $0, 1$ to variables a_1, a_2 makes $CSP(\mathcal{H})$ isomorphic to the NOT-ALL-EQUAL SATISFIABILTY problem which is known to be **NP**-complete.

□

2.6 Essentially unary operations and the necessary condition of tractability

At last we look at essentially unary operations. We will show that these operations only gives **NP**-completeness. As the first step we will make an image of the universe of the structure under the essentially unary operation. Then we will distinguish two cases according to the size of the image. If the size of the image will be two we will define a new relation which will correspond to NAESAT and show that conditions of lemma B are fullfilled. Thus we will reduce an **NP**-complete problem to the required CSP of the image of the structure.

If the image of the universe will be more than two we will use the disequality relation and we will reduce GRAPH COLORABILITY problem via lemma B to the CSP of the image of the structure. The last part will be to find a reduction from the CSP of the image of the structure back to the CSP of the structure.

The last part of this section and also of the chapter shows what kind of operations can be in the set of all polymorphisms if CSP is tractable. Results of this section are in [3] Jeavons: On the algebraic structure of combinatorial problems and in [4] Jeavons, Cohen, Gyssens: Closure Properties of the Constraints.

Theorem 2.6.1. *Only essentially unary operation in $Pol(\mathcal{H})$ yields* **NP***-completeness of $CSP(\mathcal{H})$.*

Proof. In the proof we will apply reductions from **NP**-complete problems to $CSP(\mathcal{H})$.

At first let us identify a non-constant function f with an essentially unary operation from $Pol(\mathcal{H})$. We will denote by $f(H) = \{f(h); h \in H\}$ a set of all possible images of the set H under the operation f. We do not impose any restriction on injectivity of the operation f and therefore, if f is not injective, $|f(H)| < |H|$. Among all unary operations f from $Pol(\mathcal{H})$ we will take one with the minimal size of $f(H)$ and denote $f(\mathcal{H}) = \{f(R^{\mathcal{H}})\}$.

As an interphase we will show reduction to $CSP(f(\mathcal{H}))$. We will make use of knowledge that $Pol(f(\mathcal{H}))$ contains only permutations. As g is in $Pol(f(\mathcal{H}))$ and f is in $Pol(\mathcal{H})$,gf is in $Pol(\mathcal{H})$. Thus g is a unary operation and since $f(H)$ is of minimal size, g is an injective operation.

Now we will distinguish two cases. Function f is not constant and thus $|f(H)| > 1$.

1. If $|f(H)| = 2$ let us suppose, without loss of generality, $f(H) = \{0, 1\}$. Assume a relation

$$N = \{(1,1,0),(1,0,1),(0,1,1),(0,0,1),(0,1,0),(1,0,0)\}$$

and a permutation p over three elements.

An element (a_1, a_2, a_3) is in N if and only if $(a_1 \neq a_2 \vee a_2 \neq a_3 \vee a_3 \neq a_1)$ which is same as $(p(a_1) \neq p(a_2) \vee p(a_2) \neq p(a_3) \vee p(a_3) \neq p(a_1))$ as p is the permutation. Thus $(a_1, a_2, a_3) \in N \Leftrightarrow p(a_1, a_2, a_3) \in N$ and $Pol(f(\mathcal{H})) \subseteq Pol((\{0, 1\}, N))$.

However ({0,1},N) constitutes a NOT-ALL-EQUAL SATISFIABILITY problem which is **NP**- complete. By lemma B about reduction we can reduce NOT-ALL-EQUAL problem to $CSP(f(\mathcal{H}))$ in polynomial-time and thus it is **NP**-complete.

2. If $|f(H)| > 2$ we will take a disequality relation $dis = \{(a, b) \in H^2 : a \neq b\}$ and a permutation p from $Pol(f(\mathcal{H}))$. Elements a and b are distinct if and only if elements $p(a)$ and $p(b)$ are distinct since p is the injective permutation. Thus $Pol(f(H)) \subseteq Pol((f(H), dis))$ where $(f(H), dis)$ constitute a GRAPH $|f(H)|$-COLOLABITY problem which is **NP**-complete when $|f(H)| > 2$. Using the same lemma B about reduction gives a polynomial-time reduction to $CSP(f(\mathcal{H}))$.

The final step is a reduction from $CSP(f(\mathcal{H}))$ to $CSP(\mathcal{H})$.

Every relation in $f(\mathcal{H})$ will be replaced by a corresponding relation in $\mathcal{H}$. This replacing needs logarithmic-space. It is a reduction since $R^{(f(\mathcal{H}))} = f(R^{\mathcal{H}}) \subseteq R^{\mathcal{H}}$. A solution to $CSP(f(\mathcal{H}))$ is also a solution to $CSP(\mathcal{H})$. If a homomorphism h is a solution to $CSP(\mathcal{H})$, a composition fh is a solution to $CSP(f(\mathcal{H}))$.

We have established a reduction from an **NP**-comlete problem to $CSP(\mathcal{H})$ in polynomial time and thus $CSP(\mathcal{H})$ is **NP**-complete. □

Proposition 2.6.2. *If $CSP(\mathcal{H})$ is tractable, $Pol(\mathcal{H})$ must contain at least one from the following operations: a constant operation, a majority operation, an idempotent binary operation which is not a projection, an affine operation or a semiprojection.*

Proof. If $Pol(\mathcal{H})$ contains essentially unary operations only, $CSP(\mathcal{H})$ is **NP**-complete. A set of all polymorphisms, $Pol(\mathcal{H})$, can contain only a limited

number of types of operations, exactly essentially unary operations or operations mentioned in the proposition which gives a necessary condition for tractability. □

Now we have established the necessary condition for the tractability of $CSP(\mathcal{H})$.

Chapter 3

Algebras and CSPs

This chapter connects algebras and constraint satisfactions problems. First of all we state the (l, k)-algorithm to be able to define bounded width. If CSP has bounded width it is also tractable.

In the next section we introduce algebras and variaties. In lemmas and theorems we will often need to know what term operations are. With use of $CSPs$ we define bounded width algebras and tractable algebras. We will look at subalgebras, homomorphic images and direct powers to extend tractability and bounded width to varieties generated by algebras.

Later we will turn the attention to special types of algebras, i.e. to surjective, term induced algebras and full idempotent reducts of algebras. We will show that an algebra and its term induced algebra has the same complexity which is powerfull in a way that we do not have to examine an algebra but it is sufficient to examine only its term induced algebra.

More deeply we will look at surjective algebras and we will show that surjective algebras and its full idempotent reducts have also the same complexity.

3.1 Algebras

The first section is about algebras, definitions of bounded width $CSPs$ by the (l, k)-algorithm. The (l, k)-algorithm is a polynomial algorithm which is based on intersections of specific k-substructures and on decresing the number of solutions to the subproblems. As an output we receive solutions for k-substructures.

Finally we will turn our attention to algebras. We will briefly remind

what algebra is and what term operations are. We will also define varieties and especially for the next sections varieties generated by algebras. With use of $CSPs$ we define complexity of an algebra and of a bounded width algebra.

Key texts for this section are [2] Larose, Zadori: Bounded width problems and algebras, [6] Bulatov, Valeriote: Recent results on the algebraic approach to the CSP.

Suppose we have a concrete problem $P = (\mathcal{G}, \mathcal{H})$. For a subset K of the set G, a problem $P_{|K} = (\mathcal{G}', \mathcal{H}')$ is a subproblem of P restricted to the set K as in the section 2.2 about consistency. This restriction can be done by projection: go through every tuple from every relation $R^{\mathcal{G}}$ from structure $\mathcal{G}$ and project this tuple to a new tuple containing only elements from the set K. By the same projection project every tuple from the corresponding relation $R^{\mathcal{H}}$ from structure $\mathcal{H}$ to obtain a new corresponding relation.

As a k-subset we take any subset of size at the most k where k is a positive integer. In a similar way a k-substructure is a substructure with the size of its universe at the most k. Relations are the restricted ones.

We denote ρ_K the set of all solutions to the problem $P_{|K}$. When we connect it with notion of consistency from section 2.2, ρ_K is the set of all locally consistent assignments for set K. As we know set ρ_K can be bigger than the set of all solutions to the whole problem P.

Now we will state an algorithm which we will use in the definition of bounded width algebras.

Algorithm ((l,k)-algorithm)**.** *Assume a problem $P = (\mathcal{G}, \mathcal{H})$ and $1 \leq l < k$.*

1. *Take two k-substructures $\mathcal{I}, \mathcal{J}$ of $\mathcal{G}$ with properties:*
   - $|I \cap J| \leq l$
   - *There is a homomorphism φ from ρ_I such that for any homomorphism ψ from ρ_J, $\varphi_{|I \cap J} \neq \psi_{|I \cap J}$.*
2. *If there is such φ put it away from ρ_I.*

The algorithm ends if there are no such substructures. As an output of the (l, k)-algorithm we take sets of all solutions for k-substructures of $\mathcal{G}$, i.e. a set of all homomorphisms from any k-substructure of $\mathcal{G}$ to the structure $\mathcal{H}$.

Remark. Every structure has only a polynomial amount of k-substructures since the amount of k-substructures is $\binom{|G|}{k}$ which is less than $|G|^k$. A set ρ_I is finite and during the algorithm its size is decreasing. Thus the (l,k)-algorithm is polynomial in the size of G. If we run two different version of the (l,k)-algorithm we will receive the same output. Proof of this state is in [2] Larose, Zadori: Bounded width problems and algebras.

Definition 3.1.1. We say that $CSP(\mathcal{H})$ has *width (l,k)* if the non-emty output of the (l,k)-algorithm on any structure $\mathcal{G}$ and on the structure $\mathcal{H}$ yields a homomorphism from $\mathcal{G}$ to $\mathcal{H}$.

Definition 3.1.2. We say that $CSP(\mathcal{H})$ has *width* l if it has width (l,k) for some k and we say that it has *bounded width* if it has width (l,k) for some l and k.

Proposition 3.1.3. *If $CSP(\mathcal{H})$ has bounded width, it is tractable.*

Proof. The result follows from the definition of the bounded width and from the run of the polynomial (l,k)-algorithm which is known to be polynomial. □

Example. *Let us assume problem $P = (\mathcal{G}, \mathcal{H})$ where $G = \{a,b,c\}, H = \{1,2\}$. Relations are $R^{\mathcal{G}} = \{(abac),(bcaa),(aabb)\}, R^{\mathcal{H}} = \{(1111),(1121),(1122),(1212),(2112),(2211),(2212)\}$. We will ilustrate the run of the $(1,2)$-algorithm.*

We need all 2-substructures and its sets ρ. At first we will look at a substructure with universe $\{a\}$. We will find a restriction $P_{|\{a\}}$.

If we project the tuple $(abac)$ we get a new tuple (aa). Projecting the relation $R^{\mathcal{H}}$ gives tuples: $(11),(12),(12),(11),(21),(21),(21)$. Thus the first relations are $R_1^{\mathcal{G}} = \{(aa)\}$,$R_1^{\mathcal{H}} = \{(11),(12),(21)\}$. If we project the second tuple $(bcaa)$ and by the same projection also relation $R^{\mathcal{H}}$ we get: (aa),$(11),(21),(22),(12),(12),(11),(12)$ and thus the second relations are $R_2^{\mathcal{G}} = \{(aa)\}$, $R_2^{\mathcal{H}} = \{(11),(12),(21),(22)\}$. Finally we will look at tuple $(aabb)$ and its projection. We get $(aa),(11),(11),(11),(12),(21),(22),(22)$ and thus the third relations are $R_3^{\mathcal{G}} = \{(aa)\}, R_3^{\mathcal{H}} = \{(11),(12),(21),(22)\}$. This new problem has the only one solution $a = 1$.

We will not show here the same method for other subsets but we make a table of results.

- $\rho_{|\{a\}} = \{a = 1\}$

- $\rho_{|\{b\}} = \{b = 1, b = 2\}$
- $\rho_{|\{c\}} = \{c = 1, c = 2\}$
- $\rho_{|\{ab\}} = \{ab = 12\}$
- $\rho_{|\{ac\}} = \{ac = 11, ac = 12\}$
- $\rho_{|\{bc\}} = \{bc = 11, bc = 12, bc = 21, bc = 22\}$

Now we can start the $(1, 2)$*-algortihm.*

At first we take sets $\{a\}$ *and* $\{ab\}$*. In* $\rho_{|\{a\}}$ *there is the only one* ρ*. In* $\rho_{|\{ab\}}$ *there is also the only one* ψ*. Restrictions to* a *gives* $a = 1$ *in both* ρ, ψ *and thus we cannot go to step* 2*.*

The same result is if we take sets in the inverse order. For sets $\{a\}, \{ac\}$ *if* $a = 1$ *there is* $ac = 11$ *and thus we cannot go to step* 2 *with* $\rho_{|\{a\}}$*. Conversely if we assume* $ac = 11$ *there is* $a = 1$ *in* $\rho_{|\{a\}}$ *and thus we again cannot go to step* 2*. Similarly for* $ac = 12$*.*

If we take sets $\{b\}, \{ab\}$ *for* $b = 1$ *there is only* $ab = 12$ *in* $\rho_{|\{ab\}}$ *and thus restriction to* b *is* $b = 2$ *which is different from* $b = 1$ *in* $\rho_{|\{b\}}$*. Therefore we continue to step* 2 *and throw* $b = 1$ *from* $\rho_{|\{b\}}$*. If we continue with* $b = 2$ *we can take* $ab = 12$ *where values are* $b = 2$ *in both and thus we do not continue with step* 2*. Conversely for* $ab = 12$ *there is* $b = 2$ *and we do not continue with step* 2 *as well.*

Now we will take sets $\{b\}, \{bc\}$*. The only possibility for* b *is* $b = 2$*. We can take from* $\rho_{|\{bc\}}$ *for example* $bc = 21$ *and thus we cannot go to step* 2*. For the second direction we will start with* $bc = 11$*. In this case the value is* $b = 1$ *which is different from the only one possible value* $b = 2$ *in* $\rho_{|\{b\}}$ *and thus we go to step* 2 *and throw away* $bc = 11$ *from* $\rho_{|\{bc\}}$*. Similarly with* $bc = 12$*. Other assignments* $bc = 21, 22$ *stay in* $\rho_{|\{bc\}}$*.*

If we take sets $\{c\}, \{ac\}$ *we cannot go to step* 2*. For sets* $\{c\}, \{bc\}$ *is situation the same. If we continue with sets* $\{ab\}, \{ac\}$ *and* $\{ab\}, \{bc\}$ *and* $\{ac\}, \{bc\}$ *we will discover that we never move to step* 2 *and thus the algorithm ends as we have gone through all* 2*-substructures*

If the intersection of two substructures is empty we always get equality of assignments restricted to this intersection and thus we do not have to run the algorithm for them. The output of the algorithm is:

- $\rho_{|\{a\}} = \{a = 1\}$
- $\rho_{|\{b\}} = \{b = 2\}$

- $\rho_{|\{c\}} = \{c = 1, c = 2\}$
- $\rho_{|\{ab\}} = \{ab = 12\}$
- $\rho_{|\{ac\}} = \{ac = 11, ac = 12\}$
- $\rho_{|\{bc\}} = \{bc = 21, bc = 22\}$

Now we will try to connect algebras and constraint satisfaction problems. Suppose an algebra $\mathbb{A}$ with the universe A and with a set of basic operations F. In the whole text suppose that the universe of an algebra is always denoted by the same letter in an italic font as the whole algebra.

Definition 3.1.4. A set of all term operations, compositions of basic operations and projections, of the algebra $\mathbb{A}$ is denoted $Term(\mathbb{A})$. Any operation φ is a term operation if and only if $\varphi \in Pol(Inv(F))$. For easier manipulation we will write $Inv(\mathbb{A})$ instead of $Inv(F)$.

Every $CSP(\mathcal{H})$ can be associated with an algebra $Alg(\mathcal{H}) = (H, Pol(\mathcal{H}))$. For the other direction we will need another class of structures.

Definition 3.1.5. Denote $Str(\mathbb{A})$ as a class of finite structures $\mathcal{A}$ with property $Term(\mathbb{A}) \subseteq Pol(\mathcal{A})$. This property can be said as $R^{\mathcal{A}} \in Inv(\mathbb{A})$ for every relation $R^{\mathcal{A}}$.

Remark. We will write $\mathcal{A} \subseteq Inv(\mathbb{A})$ to say that for every relation $R^{\mathcal{A}}$ from structure $\mathcal{A}$ holds $R^{\mathcal{A}} \in Inv(\mathbb{A})$. We will use this marking for the above definition.

Now we can define a uniform constraint satisfaction problem.

Definition 3.1.6. A *uniform constraint satisfaction problem*, $CSP(\mathfrak{H})$,where $\mathfrak{H}$ is a class of relational structures, is a problem of deciding whether there exists a homomorphism from a given structure $\mathcal{G}$ to every structure $\mathcal{H}$ from $\mathfrak{H}$.

Every algebra $\mathbb{A}$ can be associated in a natural way with a uniform constraint satisfaction problem $CSP(Str(\mathbb{A}))$.

Definition 3.1.7. An algebra $\mathbb{A}$ has *bounded width* if $CSP(\mathcal{A})$ has bounded width for every structure $\mathcal{A}$ from $Str(\mathbb{A})$.

An algebra is *locally tractable* if for every structure $\mathcal{A}$ from $Str(\mathbb{A})$, a non-uniform problem $CSP(\mathcal{A})$ is tractable and is *globally tractable* if also union of $CSP(\mathcal{A})$ is tractable. If any structure from $Str(\mathbb{A})$ is **NP**-complete, the algebra $\mathbb{A}$ is also **NP**-complete.

Now we will define a subalgebra, a homomorphic image of an algebra and a direct product of an algebra. We will have enough material for telling what a variety is. In next two sections we will take a look at tractability and bounded width of a variety

Definition 3.1.8. An algebra $\mathbb{B} = (B, G)$ is a *subalgebra* of $\mathbb{A} = (A, F)$ if B is a subset of A and basic operations from G are restrictions of basic operations from F to the set B.

It holds that $f(b_1, \ldots, b_n)$ is in B for any basic operation f from F and any tuple of elements from B.

Definition 3.1.9. Let $\mathbb{A} = (A, F), \mathbb{B} = (B, G)$ be two algebras and suppose that F, G contains the same amount of operations of the same arity. A *homomorphism* from $\mathbb{A}$ to $\mathbb{B}$ is a mapping f such that $f(\varphi(a_1, a_2, \ldots, a_n)) = \psi(f(a_1), f(a_2), \ldots, f(a_n))$ for any operation φ from F and any operation ψ from G. In this case the algebra $\mathbb{B}$ is a *homomorphic image* of the algebra $\mathbb{A}$.

Definition 3.1.10. The *n-th direct power* of an algebra $\mathbb{A} = (A, F)$ is an algebra $\mathbb{A}^n = (A^n, F^n)$ where operations are taken component-wise as in the definition of k-ary operations.

Definition 3.1.11. The *direct product* of algebras is an algebra where operations are taken component-wise.

Definition 3.1.12. A *variety* *Var* is a class of algebras which satisfies the following three conditions:

1. A subalgebra of an algebra from the class *Var* is also in the class.
2. A homomorphic image of an algebra from the class *Var* is also in the class.
3. A direct product of algebras from the class *Var* is also in the class.

A *variety generated by an algebra* $\mathbb{A}$, $var(\mathbb{A})$, is the smallest variety which contains the algebra $\mathbb{A}$. It is known that $var(\mathbb{A})$ is the class consisting of homomorphic images of subalgebras of direct powers of the algebra $\mathbb{A}$.

3.2 Tractability

We will show that subalgebras, homomorphic images and direct products of tractable algebras are also tractable. When we want to show that an algebra $\mathbb{B}$ is tractable we need to show that $CSP(\mathcal{B})$ is tractable for any finite structure $\mathcal{B}$ from $Inv(\mathbb{B})$ that is equivalent to condition $Term(\mathbb{B}) \subseteq Pol(\mathcal{B})$.

Finally we get the main theorem of this section that a variety generated by a tractable algebra is also tractable which we will get as a combination of former lemmas about subslgebras, homomorphic images and direct products. You can find it in [7] Bulatov, Jeavons, Krohkin: Classifying the complexity of constraints, [15] Krohkin:Universal Algebra and CSP: The Basics.

Proof of tractability of a subalgebra is based on inclusion of set of invariants of the subalgebra and set of invariants of the algebra and closeness of algebras. The proof about homomorphic images is based on inverse relations and showing that it is an invariant to the algebra. Then we just need to find any reduction between CSPs. Proof of lemma about direct power is based on introducing a new longer mn-ary relation made from an m-ary relation if we have an nth power of the algebra. Then we just need to find a reduction.

Lemma. *A subalgebra of a tractable algebra is also tractable.*

Proof. Let us take a tractable algebra $\mathbb{A}$, its subalgebra $\mathbb{B}$ and a finite structure $\mathcal{B}$ such that $\mathcal{B} \subseteq Inv(\mathbb{B})$.We will show that $Inv(\mathbb{B}) \subseteq Inv(\mathbb{A})$.

It must be true because $\mathcal{B}$ has relations on B and $\mathbb{B}$ is operation closed. Thus $\mathcal{B} \subseteq Inv(\mathbb{A})$ and $CSP(\mathcal{B})$ is tractable from tractability of the algebra $\mathbb{A}$. □

Lemma. *If any subalgebra is* **NP***-complete, the algebra is also* **NP***-complete.*

Proof. Let us take an algebra $\mathbb{A}$, its **NP**-complete subalgebra $\mathbb{B}$ and a finite structure $\mathcal{B}$ from $Str(\mathbb{B})$ such that $CSP(\mathcal{B})$ is **NP**-complete. From the above proof we know that $\mathcal{B}$ is in $Str(\mathbb{A})$ and thus $\mathbb{A}$ is **NP**-complete from the definition. □

Lemma. *A homomorphic image of a tractable algebra is also tractable.*

Proof. Let us take a tractable algebra $\mathbb{A}$, a homomorphism φ and an associated homomorphic image $\mathbb{B}$. Suppose that a structure $\mathcal{B}$ is from $Inv(\mathbb{B})$ and let us take an n-ary relation from $R^{\mathcal{B}}$. We will show that its inverse relation is an invariant of the algebra $\mathbb{A}$.

We can see that $\varphi^{-1}(R^{\mathcal{B}}) = \{(a_1, \ldots, a_n); \varphi(a_1, \ldots, a_n) \in R^{\mathcal{B}}\}$ is also an n-ary relation and since the homomorphism φ maps from $\mathbb{A}$ to $\mathbb{B}$ it is a relation on A. From inclusion and equalities

$$f(\varphi^{-1}(R^{\mathcal{B}})) = \varphi^{-1}(\varphi f(\varphi^{-1}(R^{\mathcal{B}}))) = \varphi^{-1}(g(\varphi\varphi^{-1}(R^{\mathcal{B}}))) \subseteq \varphi^{-1}(R^{\mathcal{B}}),$$

$\varphi^{-1}(R^{\mathcal{B}})$ is an invariant where f is a basic operation from the algebra $\mathbb{A}$ and g is a basic operation from the algebra $\mathbb{B}$. We have used the definition of a homomorphic image and knowledge that $\mathcal{B}$ is an invariant.

Now we will reduce $CSP(\mathcal{B})$ to $CSP(\mathcal{A})$ where the structure $\mathcal{A}$ has relations $\varphi^{-1}(\mathcal{B})$. Let us take a concrete problem P from $CSP(\mathcal{B})$ and a problem P' with the same instance as P from $CSP(\mathcal{A})$. If ψ is a solution to P' then $\varphi\psi$ is a solution to P. Conversely if ξ is a solution to P then any ψ such that $\varphi\psi(b) = \xi(b)$ is a solution to P'. □

Lemma. *If any homomorphic image of an algebra is* **NP***-complete, the algebra is also* **NP***-complete.*

Proof. The proof follows from the definition of an **NP**-complete algebra and from the above lemma. □

Remark 3.2.1. *Let us take an n-ary relation R from $Inv(\mathbb{A})$ and assume a basic k-ary operation f from basic operations of the algebra $\mathbb{A}$. Imposing the operation f on k tuples from the relation R yields an n-tuple from the relation since the relation is invariant under the operation. From the definition of a relation, the relation R is a subset of A^n and thus from the definition of a subalgebra, the relation R rises a subalgebra of the algebra $\mathbb{A}^n$.*

Lemma 3.2.2. *A direct power of a tractable algebra is also tractable.*

Proof. Let us take a tractable algebra $\mathbb{A}$, its direct power $\mathbb{B} = \mathbb{A}^n$ and a finite structure $\mathcal{B}$ from $Inv(\mathbb{B})$. Now we will try to make from an m-ary relation an mn-ary relation.

For this matter let us take an m-tuple $(b_1, b_2, \ldots, b_m)$ from an m-ary relation $R^{\mathcal{B}}$. Since the universe B has as elements n-tuples from A, any element b_i is an n-tuple from A and thus we can define mn-tuples and also an mn-ary relation $R^{\mathcal{A}}$ in a following way:

$$((b_{11}, b_{12}, \ldots, b_{1n}), \ldots, (b_{m1}, b_{m2}, \ldots, b_{mn})) \in R^{\mathcal{B}}$$

$$\Downarrow$$

$$(b_{11}, b_{12}, \ldots, b_{1n}, \ldots, b_{m1}, b_{m2}, \ldots, b_{mn}) \in R^{\mathcal{A}}.$$

Since $\mathcal{B} \subseteq Inv(\mathbb{B})$, follows from the definition that $\mathcal{A} = (A, R^{\mathcal{A}}) \subseteq Inv(\mathbb{A})$.

Next step of the proof is to find of a reduction from $CSP(\mathcal{B})$ to $CSP(\mathcal{A})$. Suppose an m-tuple $(a_1, a_2, \ldots, a_m)$ in an instance of $CSP(\mathcal{B})$. We will introduce a new mn-tuple $(a_1^1, a_1^2, \ldots, a_1^n, a_2^1, \ldots, a_2^n, \ldots, a_m^1, \ldots, a_m^n)$ which will be in an instance of $CSP(\mathcal{A})$. In this manner we can transform problem in $CSP(\mathcal{B})$ to problem in $CSP(\mathcal{A})$. This procedure can be made in polynomial time. A homomorphism is a solution to $CSP(\mathcal{B})$ if and only if it is a solution to $CSP(\mathcal{A})$.

We have found a reduction to a tractable problem and thus $\mathbb{B}$ is also tractable. □

Lemma. *If any direct power of an algebra is* **NP***-complete, the algebra is also* **NP***-complete.*

Proof. The proof follows from the definition of an **NP**-complete algebra and from the definition of a direct power. □

Theorem 3.2.3. *A variety generated by a tractable algebra is also tractable.*

Proof. The proof follows from the above lemmas. □

Theorem 3.2.4. *If there is an* **NP***-complete algebra in a variety, the variety is also* **NP***-complete.*

Proof. The proof follows from the above lemmas. □

3.3 Bounded width

In this section we take a look at algebras of bounded width and we prove that a subalgebra, a homomorphic image and a direct power of a bounded width algebra has also bounded width. We can divide a procedure of showing that an algebra $\mathbb{B}$ has bounded width into three steps:

1. We take a structure $\mathcal{B}$ from $Str(\mathbb{B})$, in other words $\mathcal{B}$ such that $\mathcal{B}$ is from $Inv(\mathbb{B})$.

2. We take a structure $\mathcal{I}$ similiar to $\mathcal{B}$ for which the *(l,k)*-algorithm on $(\mathcal{I}, \mathcal{B})$ yields a non-emtpy output.

3. We find a homomorphism from $\mathcal{I}$ to $\mathcal{B}$.

In the case of an subalgebra we define a new structure for the algebra and show that it satisfies the first point of the procedure. Then we take a similar structure and with a similar structure for the subalgebra we show that the (l, k)-algorithms give the same output. As the algebra has bounded width we can find a homomorphism. If the new structure is well defined the homomorphism will also be a homomorphism to the substructure.

For the algebra $\mathbb{A}$ we define structure $\mathcal{A}$ such that its homomorphic image is structure $\mathcal{B}$ which satisfies two first conditions for the homomorphic image $\mathbb{B}$. A structure similar to $\mathcal{B}$ is also similar to $\mathcal{A}$. By means of composition of homomorphisms we will show that the (l, k)-algorithm for $\mathcal{A}$ gives also non-empty solutions. Homomorphism for the structure $\mathcal{B}$ will be a composition with the homomorphism for the structure $\mathcal{A}$ which we get from the properties of bounded width algebra $\mathbb{A}$.

Finally we take a look at a direct power. We define new structure $\mathcal{A}$ for the algebra $\mathbb{A}$ from the structure $\mathcal{B}$ of the direct power $\mathbb{B}$. We take a similar structure and with use of making copies and decomposing we show that the (l, k)-algorithm gives also an non-empty output for $\mathcal{A}$. The last step is to take a homomorphism and convert it to the homomorphism for the structure $\mathcal{B}$.

A key text for this section is [2] Larose, Zadori: Bounded width problems and algebras.

Lemma. *A subalgebra of a bounded width algebra has also bounded width.*

Proof. Let us take a bounded width algebra $\mathbb{A}$, its subalgebra $\mathbb{B}$ and a structure $\mathcal{B} = (B, R^{\mathcal{B}})$ from $Inv(\mathbb{B})$. Without loss of generality we can assume that the algebra $\mathbb{A}$ has width (l, k) where $l = k - 1$ as from the definition of width holds that if an algebra has width (l, k) it also has width (l', k) where $l' > l$. Now we will take a structure $\mathcal{I}$ similar to $\mathcal{B}$ for which the (l, k)-algorithm on $(\mathcal{I}, \mathcal{B})$ yields a non-empty output. Our task is to find a homomorphism from $\mathcal{I}$ to the structure $\mathcal{B}$.

For this purpose we will define a structure $\mathcal{A} = (A, R^{\mathcal{B}} \cup B)$ and a similar structure $\mathcal{J} = (I, R^{\mathcal{I}} \cup I)$ where B and I are taken as unary relations. We will show that $\mathcal{A} \subseteq Inv(\mathbb{A})$ and that the (l, k)-algorithm on $(\mathcal{J}, \mathcal{A})$ yields a non-empty output.

Since $\mathbb{B}$ is a subalgebra of $\mathbb{A}$ and $\mathcal{B}$ from $Inv(\mathbb{B})$, from the definition of $\mathcal{A}$ follows that $\mathcal{A}$ is from $Inv(\mathbb{A})$.

We will show that an (l, k)-algorithm on $(\mathcal{I}, \mathcal{B})$ has the same output as on $(\mathcal{J}, \mathcal{A})$. Assume a k-substructure $\mathcal{K}$ of the structure $\mathcal{I}$ and a k-substructure

$\mathcal{L}$ of the structure $\mathcal{J}$ with the same universe as $\mathcal{K}$. Any homomorphism from $\mathcal{L}$ to $\mathcal{A}$ is also a homomorphism from $\mathcal{K}$ to $\mathcal{B}$ from the definition of relations $\mathcal{J}, \mathcal{A}$.

For a homomorphism from $\mathcal{K}$ to $\mathcal{B}$ we have to show that it maps tuples from the unary relation K to tuples from the unary relation B. This is true as it is a homomorphism from K to B. Thus a set of all homomorphisms from $\mathcal{L}$ to $\mathcal{A}$ is same as a set of all homomorphisms from $\mathcal{K}$ to $\mathcal{B}$.

Since the (l, k)-algorithm on $(\mathcal{I}, \mathcal{B})$ gives a non-empty output, the (l, k)-algorithm on $(\mathcal{J}, \mathcal{A})$ gives also a non-empty output. As the algebra $\mathbb{A}$ has bounded width there is a homomorphism from $\mathcal{J}$ to $\mathcal{A}$ and from the definition of structures it is also a homomorphism from $\mathcal{I}$ to $\mathcal{B}$ and thus $\mathbb{B}$ has bounded width. □

Lemma. *A homomorphic image of a bounded width algebra has also bounded width.*

Proof. Let us take a bounded width algebra $\mathbb{A}$, a homomorphism φ and a homomorphic image $\mathbb{B} = \varphi(\mathbb{A})$. Suppose a structure $\mathcal{B} = (B, R^{\mathcal{B}})$ from $Str(\mathbb{B})$. We will define a structure $\mathcal{A} = (A, R^{\mathcal{A}})$ where $\varphi(\mathcal{A}) = \mathcal{B}$.

From the definition of a homomorphic image and invariant property, the structure $\mathcal{A}$ is also an invariant to $\mathbb{A}$. Now we will take a structure $\mathcal{I}$ similar to the structure $\mathcal{B}$ for which an *(l,k)-algorithm* on $(\mathcal{I}, \mathcal{B})$ yields non-empty output. However the structure $\mathcal{I}$ is also similar to the structure $\mathcal{A}$ and thus we can impose the same (l, k)-algorithm on $(\mathcal{I}, \mathcal{A})$.

We will show that the (l, k)-algorithm gives a non-empty output for $(\mathcal{I}, \mathcal{A})$. We can take a homomorphism ψ from a k-substructure of $\mathcal{I}$ to the structure $\mathcal{B}$ because we know that such a homomorphism exists. A composition $\varphi^{-1}\psi$ is a homomorphism from the same k-substructure but at this time to the structure $\mathcal{A}$.

Since the algebra $\mathbb{A}$ has bounded width, there is a homomorphism from $\mathcal{I}$ to $\mathcal{A}$. Composing with φ yields a homomorphism from $\mathcal{I}$ to $\mathcal{B}$ and therefore $\mathbb{B}$ has also bounded width. □

Lemma. *A direct power of a bounded width algebra has also bounded width.*

Proof. Let us take a bounded width algebra $\mathbb{A}$, its direct power $\mathbb{B} = \mathbb{A}^n$ and a structure $\mathcal{B}$ from $Inv(\mathbb{B})$. The beginning of the proof will be the same as in the proof of the lemma 3.2.2 of tractability of a direct power.

We will define in the same way an mn-ary relation from an m-ary relation and a structure $\mathcal{A}$ from the structure $\mathcal{B}$.

$$((b_{11}, b_{12}, \ldots, b_{1n}), \ldots, (b_{m1}, b_{m2}, \ldots, b_{mn})) \in R^{\mathcal{B}}$$

$$\Downarrow$$

$$(b_{11}, b_{12}, \ldots, b_{1n}, \ldots, b_{m1}, b_{m2}, \ldots, b_{mn}) \in R^{\mathcal{A}}.$$

Since $\mathcal{B} \subseteq Inv(\mathbb{B})$, follows from the definition that $\mathcal{A} = (A, R^{\mathcal{A}}) \subseteq Inv(\mathbb{A})$.

As a next step we will take a structure $\mathcal{I}$ similar to the structure $\mathcal{B}$ for which an (l, k)-algorithm on $(\mathcal{I}, \mathcal{B})$ gives non-empty output. We will define a structure $\mathcal{J}$ similar to the structure $\mathcal{A}$ and we will show that the same (l, k)-algorithm on $(\mathcal{J}, \mathcal{A})$ gives non-empty output. For this purpose we will make n copies of the structure $\mathcal{I}$.

If we have an element b_j from the ith copy of an m-tuple $(b_1, b_2, \ldots, b_m)$ from $R^{\mathcal{I}}$, we will write b_{ji} instead of b_j to emphasize from which copy it is taken. Now we are ready to define the structure $\mathcal{J}$ where $J = \bigcup_{i=1}^{n} I$ and for relations hold:

$$(b_1, b_2, \ldots, b_m) \in R^{\mathcal{I}}$$

$$\Downarrow$$

$$(b_{11}, b_{12}, \ldots, b_{1n}, \ldots, b_{m1}, \ldots, b_{mn}) \in R^{\mathcal{J}}.$$

We will look at the output of the (l, k)-algorithm on $(\mathcal{J}, \mathcal{A})$. Let us take a k-substructure $\mathcal{K}$ of the structure $\mathcal{J}$ and decompose it into n substructures $\mathcal{D}_i$ according to the copies of the structure $\mathcal{I}$ in the structure $\mathcal{J}$. We can find a k-substructure $\mathcal{H}$ from $\mathcal{I}$ such that its copies have all the elements in $\mathcal{K}$. Suppose that t is in an output of the (l, k)-algorithm on $(\mathcal{I}, \mathcal{B})$ for the k-substructure $\mathcal{H}$. We will make a tuple r such that for every $d \in D_i$ holds $r(d) = t(h)$ where d is an i-th copy of $h \in \mathcal{H}$ in $\mathcal{J}$. Thus r is an output of the (l, k) algorithm on $(\mathcal{J}, \mathcal{A})$.

As the (l, k)-algorithm on $(\mathcal{I}, \mathcal{B})$ gives non-empty output also the (l, k)-algorithm on $(\mathcal{J}, \mathcal{A})$ gives non-empty output. As the algebra $\mathbb{A}$ has bounded width we can take a homomorphism from $\mathcal{J}$ to the structure $\mathcal{A}$. Let us denote it by φ and define a map $g : \mathcal{I} \to \mathcal{B}$ such that $x \mapsto (\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_n))$ where x_i is the i-th copy of x from the structure $\mathcal{I}$ in the structure $\mathcal{J}$. Therefore the algebra $\mathbb{B}$ has bounded width. □

Theorem 3.3.1. *A variety generated by a bounded width algebra has also bounded width.*

Proof. The proof follows immediately from above lemmas and the definition of a variety generated by an algebra. □

Theorem 3.3.2. *If a variety generated by an algebra contains an algebra which does not have bounded width, the variety also does not have bounded width.*

Proof. Follows from the above lemmas. □

3.4 Types of algebras

This section is about special types of algebras, especially about surjective and idempotent ones and also about full idempotent reducts of algebras. We will remind that complexity of $CSP(\mathcal{H})$ is same as complexity of $CSP(f(\mathcal{H}))$ if f is a unary surjective operation. We will also remind that unary surjective operations form group. We will proof that the complexity of an algebra is same as the complexity of a term induced algebra.

For this section you can use [6] Bulatov, Valeriote: Recent Result on the Algebraic Approach to the CSP or [7] Bulatov, Jeavons, Krohkin: Classifying the Complexity of Constraints Using Finite Algebras.

Definition 3.4.1. An algebra is *surjective* if every unary term operation of the algebra is surjective. An algebra is *idempotent* if every term operation of the algebra is idempotent.

Definition 3.4.2. A *full idempotent reduct* of an algebra $\mathbb{A}$ is an algebra whose basic operations are idempotent term operations of the algebra $\mathbb{A}$. We will denote the full idempotent reduct of an algebra $\mathbb{A}$ as $Id(\mathbb{A})$.

Remark 3.4.3. *From the proof of the theorem 2.6.1 about tractability and essentially unary operations it follows that unary surjective operations form a group as a unary operation is surjective if and only if it is injective. We have also showed that we can reduce $CSP(f(\mathcal{H}))$ to $CSP(\mathcal{H})$ in logarithmic space. In the same manner we can prove a reduction from $CSP(\mathcal{H})$ to $CSP(f(\mathcal{H}))$ in logarithmic space. Thus both problems are log-space equivalent for an unary surjective operation f. As a consequence $CSP(\mathcal{H})$ is tractable, respectively* **NP***-complete, if and only if $CSP(f(\mathcal{H}))$ is tractable, respectively* **NP***-complete.*

We will turn our attention to surjective algebras where we will use the above remark in proofs. At first we will try to make from an algebra a surjective one.

Definition 3.4.4. Let us assume an algebra $\mathbb{A}$. We will need a unary term operation g with the minimal $|g(A)|$. We define an algebra $g(\mathbb{A})$ as an algebra with universe $g(A)$. Basic operations will be compositions of the operation g and any term operation of the algebra $\mathbb{A}$ restricted to the set $g(A)$. We call a new algebra $g(\mathbb{A})$ a *term induced algebra*.

Remark. The term induced algebra from the above definition is surjective since $g(A)$ was of minimal size.

Proposition 3.4.5. *An algebra is tractable, respectively* **NP***-complete, if and only if its term induced algebra is tractable, respectively* **NP***-complete.*

Proof. The proof follows from the definition of a term induced algebra and the above remark 3.4.3 about equivalence between problems. □

The last lemma gives us a powerful tool as we do not have to test algebras but it is sufficient to look at the term induced algebras since the complexity of both is the same.

3.5 Surjective algebras

We will show that a surjective algebra is tractable if and only if its full idempotent reduct is tractable. For this purpose we have to characterize idempotent operations and define a special relation. For characterisation of idempotent operations we will need unary relations which contain exactly one tuple. As a new relation we will define relation derived from a permutation group. We will connect them with idempotent operations and invariants.

Main theorem of this section is about the complexity of a finite surjective algebra and its full idempotent reduct. In the proof we will use a lemma about equivalence. Proof of tractability is based on chains of reductions. Key text for this section is [7] Bulatov, Jeavons, Krohkin: Classifying the Complexity of Constraints Using Finite Algebras. The last part of remark 3.5.3 can be found in [10] Szendrei: Clones in Universal Algebra.

We remind the definition of a surjective algebra and of a full idempotent reduct. If unary term operations of an algebra are surjective, the algebra is surjective. If basic operations of an algebra are idempotent term operations of another algebra, we call the algebra a full idempotent reduct. At first we will define a new unary relation.

Let us have a unary relation $R = \{(a)\}$ on the set A consisting of one unary tuple (a) where the element a is from the set A. In the next few paragraphs we will often need such relations and therefore we will state it as a definition.

Definition 3.5.1. We will denote the set of all such unary relations as R_u^A.

Assume a k-ary idempotent operation f on a set A. Applying the operation f on a relation R from the set R_u^A is applying the operation on k tuples from the relation. However in the relation is exactly one tuple and thus, from the definition of operations on tuples, it gives $(f(a, a, \ldots, a))$ which is from the idempotency of the operation tuple (a) which is the whole relation. Thus the operation preserves relations from the set R_u^A.

For the second direction suppose that f is a k-ary operation which preserves every relation from the set R_u^A. This means that applying the operation on every k tuples from the relation which is from the above argument $(f(a, a, \ldots, a))$, gives the whole relation. However this means that $(f(a, a, \ldots, a)) = (a)$ which says that the operation is idempotent.

We have characterised idempotent operations. It is important to know that for a full idempotent reduct $Id(\mathbb{A})$ is $Inv(Id(\mathbb{A}))$ equal to the $(Inv(\mathbb{A}) \cup R_u^A)^\star$.

The next basic stone will be a relation R_{Gr}^A for a finite algebra. Assume a surjective finite algebra $\mathbb{A}$ on a set $A = \{a_1, a_2, \ldots, a_k\}$. From the above remark 3.4.3 we know that unary surjective term operations form a permutation group.

Definition 3.5.2. Denote the above group as Gr and define a relation $R_{Gr}^A = \{(g(a_1), g(a_2), \ldots, g(a_k)); g \in Gr\}$.

The relation is defined by surjective unary term operations. It holds that any n-ary operation f is in $Pol(R_{Gr}^A)$ if and only if $f(g_1, \ldots, g_n)$ is again a unary surjective term operation. Applying the operation f on any n elements from the relation R_{Gr}^A gives $f(g_1, \ldots, g_n)(a_1, \ldots, a_k)$. If the operation f is a polymorphism, the composition $f(g_1, \ldots, g_n)$ has to be an element of G as only $g(a_1, \ldots, a_k)$ are in the relation. Conversely if $f(g_1, \ldots, g_n)$ is an element of G, then $f(g_1, \ldots, g_n)(a_1, \ldots, a_k) = g(a_1, \ldots, a_n)$ which is in the relation from the definition.

To show that $f(R_{Gr}^A)$ is a subset of R_{Gr}^A we have to show that for any n-ary operation f from F follows $f(g_1, \ldots, g_n)$ is in G. Since the $Term$ is a clone, $f(g_1, \ldots, g_n)$ is again a term operation and as g_i is a surjective unary operation, it is also surjective unary term operation and thus in G.

Therefore R^A_{Gr} is in $Inv(f)$ for any term operation which yields our aim since $Inv(Term(\mathbb{A})) = Inv(Pol(Inv(\mathbb{A}))) = Inv(\mathbb{A})$ is from the Galois correspondece.

Let us look at invariants. As $Inv(\mathbb{A})$ is equal to $Inv(Term(\mathbb{A}))$ and idempotent term operations are just a subset of term operations, $Inv(\mathbb{A}) \subseteq Inv(Id(\mathbb{A}))$. From the first point of the lemma also $R^A_u \subseteq Inv(Id(\mathbb{A}))$. Assume a relation R from $Inv(Id(\mathbb{A}))$. If we take an operation f from $Term(\mathbb{A})$ it is not necessary that $f(R)$ has image in R however we can change the relation(extend it,...) and then it will be in $Inv(Term(\mathbb{A}))$. Applying a basic operation from the algebra on the relation gives a tuple from the relation.

In the last part we will look at a reduction between problems but at first we will state the above notes in the form of a remark for sumarizing them. Proof will follow from the above text.

Remark 3.5.3.

1. *Any operation on the set A is idempotent if and only if it maps any relation from the set R^A_u to the same relation.*
2. *For the set of invariants of a full idempotent reduct of an algebra $\mathbb{A}$ holds equality: $Inv(Id(\mathbb{A})) = (Inv(\mathbb{A}) \cup R^A_u)^\star$.*
3. *A relation R^A_{Gr} is in $Inv(\mathbb{A})$.*

Proof. Follows from the above observations. □

Lemma 3.5.4. *For a finite surjective algebra $\mathbb{A}$, $CSP((Inv(\mathbb{A}) \cup R^A_u)$ is polynomial time equivalent to $CSP(Inv(\mathbb{A}))$.*

Proof. We will at first divide our proof into two parts depending on the way of equivalence. At first we will show reduction from $CSP(Inv(\mathbb{A}))$ and after that we will show the second direction.

- Since $Inv(\mathbb{A})$ is a subset of $Inv(\mathbb{A}) \cup R^A_u$ and Pol changes inclusions, from the lemma B about polynomial time reduction, $CSP(Inv(\mathbb{A}))$ is reducible to $CSP((Inv(\mathbb{A}) \cup R_u)$ in polynomial time.
- For the second direction we will assume a concrete problem $P = (\mathcal{G}, \mathcal{H})$ from $CSP((Inv(\mathbb{A}) \cup R_u)$. Our task is to define a new problem $P' = (\mathcal{G}', \mathcal{H}')$.

As a set G' we will take the union of the set G and a set of new variables whose size is the same as the size of the set A. We will now modify the original structures. If we should assign a unary tuple (b) to the unary relation consisting of one tuple (a) in the original problem P we replace them with a binary tuple consisting of the element b and a new variable.

We would like to assign this binary tuple to a tuple from the binary equality relation, i.e. to the tuple (a, a). Other relations in structures will stay unchanged. Finally we will add to the new problem P' a tuple consisting of all new variables in a fixed order and we would like to assign them to a tuple in the relation R_{Gr} in the same order.

This procedure can be made up in polynomial time. Since the relation R^A_{Gr} is in $Inv(\mathbb{A})$ and non-replaced relations were also from $Inv(\mathbb{A})$ we need to show that the equality relation is also in $Inv(\mathbb{A})$. Let us assume a k-ary basic operation f from the algebra $\mathbb{A}$ and apply it to k tuples from the equality relations:

$$f((b_1, b_1), \ldots, (b_k, b_k)) = (f(b_1, b_2, \ldots, b_k), f(b_1, b_2, \ldots, b_k)) = (c, c).$$

Evidently this is a binary tuple from the equality relation. Thus the new problem P' is from $CSP(\mathcal{H}')$ where $\mathcal{H}' \subseteq Inv(\mathbb{A})$.

Another part of the proof shows that we really have a reduction. We have to show that both problems have the same sets of solutions.

- Assume that φ is a solution of P'. From the definition of the relation R^A_{Gr} and the condition that the tuple of all new variables should be assigned to a tuple in the relation R^A_{Gr}, the solution φ maps a new variable v_a to the element $g(a)$ where we denote new variables with the index of an element to whose permutation it is assigned. This property holds for any element a from the set A.

  Basic property of any group is that it always contains an inversion. Since g is in Gr, also g^{-1} is in Gr and thus g^{-1} is a unary surjective term operation. We know that g^{-1} is a term operation if and only if g^{-1} is in $Pol(Inv(\mathbb{A}))$. Therefore $g^{-1}\varphi$ is also a solution since $g^{-1}\varphi(a) = g^{-1}(b) = (c)$ where a is a tuple which should be assigned to a tuple in the relation . Since φ is a solution, tuple b is in the relation and since g^{-1} is a term operation, tuple c is also in the relation.

Therefore $g^{-1}\varphi$ is also a solution to P' and the equivalence for any new variable v_a $g^{-1}\varphi(v_a) = a$ does not violate it. Denote this solution as ψ and let us look if it is a solution to the problem P.

The homomorphism ψ is a solution to the problem P since it is a solution to the problem P' and does not violate the condition where a tuple (b) should be assign to one element relation $\{(a)\}$ since it assigns tuple (b, v_a) to the equality relation and assigns v_a to the value a. We will look now on the second direction.

Let us have a homomorphism ψ' which is a solution to the problem P and extend it in a way that $\psi'(v_a) = a$. Then it is a solution to the problem P' as we have seen in the above section.

We have shown polynomial time reduction from $CSP((Inv(\mathbb{A}) \cup R_u)$ to $CSP(\mathcal{H}')$ and with use of lemma A we have polynomial time reduction to $CSP(Inv(\mathbb{A}))$.

□

We are now ready to show the main theorem about the tractability of an algebra and a full idempotent reduct.

Theorem 3.5.5. *A finite surjective algebra is tractable,* **NP***-complete respective, if and only if its full idempotent reduct is tractable,* **NP***-complete respective.*

Proof. Let us assume a finite surjective algebra $\mathbb{A}$ and its full idempotent reduct $Id(\mathbb{A})$. Assume a structure $\mathcal{B}$ from $Inv(Id(\mathbb{A}))$. From the above remark 3.5.3 we know that $\mathcal{B}$ is in $(Inv(\mathbb{A}) \cup R_u^A)^\star$. Therefore from the lemma A about logarithmic-space reduction we can reduce $CSP(\mathcal{B})$ to $CSP(Inv(\mathbb{A}) \cup R_u^A)$ in logarithmic space.

We will show both ways of the theorem. At first let us assume that the algebra $\mathbb{A}$ is tractable.

- If the algebra is tractable, $CSP(\mathcal{A})$ is tractable for any structure $\mathcal{A}$ from $Inv(\mathbb{A})$ and thus also $CSP(Inv(\mathbb{A}))$ is tractable. We can reduce $CSP(\mathcal{B})$ to $CSP(Inv(\mathbb{A}) \cup R_u^A)$ and then to tractable $CSP(Inv(\mathbb{A}))$ in polynomial time which follows from the above lemma. Thus from the section about time and space complexity, $CSP(\mathcal{B})$ is tractable. Therefore $Id(\mathbb{A})$ is tractable.

- For the second direction let us suppose that $CSP(\mathcal{B})$ is tractable for any structure $\mathcal{B}$ from $Inv(Id(\mathbb{A}))$ which means that also $CSP(Inv(Id(\mathbb{A})))$ is tractable. From the above lemma we know that $CSP(Inv(Id(\mathbb{A})))$ is equal to $CSP((Inv(\mathbb{A}) \cup R_u^A)^\star)$. We can reduce problem $CSP((Inv(\mathbb{A}) \cup R_u^A))$ to the problem $CSP((Inv(\mathbb{A}) \cup R_u^A)^\star)$ in polynomial time since $(Inv(\mathbb{A}) \cup R_u^A)$ is a subset of $(Inv(\mathbb{A}) \cup R_u^A)^\star$, Pol changes inclusions and thus we can apply lemma B about polynomial time reduction.

  Since $\mathcal{A}$ is a substructure in $Inv(\mathbb{A})$ from the lemma B about reduction $CSP(\mathcal{A})$ is polynomial time reducible to $CSP(Inv(\mathbb{A}))$ which is again reducible to tractable $CSP(Inv(\mathbb{A}) \cup R_u^A)$ in polynomial time from the above lemma. Thus from the section about time and space complexity, $CSP(\mathcal{A})$ is tractable and therefore the algebra $\mathbb{A}$ is tractable.

For **NP**-completness let us assume that the algebra $\mathbb{A}$ is **NP**-complete with **NP**-complete $CSP(\mathcal{A})$. From the second point of showing tractability we can reduce $CSP(\mathcal{A})$ to the problem $CSP(Inv(Id(\mathbb{A})))$ which has to be therefore **NP**-complete and thus also the algebra $Id(\mathbb{A})$ is **NP**-complete.

For the second direction let us assume that the algebra $Id(\mathbb{A})$ is **NP**-complete with the **NP**-complete problem $CSP(\mathcal{B})$. From the first part of showing tractability we can reduce $CSP(\mathcal{B})$ to the problem $CSP(Inv(\mathbb{A}))$ which therefore has to be **NP**-complete and thus also the algebra $\mathbb{A}$ is **NP**-complete. □

Chapter 4

Appendix

You can find here an algorithm for strong consistency. For more material and proof of its complexity look at [14] Cooper: An Optimal k-Consistency Algorithm. Its complexity is polynomial.

Algorithm (Strong k-consistency)**.** *Input: problem* $P = (\mathcal{G}, \mathcal{H})$

1. *Step 1: initialization*

   LIST:=EmptyList;
   R:=pair of corresponding relations $R^{\mathcal{G}}, R^{\mathcal{H}}$*;*
   M:=0;
   C:=$|H|$*;*
   for $i := 1$ *to* k *do*

   - *for every ordered(<) i-tuple* G_i *of different elements from the set* G *do*
     begin
     - *for every i-tuple* H_i *of elements from the set* H *do*
       begin
       - *if* H_i *is not a locally consistent assignment to* G_i *then*
         begin
         - *Append(LIST, (*G_i, H_i, i*));*
         - *M[*G_i, H_i*]:=1;*

         end;

       end

     end

2. *Step 2*

while LIST $\neq$ *EmptyList do*

begin

- *Remove an element* (G_i, H_i, i) *from LIST;*
- *if* $i < k$ *then for each* $g \notin G_i, h \in H$ *do*
  begin
  - $G_{i+1} :=$ *ordered* $(i+1)$*-tuple made from* G_i *and* g*;*
  - $H_{i+1} :=$ *tuple made from* H_i *and* h *where* h *is put at the same position as* g *in* G_{i+1}
  - *if* $M[G_{i+_1}, H_{i+1}]$*=0 then*
    begin
    - *Append(LIST,* $(G_{i+1}, H_{i+1}, i+1)$*);*
    - $M[G_{i+1}, H_{i+1}]$*:=1;*
    - *we set that* G_{i+1} *and* H_{i+1} *does not form a locally consistent assignment*

    end

  end
- *if* $i > 1$ *then for* $j := 1$ *to* i *do*
  begin
  - *assume that* $G_i = \{g_1, g_2, \ldots, g_i\}, H_i = \{h_1, h_2, \ldots, h_i\}$
  - $G_{i-1} := G_i - g_j$
  - $H_{i-1} := H_i - h_j$
  - $C[G_{i-1}, H_{i-1}, g_j]--$
  - *If* $C[G_{i-1}, H_{i-1}, g_j]$*=0 and* $M[G_{i-1}, H_{i-1}]$*=0 then*
    begin
    - *Append(LIST,*$(G_{i-1}, H_{i-1}, i-1)$*);*
    - $M[G_{i-1}, H_{i-1}]$*:=1;*
    - *we set that* G_{i-1} *and* H_{i-1} *does not form a locally consistent assignment*

    end

  end

end

Bibliography

[1] Garey M. R., Johnson D. S.: *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and co., New York, 1979.

[2] Larose B., Zádori L.: *Bounded width problems and algebras*, Algebra Universalis, Vol. 56, No. 3-4 (2006) 439-466.

[3] Jeavons P.:*On the algebraic structure of combinatorial problems*, Theoretical Computer Science, Vol. 200, I. 1-2 (1998) 185-204.

[4] Jeavons P., Cohen D., Gyssens M.: *Closure Properties of Constraints*, JACM, Vol. 44, No. 4 (1997) 527-548.

[5] Feder T., Vardi M.Y.: *The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory*, SIAM Journal on Computing, Vol. 28, No. 1 (1999) 57-104.

[6] Bulatov A. A., Valeriote M. A.: *Recent Results on the Algebraic Approach to the CSP, in: Complexity of Constraints: An Overview of Current Research Themes 69-92*, Springer-Verlag, Heidelberg, 2008.

[7] Bulatov A. A., Jeavons P., Krohkin A.: *Classifying the Complexity of Constraints Using Finite Algebras*, SIAM Journal on Computing, Vol. 34. No. 3 (2005) 720-742.

[8] Dechter R.: *From local to global consistency*, Artifical Intelligence, Vol. 55, No. 1 (1992) 87-107.

[9] Rosenberg I. G.: *Minimal clones I: the five types, in : Lectures in Universal algebra 405-427*, North-Holland, Amsterdam, 1986.

[10] Szendrei Á.: *Clones in Universal Algebra*, Seminaires de Mathematiques Superieures, Vol. 99, University of Montreal, Montreal, 1986.

[11] Furst M., Hopcroft J., Luks E.: *Polynomial-time algorithms for permutation groups, in: Proceedings of the 21st Annual Symposium on Foundations of Computer Science 36-41*, IEEE Computer Society, Washington, 1980.

[12] Hoffmann C. M.: *Group-Theoretic Algorithms and Graph Isomorphism, Cha. 2*, Lecture Notes in Computer Science, Vol. 136, Springer-Verlag, New York, 1982.

[13] Jeavons P., Cohen D., Gyssens M.: *A Unifying Framework for Tractable Constraints*, Lecture Notes In Computer Science, Vol. 976 (1995) 276-291.

[14] Cooper M. C.: *An optimal k-consistency algorithm*, Artificial Intelligence, Vol. 41, No. 1 (1989) 89-95.

[15] Krohkin A.: *Universal Algebra and CSP: The Basics*, lecture notes Durham University, UASCP 2007.