Chapter 6

The Complexity of Constraint Languages

David Cohen & Peter Jeavons

6.1 Introduction and Outline

One of the most fundamental challenges in constraint programming is to understand the computational complexity of problems involving constraints. It has been shown that the class of all constraint satisfaction problem instances is NP-hard [71], so it is unlikely that efficient general-purpose algorithms exist for solving all forms of constraint problem. However, in many practical applications the instances that arise have special forms that enable them to be solved more efficiently [11, 25, 69, 82].

One way in which this occurs is that there is some special structure in the way that the constraints overlap and intersect each other. The natural theory for discussing the structure of such interaction between constraints is the mathematical theory of hypergraphs. Much work has been done in this area, and many tractable classes of constraint problems have been identified based on structural properties (see Chapter 5). There are strong parallels between this work and similar investigations into the structure of so-called *conjunctive queries* in relational databases [41, 58].

Another way in which constraint problems can be defined which are easier to solve than in the general case is when the *types of constraints* are limited. The natural theory for discussing the properties of constraint types is the mathematical theory of relations and their associated algebras. Again considerable progress has been made in this investigation over the past few years. For example, a complete characterisation of tractable constraint types is now known for both 2-element domains [85] and 3-element domains [14]. In addition, a number of novel efficient algorithms have been developed for solving particular types of constraint problems over both finite and infinite domains [3, 8, 16, 25, 26, 28, 63].

In this chapter we will focus on the second approach. That is, we will investigate how the complexity of solving constraint problems varies with the types of constraints which are allowed. One fundamental open research problem in this area is to characterise exactly which types of constraints give rise to constraint problems which can be solved

in polynomial time. This problem is important from a theoretical perspective, because it helps to clarify the boundary between tractability and intractability in a wide range of combinatorial search problems [27, 37, 49, 62]. It is also important from a practical perspective, as it allows the development of constraint programming languages which exploit the existence of diverse families of tractable constraints to provide more efficient solution techniques [69, 82].

In this chapter a set of types of constraints will be called a *constraint language*. Section 6.2 gives the basic definitions, and Section 6.3 lists some typical examples of tractable (and intractable) constraint languages.

In Section 6.4 we present the mathematical theory that leads us to the major results in the area: we will characterise the complexity of constraint languages (over finite domains) in terms of properties of associated finite algebras.

In Section 6.5 we show how the algebraic theory can be used to identify tractable constraint languages and select an appropriate algorithm. This section presents a strong conjecture for a simple algebraic characterisation of all tractable constraint languages. We will also show that a direct result of the theory is that if the decision problem for a constraint language can be solved in polynomial time, then so can the search problem. In other words, for any language for which it can be decided in polynomial time whether a solution exists, a solution can be found in polynomial time.

In Section 6.6 we consider how the algebraic theory can be extended to deal with constraint languages over infinite domains, and in Section 6.7 we consider multi-sorted constraint languages (where different variables can take their values from different sets).

Finally, in Section 6.8 we briefly consider some alternative approaches, including a constructive approach which builds new tractable constraint languages by combining simpler languages. This theory applies to constraint languages over both finite and infinite domains. This constructive approach has a rather different flavour from the more descriptive algebraic approach, and the two approaches have not yet been fully unified.

We conclude the chapter in Section 6.9 with a discussion of possible future work in this exciting area.

6.2 Basic Definitions

In this section we begin by defining the fundamental decision problem associated with any given constraint language. It is the complexity of this decision problem that is the main focus of this chapter.

The central notion in the study of constraints and constraint satisfaction problems is the notion of a *relation*.

Definition 1. *For any set D, and any natural number n, the set of all n-tuples of elements of D is denoted by D^n. The ith component of a tuple t will be denoted by $t[i]$.*

*A subset of D^n is called an n-ary **relation** over D. The set of all finitary relations over D is denoted by $\mathbf{R}_D$.*

*A **constraint language** over D is a subset of $\mathbf{R}_D$.*

The 'constraint satisfaction problem' was introduced by Montanari [75] in 1974 and has been widely studied [33, 37, 65, 71, 72, 73] (see Chapter 2). In this chapter we study a parameterised version of the standard constraint satisfaction problem, in which the parameter is a constraint language specifying the possible forms of the constraints.

Definition 2. *For any set D and any constraint language Γ over D, the **constraint satisfaction problem** $\mathrm{CSP}(\Gamma)$ is the combinatorial decision problem with*

Instance: *A triple $\langle V, D, \mathcal{C}\rangle$, where*

- *V is a set of **variables**;*
- *$\mathcal{C}$ is a set of **constraints**, $\{C_1, \ldots, C_q\}$.*
- *Each constraint $C_i \in \mathcal{C}$ is a pair $\langle s_i, R_i\rangle$, where*
  - *s_i is a tuple of variables of length n_i, called the **constraint scope**;*
  - *$R_i \in \Gamma$ is an n_i-ary relation over D, called the **constraint relation**.*

Question: *Does there exist a **solution**, that is, a function φ, from V to D, such that, for each constraint $\langle s, R\rangle \in \mathcal{C}$, with $s = \langle v_1, \ldots, v_n\rangle$, the tuple $\langle \varphi(v_1), \ldots, \varphi(v_n)\rangle$ belongs to the relation R?*

The set D, specifying the possible values for the variables, is called the **domain** of the problem. The set of solutions to a CSP instance $\mathcal{P} = \langle V, D, C\rangle$ will be denoted $\mathrm{Sol}(\mathcal{P})$.

In order to determine the computational complexity of a constraint satisfaction problem we need to specify how instances are encoded as finite strings of symbols. The *size* of a problem instance can be taken to be the length of a string specifying the variables, the domain, all constraint scopes and corresponding constraint relations. We shall assume in all cases that this representation is chosen so that the complexity of determining whether a constraint allows a given assignment of values to the variables in its scope is bounded by a polynomial function of the length of the representation. For finite domains it is most straightforward to assume that the tuples in the constraint relations are listed explicitly (although in practice the constraint relations are likely to be defined implicitly).

Throughout the chapter we shall be concerned with distinguishing between constraint languages which give rise to tractable problems (i.e., problems for which there exists a polynomial-time solution algorithm) and those which do not. To ensure that tractability does not depend on the way in which the relations are encoded, we define the notion of a tractable constraint language in a way that depends on finite subsets only.

Definition 3. *A constraint language, Γ, is said to be **tractable** if $\mathrm{CSP}(\Gamma')$ can be solved in polynomial time, for each finite subset $\Gamma' \subseteq \Gamma$.*

*A constraint language, Γ, is said to be **NP-complete** if $\mathrm{CSP}(\Gamma')$ is NP-complete, for some finite subset $\Gamma' \subseteq \Gamma$.*

There are known to be infinitely many computational problems which are neither solvable in polynomial time nor NP-complete [66], but we shall see below that all constraint languages over domains of size 2 and 3 are known to be either tractable or NP-complete. The same dichotomy is conjectured to hold for all constraint languages over any finite domain (see Conjecture 52 below), although this question is still open [11, 37].

6.3 Examples of Constraint Languages

This section introduces some typical constraint languages that we will be concerned with in this chapter. For each language mentioned we simply state in this section whether it is known to be tractable or NP-complete. A more detailed discussion of many of these languages can be found later in the chapter.

Example 4. Let D be any *field* (that is, a set on which the operations of addition, subtraction, multiplication and division are defined, such as the rational numbers). Let Γ_{LIN} be the constraint language consisting of all those relations over D which consist of all the solutions to some system of *linear equations* over D.

Any relation from Γ_{LIN}, and therefore any instance of $\text{CSP}(\Gamma_{\text{LIN}})$, can be represented by a system of linear equations[1] over D, and so can be solved in polynomial time (e.g., by Gaussian elimination). Hence Γ_{LIN} is a tractable constraint language. □

Example 5. A constraint language over a two-element set $D = \{d_0, d_1\}$ is known as a **Boolean** constraint language. Using such languages we can express the standard propositional SATISFIABILITY problem [38, 77] as a constraint satisfaction problem, by identifying the 2 elements of D with the logical values TRUE and FALSE.

It was established by Schaefer in 1978 [85] that a Boolean constraint language, Γ, is tractable if (at least) one of the following six conditions holds:

1. Every relation in Γ contains a tuple in which all entries are equal to d_0;
2. Every relation in Γ contains a tuple in which all entries are equal to d_1;
3. Every relation in Γ is definable by a conjunction of clauses, where each clause has at most one negative literal;
4. Every relation in Γ is definable by a conjunction of clauses, where each clause has at most one positive literal (i.e., a conjunction of **Horn clauses**);
5. Every relation in Γ is definable by a conjunction of clauses, where each clause contains at most 2 literals;
6. Every relation in Γ is the set of solutions of a system of linear equations over the finite field with 2 elements, GF(2).

In all other cases Γ is NP-complete.

This result establishes a *dichotomy* for Boolean constraint languages: any Boolean constraint language is either tractable or NP-complete. Hence this result is known as **Schaefer's Dichotomy Theorem** [85].

Similar dichotomy results have also been obtained for many other combinatorial problems over a Boolean domain which are related to the Boolean constraint satisfaction problem [62, 27]. □

Example 6. It follows from Schaefer's Dichotomy Theorem [85] (Example 5) that some Boolean constraint languages containing just a *single relation* are NP-complete.

For example, for any 2-element set $D = \{d_0, d_1\}$, let N_D be the ternary **not-all-equal** relation over D defined by

$$\begin{aligned} N_D &= D^3 \setminus \{\langle d_0, d_0, d_0\rangle, \langle d_1, d_1, d_1\rangle\} \\ &= \{\langle d_0, d_0, d_1\rangle, \langle d_0, d_1, d_0\rangle, \langle d_0, d_1, d_1\rangle, \langle d_1, d_0, d_0\rangle, \langle d_1, d_0, d_1\rangle, \langle d_1, d_1, d_0\rangle\}. \end{aligned}$$

The problem $\text{CSP}(\{N_D\})$ corresponds to the NOT-ALL-EQUAL SATISFIABILITY problem [85] which is known to be NP-complete[2].

[1] Moreover, this system of equations can be computed from the relations in polynomial time - see [11]

[2] The standard version of NOT-ALL-EQUAL SATISFIABILITY given in [38, 77] is slightly more general, but can be shown to be polynomial-time equivalent to $\text{CSP}(\{N_D\})$.

Similarly, let T_D be the ternary **one-in-three** relation over D defined by

$$T_D = \{\langle d_0, d_0, d_1\rangle, \langle d_0, d_1, d_0\rangle, \langle d_1, d_0, d_0\rangle\}.$$

The problem $\mathrm{CSP}(\{T_D\})$ corresponds to the ONE-IN-THREE SATISFIABILITY problem (with positive literals) [85, 38, 27] which is known to be NP-complete. □

Example 7. The class of constraints known as **max-closed** constraints was introduced in [54] and shown to be tractable. This class of constraints has been used in the analysis and development of a number of industrial scheduling tools [69, 82].

Max-closed constraints are defined in [54] for arbitrary finite domains which are totally ordered. This class of constraints includes all of the 'basic constraints' over the natural numbers in the constraint programming language CHIP [89], as well as many other forms of constraint. The following are examples of max-closed constraints over a domain D which can be any fixed finite set of natural numbers:

$$\begin{gathered} 3x_1 + x_5 + 3x_4 \geq 2x_2 + 10, \\ 4x_1 \neq 8, \\ x_1 \in \{1, 2, 3, 5, 7, 11, 13\}, \\ 2x_1x_3x_5 \geq 3x_2 + 1, \\ (3x_1 \geq 7) \vee (2x_1 \geq 4) \vee (5x_2 \leq 7). \end{gathered}$$

Hence the constraint language comprising all relations of these forms is tractable. □

Example 8. Let D be any finite set, and let Γ_{ZOA} be the set of all relations of the following forms:

- All unary relations;
- All binary relations of the form $D_1 \times D_2$ for subsets D_1, D_2 of D;
- All binary relations of the form $\{\langle d, \pi(d)\rangle \mid d \in D_1\}$, for some subset D_1 of D and some permutation π of D;
- All binary relations of the form $\{\langle a, b\rangle \in D_1 \times D_2 \mid a = d_1 \vee b = d_2\}$ for some subsets D_1, D_2 of D and some elements $d_1 \in D_1, d_2 \in D_2$.

These relations were introduced in [26], where they are called **0/1/all relations**.

It was shown in [26] that Γ_{ZOA} is tractable, and that for any binary relation R over D which is *not* in Γ_{ZOA}, $\Gamma_{\mathrm{ZOA}} \cup \{R\}$ is NP-complete. □

Example 9. The class of binary constraints known as **connected row-convex** constraints was introduced in [35] and shown to be tractable. This class properly includes the 'monotone' relations, identified and shown to be tractable by Montanari in [75].

Let the domain D be the ordered set $\{d_1, d_2, \ldots, d_m\}$, where $d_1 < d_2 < \cdots < d_m$. The definition of connected row-convex constraints given in [35] uses a standard matrix representation for binary relations: the binary relation R over D is represented by the $m \times m$ 0-1 matrix M, by setting $M_{ij} = 1$ if the relation contains the pair $\langle d_i, d_j\rangle$, and $M_{ij} = 0$ otherwise.

A relation is said to be connected row-convex if the following property holds: the pattern of 1's in the matrix representation (after removing rows and columns containing only

0's) is connected along each row, along each column, and forms a connected 2-dimensional region (where some of the connections may be diagonal).

Here are some examples of connected row-convex relations:

$$\begin{pmatrix} 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&1&1&0&0&0\\ 0&0&0&1&1&1&1&0&1&0\\ 0&1&1&1&1&1&1&0&1&0\\ 1&1&1&1&1&1&1&0&1&1\\ 0&1&1&1&1&1&1&0&1&0\\ 0&0&1&1&1&1&1&0&1&0\\ 0&0&1&1&1&1&1&0&1&0\\ 0&0&0&1&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0 \end{pmatrix} \quad \begin{pmatrix} 1&1&0&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0&0&0&0\\ 0&0&1&1&1&0&0&0&0&0\\ 0&0&1&1&1&0&0&0&0&0\\ 0&0&1&1&1&0&0&0&0&0\\ 0&0&0&0&0&1&1&1&1&1\\ 0&0&0&0&0&1&1&1&1&1\\ 0&0&0&0&0&1&1&1&0&0\\ 0&0&0&0&0&1&1&1&0&0 \end{pmatrix} \quad \begin{pmatrix} 0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&1&1&1\\ 0&0&0&0&0&0&1&1&1&0\\ 0&0&0&0&1&1&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0\\ 1&1&1&1&0&0&0&0&0&0\\ 0&1&1&1&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0&0&0 \end{pmatrix}$$

□

Example 10. The binary **inequality** relation over an ordered set D is defined as follows:

$$<_D = \{\langle d_1, d_2\rangle \in D^2 \mid d_1 < d_2\}.$$

When D is the set of natural numbers, $\mathbb{N}$, the class of constraint satisfaction problem instances $\mathrm{CSP}(\{<_D\})$ corresponds to the ACYCLIC DIGRAPH problem [4]. An instance of this problem is a directed graph G, and the question is whether G is *acyclic*, that is, contains no directed cycles. Note that a directed graph is acyclic if and only if its vertices can be numbered in such a way that every arc leads from a vertex with smaller number to a vertex with a greater one.

Since the ACYCLIC DIGRAPH problem is tractable, it follows that $\{<_{\mathbb{N}}\}$ is a tractable constraint language. □

Example 11. The binary **disequality** relation over a set D is defined as follows:

$$\neq_D = \{\langle d_1, d_2\rangle \in D^2 \mid d_1 \neq d_2\}.$$

The class of constraint satisfaction problem instances $\mathrm{CSP}(\{\neq_D\})$ corresponds to the GRAPH COLORABILITY problem [38, 77] with $|D|$ colours. This problem is tractable when $|D| \leq 2$ or $|D| = \infty$, and NP-complete when $3 \leq |D| < \infty$. □

Example 12. The ternary **betweenness** relation over an ordered set D is defined as follows:

$$B_D = \{\langle x, y, z\rangle \in D^3 \mid x < y < z \text{ or } x > y > z\}.$$

For a finite set D, the constraint language $\{B_D\}$ is tractable when $|D| \leq 4$ and is NP-complete when $|D| \geq 5$ (see Example 45).

For an infinite set D, the constraint language $\{B_D\}$ is NP-complete because the class of constraint satisfaction problem instances $\mathrm{CSP}(\{B_D\})$ corresponds to the BETWEENNESS problem, which is known to be NP-complete [38]. An instance of this problem is a pair $\langle A, T\rangle$ where A is a finite set and $T \subseteq A^3$; the question is whether there is a function $f : A \to \{1, \ldots, |A|\}$ such that, for every triple $\langle a, b, c\rangle \in T$, we have either $f(a) < f(b) < f(c)$ or $f(a) > f(b) > f(c)$. □

Example 13. The class of constraints known as **linear Horn** constraints was introduced in [55, 61] and shown to be tractable.

The constraint relation of a linear Horn constraint is a relation over an infinite ordered set which is specified by a disjunction of an arbitrary finite number of linear disequalities and at most one weak linear inequality. The following are examples of linear Horn constraints:

$$3x_1 + x_5 - 3x_4 \leq 10,$$
$$x_1 + x_3 + x_5 \neq 7,$$
$$(3x_1 + x_5 - 4x_3 \leq 7) \vee (2x_1 + 3x_2 - 4x_3 \neq 4) \vee (x_2 + x_3 + x_5 \neq 7),$$
$$(4x_1 + x_3 \neq 3) \vee (5x_2 - 3x_5 + x_4 \neq 6).$$

Linear Horn constraints are an important class of linear constraints for expressing problems in temporal reasoning [55]. In particular, the class of linear Horn constraints properly includes the point algebra of [90], the (quantitative) temporal constraints of [59, 60] and the ORD-Horn constraints of [76]. □

6.4 Developing an Algebraic Theory

A series of papers by Jeavons and co-authors [50, 51, 52, 54] has shown that the complexity of constraint languages over a finite domain can be characterised using algebraic properties of relations (see Figure 6.1).

The first step in the algebraic approach to constraint languages exploits the well-known idea that, given an initial set of constraint relations, there will often be further relations that can be added to the set without changing the complexity of the associated problem class. In fact, it has been shown that it is possible to add all the relations that can be derived from the initial relations using certain simple rules. The larger sets of relations obtained using these rules are known as *relational clones* [34, 80]. Hence the first step in the analysis is to note that it is sufficient to analyse the complexity only for those sets of relations which are relational clones (see Section 6.4.1).

The next step in the algebraic approach is to note that relational clones can be characterised by their *polymorphisms*, which are algebraic *operations* on the same underlying set [49, 52] (see Section 6.4.2). As well as providing a convenient and concise method for describing large families of relations, the polymorphisms also reflect certain aspects of the structure of the relations that can be used for designing efficient algorithms. This link between relational clones and polymorphisms has already played a key role in identifying many tractable constraint classes and developing appropriate efficient solution algorithms for them [14, 15, 17, 19, 28, 50].

The final step in the algebraic approach links constraint languages with finite universal algebras (see Section 6.4.3). The language of finite algebras provides a number of very powerful new tools for analysing the complexity of constraints, including the deep structural results developed for classifying the structure of finite algebras [45, 74, 87].

6.4.1 Step I: From relations to relational clones

As stated above, the first step in the algebraic approach is to consider what additional relations can be added to a constraint language without changing the complexity of the corresponding problem class. This technique has been widely used in the analysis of Boolean

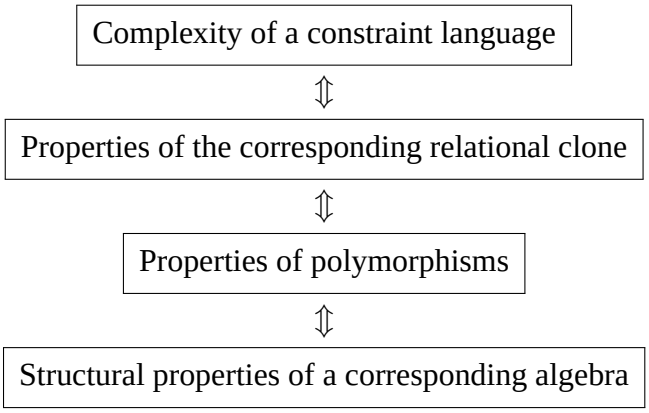

Figure 6.1: Translating questions about the complexity of constraint languages into questions about the properties of algebras.

constraint satisfaction problems [27, 85], and in the analysis of temporal and spatial constraints [36, 76, 83, 63, 64]; it was introduced for the study of constraints over arbitrary finite sets in [49].

Definition 14. *A constraint language* Γ ***expresses*** *a relation* R *if there is an instance* $\mathcal{P} = \langle V, D, C\rangle \in \mathrm{CSP}(\Gamma)$ *and a list* $\langle v_1, \ldots, v_n\rangle$ *of variables in* V *such that*

$$R = \{\langle \varphi(v_1), \ldots, \varphi(v_n)\rangle \mid \varphi \in \mathrm{Sol}(\mathcal{P})\}$$

For any constraint language Γ, the set of all relations which can be expressed by Γ will be called the **expressive power** of Γ. The expressive power of a constraint language Γ can be characterised in a number of different ways [53]. For example, it is equal to the set of all relations that can be obtained from the relations in Γ using the *relational join* and *project* operations from relational database theory [43]. It has also been shown to be equal to the set of relations definable by *primitive positive formulas* over the relations in Γ together with the equality relation, where a primitive positive formula is a first-order formula involving only conjunction and existential quantification [11]. In algebraic terminology [34, 80], this set of relations is called the **relational clone** generated by Γ, and is denoted by $\langle\Gamma\rangle$.

Example 15. Consider the Boolean constraint language $\Gamma = \{R_1, R_2\}$ where $R_1 = \{\langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$ and $R_2 = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle\}$.

It is straightforward to check that all 16 binary Boolean relations can be expressed by a primitive positive formula involving R_1 and R_2. For example, the relation $R_3 = \{\langle 0,0\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$ can be expressed by the formula $R_3 = \exists y R_1(x,y) \wedge R_2(y,z)$. Hence $\langle\Gamma\rangle$, the relational clone generated by Γ, includes all 16 binary Boolean relations.

In fact it can be shown that, for this constraint language Γ, the set $\langle\Gamma\rangle$ consists of precisely those Boolean relations (of any arity) that can be expressed as a conjunction of unary or binary Boolean relations [81, 87]. This is equivalent to saying that the constraint language Γ expresses precisely this set of relations. □

The link between these notions and the complexity of constraint languages is established by the next result.

Theorem 16 ([11, 49]). *For any constraint language Γ and any finite subset $\Gamma_0 \subseteq \langle\Gamma\rangle$ there is a polynomial time reduction from $\mathrm{CSP}(\Gamma_0)$ to $\mathrm{CSP}(\Gamma)$.*

Corollary 17. *A constraint language Γ is tractable if and only if $\langle\Gamma\rangle$ is tractable. Similarly, Γ is NP-complete if and only if $\langle\Gamma\rangle$ is NP-complete.*

This result reduces the problem of characterising tractable constraint languages to the problem of characterising tractable relational clones.

6.4.2 Step II: From relational clones to sets of operations

We have shown in the previous section that to analyse the complexity of arbitrary constraint languages over finite domains it is sufficient to consider only relational clones. This considerably reduces the variety of languages to be studied. However, it immediately raises the question of how to represent and describe relational clones. For many relational clones the only known generating sets are rather sophisticated, and in some cases no generating sets are known.

Very conveniently, it turns out that there is a well-known alternative way to represent and describe any relational clone: using *operations*.

Definition 18. *Let D be a set, and k a natural number. A mapping $f : D^k \to D$ is called a k-ary **operation** on D. The set of all finitary operations on D is denoted by $\mathbf{O}_D$.*

We first describe a fundamental algebraic relationship between operations and relations. First, observe that any operation on a set D can be extended in a standard way to an operation on tuples of elements from D, as follows. For any (k-ary) operation f and any collection of tuples $t_1, \ldots, t_k \in D^n$, define $f(t_1, \ldots, t_k)$ to be the tuple $\langle f(t_1[1], \ldots, t_k[1]), \ldots, f(t_1[n], \ldots, t_k[n])\rangle$.

Definition 19 ([34, 74, 80, 87]). *A k-ary operation $f \in \mathbf{O}_D$ **preserves** an n-ary relation $R \in \mathbf{R}_D$ (or f is a **polymorphism** of R, or R is **invariant** under f) if $f(t_1, \ldots, t_k) \in R$ for all choices of $t_1, \ldots, t_k \in R$.*

For any given sets $\Gamma \subseteq \mathbf{R}_D$ and $F \subseteq \mathbf{O}_D$, we define the mappings Pol and Inv as follows:

$$\mathrm{Pol}(\Gamma) = \{f \in \mathbf{O}_D \mid f \text{ preserves each relation from } \Gamma\},$$
$$\mathrm{Inv}(F) = \{R \in \mathbf{R}_D \mid R \text{ is invariant under each operation from } F\}.$$

We remark that the mappings Pol and Inv form a *Galois correspondence* between $\mathbf{R}_D$ and $\mathbf{O}_D$ (see Proposition 1.1.14 of [80]). Brief introductions to this correspondence can be found in [34, 79], and a comprehensive study in [80]. We note, in particular, that $\mathrm{Inv}(F) = \mathrm{Inv}(\mathrm{Pol}(\mathrm{Inv}(F)))$, for any set of operations F.

It is a well-known result in universal algebra that the relational clone generated by a set of relations over a finite set is determined by the polymorphisms of those relations [80]. Here we will establish this key result using purely constraint-based reasoning.

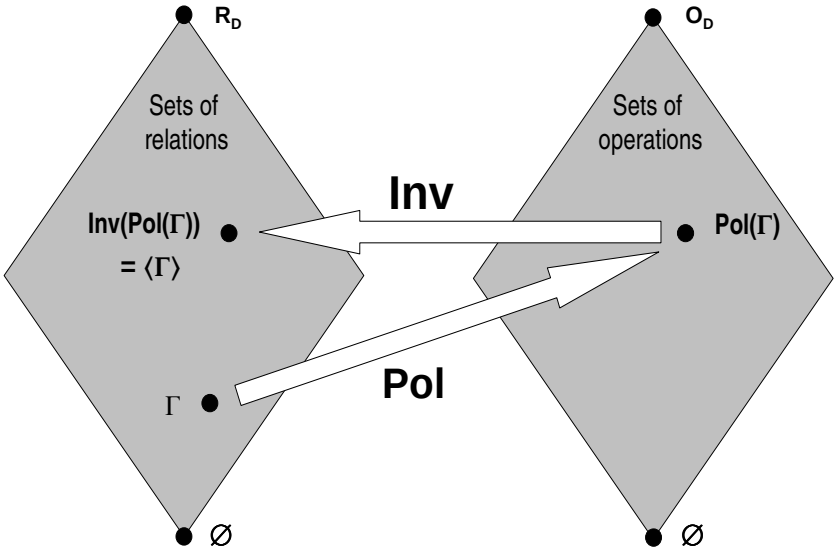

Figure 6.2: The operators Inv and Pol.

Definition 20. *Let Γ be a finite constraint language over a finite set D.*

*For any positive integer k, the **indicator problem of order** k for Γ is the CSP instance $\mathcal{P} = \langle V, D, \mathcal{C}\rangle \in \mathrm{CSP}(\Gamma)$ where:*

- *$V = D^k$ (in other words, each variable in $\mathcal{P}$ is a k-tuple of domain elements).*
- *$\mathcal{C} = \{\langle s, R\rangle \mid R \in \Gamma$ and s matches $R\}$.*

In this definition we say that that a list of k-tuples $s = \langle v_1, \ldots, v_n\rangle$ matches *a relation R if n is equal to the arity of R and for each $i \in \{1, 2, \ldots, k\}$ the n-tuple $\langle v_1[i], \ldots, v_n[i]\rangle$ is in R. Hence the CSP instance $\mathcal{P}$ has constraints from the constraint language Γ on every possible scope which matches a relation from Γ.*

Note that the solutions to the indicator problem of order k for Γ are mappings from D^k to D that preserve each of the relations in Γ, hence they are precisely the k-ary elements of $\mathrm{Pol}(\Gamma)$.

Indicator problems are described in more detail in [48], where a number of concrete examples are given. A software system for constructing and solving indicator problems for given constraint languages is described in [39].

Theorem 21 ([49, 80]). *For any constraint language Γ over a finite set, $\langle\Gamma\rangle = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$.*

Proof. If two relations both have a polymorphism f, then their conjunction also has the polymorphism f. Similarly, if a relation has a polymorphism f, then any relation obtained by existential quantification of that relation also has the polymorphism f. Finally the equality relation has every operation as a polymorphism. It follows from these observations

that for any R in the relational clone of Γ we have $\mathrm{Pol}(\{R\}) \supseteq \mathrm{Pol}(\Gamma)$. Hence $\langle\Gamma\rangle \subseteq \mathrm{Inv}(\mathrm{Pol}(\Gamma))$.

To establish the converse, let Γ be a constraint language over a finite set D, let R be an arbitrary relation in $\mathrm{Inv}(\mathrm{Pol}(\Gamma))$, and let n be the arity of R. We need to show that $R \in \langle\Gamma\rangle$, or in other words that R can be expressed using the constraint language Γ.

Let k be the number of tuples in the relation R, and construct the indicator problem $\mathcal{P}$ of order k for Γ. Choose a list of variables $t = \langle v_1, \ldots, v_n\rangle$ in $\mathcal{P}$ such that each of the n-tuples $\langle v_1[i], \ldots, v_n[i]\rangle$, for $i = 1, \ldots, k$, is a distinct element of our target relation R. Consider the relation $R_t = \{\langle f(v_1), \ldots, f(v_n)\rangle \mid f \in \mathrm{Sol}(\mathcal{P})\}$. By the observation above, the elements of $\mathrm{Sol}(\mathcal{P})$ are the k-ary polymorphisms of Γ, and these include the k projection operations which simply return one of their arguments. By the choice of t, each of these projection operations results in a distinct tuple of R being included in R_t, and so $R \subseteq R_t$. Conversely, by the choice of R, every polymorphism of Γ preserves R, and hence every element of R_t is contained in R. □

Since the relational clone $\langle\Gamma\rangle$ consists of those relations that can be expressed by the constraint language Γ, we immediately obtain the following strong link between polymorphisms and expressive power.

Corollary 22. *A relation R over a finite set can be expressed by a constraint language Γ precisely when $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\{R\})$.*

Combining Theorem 16 and Theorem 21 we obtain the following link between polymorphisms and complexity.

Corollary 23. *For any constraint languages Γ, Γ_0 over a finite set, if Γ_0 is finite and $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\Gamma_0)$, then $\mathrm{CSP}(\Gamma_0)$ is reducible to $\mathrm{CSP}(\Gamma)$ in polynomial time.*

This result implies that, for any finite constraint language Γ over a finite set, the complexity of $\mathrm{CSP}(\Gamma)$ is determined, up to polynomial-time reduction, by the polymorphisms of Γ. Hence we can translate our original problem of characterising tractable constraint languages into an equivalent problem for sets of operations.

Definition 24. *A set of operations $F \subseteq \mathbf{O}_D$ is said to be tractable if $\mathrm{Inv}(F)$ is tractable. A set $F \subseteq \mathbf{O}_D$ is said to be NP-complete if $\mathrm{Inv}(F)$ is NP-complete.*

With this definition we have translated our basic challenge into characterising tractable sets of operations.

6.4.3 Step III: From sets of operations to algebras

We have seen in the previous section that the problem of analysing the complexity of a constraint language can be translated into the problem of analysing the complexity of the set of operations which preserve all of the relations in that language. In this section we shall open the way to the use of a further set of powerful analytical tools by making the final translation step, from sets of operations to algebras.

Definition 25. *An **algebra** is an ordered pair $\mathcal{A} = \langle D, F\rangle$ such that D is a nonempty set and F is a family of finitary operations on D. The set D is called the* universe *of $\mathcal{A}$, and the operations from F are called* basic. *An algebra with a finite universe is referred to as a finite algebra.*

To make the translation from sets of operations to algebras we simply note that any set of operations F on a fixed set D can be associated with the algebra $\langle D, F\rangle$. Hence, we will define what it means for an algebra to be tractable by considering the tractability of the basic operations.

Definition 26. *An algebra $\mathcal{A} = \langle D, F\rangle$ is said to be tractable if the set of basic operations F is tractable. An algebra $\mathcal{A} = \langle D, F\rangle$ is said to be NP-complete if F is NP-complete.*

Our basic task is now translated as: characterise all tractable algebras.

It will be useful to describe an equivalence relation linking algebras that correspond to the same constraint language. As we noted earlier, the mappings Pol and Inv have the property that $\mathrm{Inv}(\mathrm{Pol}(\mathrm{Inv}(F))) = \mathrm{Inv}(F)$, so we can extend a set of operations F to the set $\mathrm{Pol}(\mathrm{Inv}(F))$ without changing the associated invariant relations. The set $\mathrm{Pol}(\mathrm{Inv}(F))$ consists of all operations that can be obtained from the operations in F, together with the set of all projection operations, by forming arbitrary compositions of operations[3]. Note that any set of operations which includes all the projection operations and is closed under composition is referred to by algebraists as a **clone** of operations. The clone of operations obtained from a set F in this way is usually referred to as the set of *term operations* over F, so we will make the following definition.

Definition 27. *For any algebra $\mathcal{A} = \langle D, F\rangle$, an operation f on D will be called a **term operation** of $\mathcal{A}$ if $f \in \mathrm{Pol}(\mathrm{Inv}(F))$.*

The set of all term operations of $\mathcal{A}$ will be denoted $\mathrm{Term}(\mathcal{A})$.

Two algebras with the same universe are called *term equivalent* if they have the same set of term operations. Since, for any algebra $\mathcal{A} = \langle D, F\rangle$, we have $\mathrm{Inv}(F) = \mathrm{Inv}(\mathrm{Term}(\mathcal{A}))$, two algebras are term equivalent if and only if they have the same set of associated invariant relations. It follows that we need to characterise tractable algebras only up to term equivalence.

We will now show that we can restrict our attention to certain special classes of algebras.

The first simplification we apply is to note that any unary polymorphism of a constraint language can be applied to all of the relations in the language without changing the complexity.

Proposition 28 ([52, 49]). *Let Γ be a constraint language over a set D, and let f be a unary operation in $\mathrm{Pol}(\Gamma)$.*

$\mathrm{CSP}(\Gamma)$ is polynomial-time equivalent to $\mathrm{CSP}(f(\Gamma))$, where $f(\Gamma) = \{f(R) \mid R \in \Gamma\}$ and $f(R) = \{f(t) \mid t \in R\}$.

If we apply Proposition 28 with a unary polymorphism f which has the smallest possible range out of all the unary polymorphisms of Γ, then we obtain a constraint language $f(\Gamma)$ whose unary polymorphisms are all surjective. Such a language will be called a **reduced** constraint language.

Definition 29. *We call an algebra **surjective** if all of its term operations are surjective*[4].

[3] If f is an m-ary operation on a set D, and $g_1, g_2, \ldots, g_m$ are k-ary operations on D, then the composition of f and $g_1, g_2, \ldots, g_m$ is the k-ary operation h on D defined by $h(a_1, a_2, \ldots, a_k) = f(g_1(a_1, \ldots, a_k), \ldots, g_m(a_1, \ldots, a_k))$.

[4] *Some authors call an algebra surjective if all of its* basic *operations are surjective. However, such algebras can have non-surjective term operations, so our definition is more restrictive.*

It is easy to verify that a finite algebra is surjective if and only if its unary term operations are all surjective, and hence form a group of permutations. It follows that an algebra $\mathcal{A} = \langle D, F\rangle$ is surjective if and only if $\mathrm{Inv}(F)$ is a reduced constraint language. Using Proposition 28, this means that we can restrict our attention to surjective algebras.

The next theorem shows that for many purposes we need consider only those surjective algebras with the additional property of being idempotent.

Definition 30. *An operation f on D is called **idempotent** if it satisfies $f(x, \ldots, x) = x$ for all $x \in D$.*

The full idempotent reduct *of an algebra $\mathcal{A} = \langle D, F\rangle$ is the algebra $\langle D, \mathrm{Termid}(\mathcal{A})\rangle$, where $\mathrm{Termid}(A)$ consists of all idempotent operations from $\mathrm{Term}(A)$.*

An operation f on a set D is idempotent if and only if it preserves all the relations in the set $\Gamma_{\mathrm{CON}} = \{\{\langle a\rangle\} \mid a \in D\}$, consisting of all unary one-element relations on D. Hence, $\mathrm{Inv}(\mathrm{Termid}(A))$ is the relational clone generated by $\mathrm{Inv}(F) \cup \Gamma_{\mathrm{CON}}$.

That is, considering only the full idempotent reduct of an algebra is equivalent to considering only those constraint languages in which we can arbitrarily fix variables to particular values from the domain.

Theorem 31 ([11]). *A finite surjective algebra $\mathcal{A}$ is tractable if and only if its full idempotent reduct $\mathcal{A}_0$ is tractable. Moreover, $\mathcal{A}$ is NP-complete if and only if $\mathcal{A}_0$ is NP-complete.*

Next we link the complexity of a finite algebra with the complexity of its sub-algebras and homomorphic images. In many cases, we can use these results to reduce the problem of analysing the complexity of an algebra to a similar problem involving an algebra with a smaller universe. In such cases we can reduce the problem of analysing the complexity of a constraint language to a similar problem for a constraint language over a smaller domain.

Definition 32. *Let $\mathcal{A} = \langle D, F\rangle$ be an algebra and U a subset of D such that, for any $f \in F$ and for any $b_1, \ldots, b_k \in B$, where k is the arity of f, we have $f(b_1, \ldots, b_k) \in B$. Then the algebra $\mathcal{B} = \langle B, F|_B\rangle$ is called a **sub-algebra** of $\mathcal{A}$, where $F|_B$ consists of the restrictions of all operations in F to B. If $B \neq A$, then $\mathcal{B}$ is said to be a* proper *sub-algebra.*

Definition 33. *Let $\mathcal{A}_1 = \langle D_1, F_1\rangle$ and $\mathcal{A}_2 = \langle D_2, F_2\rangle$ be such that $F_1 = \{f_i^1 \mid i \in I\}$ and $F_2 = \{f_i^2 \mid i \in I\}$, where both f_i^1 and f_i^2 are k_i-ary, for all $i \in I$.*

*A map $\Phi : A_1 \to A_2$ is called a **homomorphism** from $\mathcal{A}_1$ to $\mathcal{A}_2$ if*

$$f_i^1(a_1, \ldots, a_{k_i}) = f_i^2(\Phi(a_1), \ldots, \Phi(a_{k_i}))$$

holds for all $i \in I$ and all $a_1, \ldots, a_{k_i} \in A_1$.

If the map Φ is surjective, then $\mathcal{A}_2$ is said to be a homomorphic image *of $\mathcal{A}_1$.*

Definition 34. *A homomorphic image of a sub-algebra of an algebra $\mathcal{A}$ is called a **factor** of $\mathcal{A}$.*

Theorem 35 ([11]). *If $\mathcal{A}$ is a tractable finite algebra, then so is every factor of $\mathcal{A}$.*

If $\mathcal{A}$ has any factor which is NP-complete, then $\mathcal{A}$ is NP-complete.

6.5 Applications of the Algebraic Theory

6.5.1 A pre-processing algorithm

The theory described in the previous section has shown that many key properties of a constraint language are determined by its polymorphisms. Hence calculating the polymorphisms of the constraint language used in a given CSP instance can be a useful step in analysing that instance.

For example, using Construction 20 and Proposition 28 we can design a pre-processing algorithm which can sometimes simplify the presentation of a constraint satisfaction problem (Algorithm 1).

Since the indicator problem of order 1 only has $|D|$ variables, this pre-processing step is efficient for many problems and can result in an equivalent problem instance with a considerably smaller domain.

Algorithm 1: Pre-processing to reduce the domain size

Input: An instance $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$ of $\mathrm{CSP}(\Gamma)$ where D is finite.
Output: An equivalent instance $\mathcal{P}'$.

1. Find all unary polymorphisms of Γ by generating and solving the indicator problem of order 1 for Γ;
2. Choose a unary polymorphism f with the smallest number of values in its range;
3. If the range of f is smaller than D, apply f to each constraint relation in $\mathcal{P}$ to obtain a new problem instance $\mathcal{P}'$ over a smaller domain.

6.5.2 Tractable cases: using polymorphisms as algorithm selectors

In many cases, it has been shown that the existence of a single polymorphism satisfying certain simple conditions is sufficient to ensure the tractability of a constraint language and to identify an appropriate polynomial-time algorithm.

Definition 36. *Let f be a k-ary operation a set D.*

- *If $k = 2$ and f satisfies the identities $f(x, f(y, z)) = f(f(x, y), z)$ (associativity), $f(x, y) = f(y, x)$ (commutativity), and $f(x, x) = x$ (idempotency), then f is called a **semilattice** operation.*
- *If f satisfies the identity $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$, then f is called a **conservative** operation.*
- *If $k \geq 3$ and f satisfies the identities $f(y, x, \ldots, x) = f(x, y, x, \ldots, x) = \cdots = f(x, \ldots, x, y) = x$, then f is called a **near-unanimity** operation.*
- *If $k = 3$ and f satisfies the identities $f(y, y, x) = f(x, y, y) = x$, then f is called a **Mal'tsev** operation.*

Proposition 37 ([52]). *For any constraint language* Γ *over a finite set* D*, if* $\mathrm{Pol}(\Gamma)$ *contains a semilattice operation, then* Γ *is tractable, and all instances in* $\mathrm{CSP}(\Gamma)$ *can be solved by enforcing generalised arc consistency*[5].

This result has been extended to more general *semigroup* operations in [12, 31].

Example 38. The Boolean constraint language consisting of all relations that can be specified by **Horn clauses**, as described in Example 5, has the binary polymorphism $\wedge$ (conjunction) [54], and so is tractable by Proposition 37. Any collection of Horn clauses can be solved by *unit resolution*, which is a specialised form of arc consistency. □

Example 39. The max-closed constraints defined in [54] and described in Example 7 all have the binary polymorphism, $\max$, which is a semilattice operation, so they are tractable by Proposition 37. Any collection of max-closed constraints can be solved by enforcing generalised arc consistency. □

Any constraint language which contains *all unary relations* over a finite set has the property that all the operations in $\mathrm{Pol}(\Gamma)$ are conservative, by Definition 19.

Proposition 40 ([16]). *For any constraint language* Γ *over a finite set* D*, if* $\mathrm{Pol}(\Gamma)$ *contains a conservative commutative binary operation, then* Γ *is tractable.*

The algorithm for solving a collection of constraints preserved by a conservative commutative binary operation is based on a generalisation of local consistency techniques [16].

Proposition 41 ([50]). *For any constraint language* Γ *over a finite set* D*, if* $\mathrm{Pol}(\Gamma)$ *contains a* k*-ary near-unanimity operation, then* Γ *is tractable, and all instances in* $\mathrm{CSP}(\Gamma)$ *can be solved by enforcing* k*-consistency, which makes them globally consistent*[6].

In fact, it is shown in [50] that the *only* finite domain languages for which enforcing k-consistency guarantees global consistency are those which have a near-unanimity polymorphism.

Example 42. Let Γ be the Boolean constraint language consisting of all relations that can be specified by clauses with at most 2 literals. This language has the ternary polymorphism, d, given by $d(x, y, z) = (x \wedge y) \vee (y \wedge z) \vee (x \wedge z)$, which is a near-unanimity operation, so Γ is tractable by Proposition 41. A satisfying assignment for any collection of such clauses can be obtained in a backtrack-free way after enforcing path consistency. □

Example 43. The 0/1/all relations defined in [26] and described in Example 8 all have the ternary polymorphism, d, given by $d(x, y, z) = x$ when $y \neq z$ and $d(x, y, z) = y$ otherwise, which is a near-unanimity operation, so they are tractable by Proposition 41. A solution for any collection of 0/1/all constraints can be obtained in a backtrack-free way after enforcing path consistency [26, 50]. □

Example 44. The connected row-convex relations defined in [35] and described in Example 9 all have the ternary polymorphism, m, given by $m(x, y, z) =$ the *median* of x, y and z, which is a near-unanimity operation, so they are tractable by Proposition 41. A solution for any collection of connected row-convex constraints can be obtained in a backtrack-free way after enforcing path consistency [50]. □

[5] See Chapter 3 for a definition of this standard procedure, and a discussion of possible algorithms.

[6] See Chapter 3 for definitions and algorithms.

Example 45. The betweenness relation B_D on an ordered set D, described in Example 12, has a ternary near-unanimity polymorphism when $|D| \leq 4$, so the constraint language containing just this relation is tractable when $|D| \leq 4$, by Proposition 41.

The projection of B_D onto its second co-ordinate is the unary relation containing all elements of D except the largest and smallest. Hence the algebra $\langle D, \mathrm{Pol}(\{B_D\})\rangle$ has a subalgebra of size $|D|-2$. When $|D| \geq 5$ this subalgebra can be shown to be NP-complete. Hence, by Theorem 35, $\{B_D\}$ is NP-complete for finite sets D with $|D| \geq 5$. □

Proposition 46 ([15, 8]). *For any constraint language Γ over a finite set D, if $\mathrm{Pol}(\Gamma)$ contains a Mal'tsev operation, then Γ is tractable.*

The algorithm for solving a collection of constraints preserved by a Mal'tsev operation is based on a generalisation of Gaussian elimination [15]. A much more straightforward version of the algorithm is given in [8]. Note that no fixed level of consistency is sufficient to solve all problems involving constraints of this type.

Example 47. The linear constraints described in Example 4 all have the ternary polymorphism p given by $p(x, y, z) = x-y+z$, which is a Mal'tsev operation, so they are tractable by Proposition 46. A solution for any collection of linear constraints can be obtained by a Gaussian elimination algorithm on the corresponding linear equations. □

A unified approach to Mal'tsev operations and near-unanimity operations, which generalises Proposition 41 and Proposition 46 is given in [29].

6.5.3 Towards a complete classification of complexity

We have seen that the polymorphisms of a constraint language can identify many different tractable cases and suggest an appropriate efficient solution algorithm for those cases.

However, what can be said about a constraint language Γ where $\mathrm{Pol}(\Gamma)$ does *not* contain a semilattice operation, a conservative commutative binary operation, a near-unanimity operation or a Mal'tsev operation? We cannot in general immediately conclude that Γ is intractable. However, using Rosenberg's analysis of minimal clones [84, 87], we do have the following result (adapted slightly from [49]).

Definition 48. *Let f be a k-ary operation a set D.*

- *If there exists a (non-constant) unary operation g on D and an index $i \in \{1, 2, \ldots, k\}$ such that f satisfies the identity $f(x_1, x_2, \ldots, x_k) = g(x_i)$, then f is called an **essentially unary** operation. If g is the identity operation, then f is called a **projection**.*

- *If $k \geq 3$ and f satisfies the identity $f(x_1, \ldots, x_k) = x_i$ for some fixed i whenever $|\{x_1, x_2, \ldots, x_k\}| < k$, but f is* not *a projection, then f is called a **semiprojection**.*

Theorem 49. *For any reduced constraint language Γ on a finite set D, at least one of the following conditions must hold:*

1. $\mathrm{Pol}(\Gamma)$ *contains a constant operation;*
2. $\mathrm{Pol}(\Gamma)$ *contains a near-unanimity operation of arity 3;*
3. $\mathrm{Pol}(\Gamma)$ *contains a Malt'sev operation;*
4. $\mathrm{Pol}(\Gamma)$ *contains an idempotent binary operation (which is not a projection);*
5. $\mathrm{Pol}(\Gamma)$ *contains a semiprojection;*
6. $\mathrm{Pol}(\Gamma)$ *contains essentially unary surjective operations only.*

If $\mathrm{Pol}(\Gamma)$ contains a constant operation, then Γ is trivially tractable, since each (non-empty) relation in Γ contains a tuple $\langle d, d, \ldots, d\rangle$, where d is the value of the constant operation. By Propositions 37 and 46, the second and third cases also guarantee tractability. Hence the first three cases in Theorem 49 all guarantee tractability.

In the final case of Theorem 49 we observe that $\mathrm{Inv}(\mathrm{Pol}(\Gamma))$ includes the disequality relation, $\neq_D$, defined in Example 11, and when $|D| = 2$ it includes the not-all-equal relation, N_D, defined in Example 6. Hence in this case we have that $\mathrm{Inv}(\mathrm{Pol}(\Gamma))$ is NP-complete for all finite sets D, so by Theorem 21 and Corollary 17 we conclude that Γ is NP-complete in this case. Hence the final case of Theorem 49 guarantees NP-completeness.

A similar argument gives the following slightly more general result.

Proposition 50 ([49]). *Any set of essentially unary operations over a finite set is NP-complete.*

Cases 4 and 5 of Theorem 49 are inconclusive, in general, although for a Boolean domain there are only two binary idempotent operations which are not projections: the two semilattice operations $\wedge$ and $\vee$ (conjunction and disjunction). Hence, over a Boolean domain, case 4 guarantees tractability by Proposition 37. Moreover, over a Boolean domain there are no semiprojection operations, so case 5 cannot occur. These observations mean that Theorem 49 is sufficient to classify the complexity of any constraint language over a Boolean domain, and hence derive Schaefer's Dichotomy Theorem [85] (see Example 5).

Corollary 51 ([11]). *An algebra with a 2-element universe is NP-complete if all of its basic operations are essentially unary. Otherwise it is tractable.*

The single condition described in Proposition 50 is the only condition needed to establish the NP-completeness of all known NP-complete constraint languages, and has been used to establish a dichotomy theorem for several broad classes of languages [11]. There is a longstanding conjecture [18] that this condition is sufficient to characterise *all* forms of intractability in constraint languages. We state this conjecture for the special case of idempotent algebras, where the only essentially unary operations are projections.

Conjecture 52 ([18, 11]). ***Tractable algebras conjecture:*** *A finite idempotent algebra $\mathcal{A}$ is NP-complete if it has a nontrivial factor $\mathcal{B}$ all of whose operations are projections. Otherwise it is tractable.*

By Proposition 28 and Theorem 31, the problem of determining the complexity of an arbitrary constraint language can be reduced to an equivalent problem for a certain

idempotent algebra associated with the language. Therefore, this conjecture, if true, would completely solve the fundamental question of analysing the complexity of any constraint language over a finite set.

Conjecture 52 has been verified [11] for algebras with a 2-element universe, algebras with a 3-element universe, conservative algebras (i.e., those whose operations preserve all unary relations), and strictly simple surjective algebras (i.e. those with no non-trivial factors). If Conjecture 52 is true in general, then it yields an effective procedure to determine whether any finite constraint language is tractable or NP-complete, as the following result indicates.

Proposition 53 ([11]). *Let D be a fixed finite set. If Conjecture 52 is true, then for any finite constraint language Γ over D, there is a polynomial-time algorithm to determine whether Γ is NP-complete or tractable.*

In another direction, Proposition 50 was used in [70] to show that most non-trivial constraint languages over a finite domain are NP-complete. More precisely, let $R(n,k)$ denote a random n-ary relation on the set $\{1,\ldots,k\}$, for which the probability that $\langle a_1,\ldots,a_n\rangle \in R(n,k)$ is equal to 1/2 independently for each n-tuple $\langle a_1,\ldots,a_n\rangle$ where not all a_i's are equal; also, set $\langle a,a,\ldots,a\rangle \notin R(n,k)$ for all a (this is necessary to ensure that $\mathrm{CSP}(\{R(n,k)\})$ is non-trivial). It is shown in [70] that the probability that $\mathrm{Pol}\,\{R(n,k)\}$ contains only projections tends to 1 as either n or k tends to infinity.

6.5.4 Search is no harder than decision

In this chapter we have formulated the constraint satisfaction problem as a decision problem in which the question is to decide whether or not a solution exists. However, the corresponding *search problem*, in which the question is to find a solution, is often the real practical question. Using the algebraic theory in Section 6.4, we can now show that the tractable cases of these two forms of the problem coincide.

Theorem 54 ([11, 20]). *Let Γ be a constraint language over a finite set. The decision problem $\mathrm{CSP}(\Gamma)$ is tractable if and only if the corresponding search problem can be solved in polynomial time.*

Proof. Obviously, tractability of the search problem implies tractability of the corresponding decision problem.

For the converse, let Γ be a tractable set of relations over a finite domain D.

Consider any instance $\mathcal{P}$ in $\mathrm{CSP}(\Gamma)$. By the choice of Γ, we can decide in polynomial time whether $\mathcal{P}$ has a solution. If it does not then the search returns with no solution.

Otherwise, using Proposition 28 we can transform this instance to an instance $\mathcal{P}'$ over a reduced language Γ' which has a solution. Furthermore we can arrange that every solution to $\mathcal{P}'$ is a solution to $\mathcal{P}$.

Since $\mathcal{P}'$ has a solution we know that for each variable v of $\mathcal{P}'$ there must be some domain value $a \in D$ for which we can add the constraint $\langle\langle v\rangle, \{\langle a\rangle\}\rangle$ and still have a solvable instance. By considering each variable in turn, and each possible value for that variable, we can add such a constraint to each variable in turn, and hence obtain a solution to $\mathcal{P}'$. Checking for solvability for each possible value at each variable requires us to solve an instance of the decision problem $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{CON}})$ at most $|\mathcal{P}'|$ times. By Theorem 31, this can be completed in polynomial time in the size of $\mathcal{P}$. □

6.6 Constraint Languages Over an Infinite Set

Some computational problems can be formulated as constraint satisfaction problems only by using a constraint language over an *infinite* set (see Examples 10 and 12).

Many of the results of the algebraic theory described in Section 6.4 hold for both finite and infinite domains. However, Theorem 21 does not hold, in general, for arbitrary constraint languages over an infinite set. It is not hard to check that the inclusion $\langle\Gamma\rangle \subseteq \mathrm{Inv}(\mathrm{Pol}(\Gamma))$ still holds. However, for constraint languages over an infinite set this inclusion can be strict, as the next example[7] shows.

Example 55. Consider $\Gamma = \{R_1, R_2, R_3\}$ on $\mathbb{N}$, where $R_1 = \{\langle a, b, c, d\rangle \mid a = b \text{ or } c = d\}$, $R_2 = \{\langle 1\rangle\}$, and $R_3 = \{\langle a, a+1\rangle \mid a \in \mathbb{N}\}$. It is not difficult to show that every polymorphism of Γ is a projection, and hence $\mathrm{Inv}(\mathrm{Pol}(\Gamma))$ is the set of *all* relations on $\mathbb{N}$. However, one can check that, for example, the unary relation consisting of all even numbers does not belong to $\langle\Gamma\rangle$. □

However, if we impose some additional conditions, then the required equality does hold, as the next result indicates. A *relational structure* consists of a universe D, together with a collection of relations over D. A relational structure with a countably infinite universe is called *ω-categorical* if it is determined (up to isomorphism) by its first-order theory [46].

Theorem 56 ([4]). *Let $\Gamma = \{R_1, \ldots, R_k\}$ be a finite constraint language over a countably infinite set D.*

If the relational structure $\langle D, R_1, \ldots, R_k\rangle$ is ω-categorical, then $\langle\Gamma\rangle = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$.

Examples of ω-categorical structures, as well as remarks on the complexity of the corresponding constraint satisfaction problems, can be found in [3], including a complete analysis of the countably infinite ω-categorical structures with a single binary relation.

6.6.1 Allen's Interval Algebra

One form of infinite-valued CSP which has been widely studied is the case where the values taken by the variables are *intervals* on some totally ordered set. This setting is used to model the temporal behaviour of systems, where the intervals represent time intervals during which events occur. The most popular such formalism is **Allen's Interval Algebra**, introduced in [1], which concerns binary qualitative relations between intervals. This algebra contains 13 basic relations (see Table 6.1), corresponding to the 13 distinct ways in which two given intervals can be related. The complete set of relations in Allen's Interval Algebra consists of the $2^{13} = 8192$ possible *unions* of the basic relations. This set of relations will be denoted Γ_{AIA}

The constraint language Γ_{AIA} is NP-complete, and the problem of classifying the complexity of subsets of this language has attracted much attention (see, for example, [86]).

Allen's Interval Algebra has three operations on relations: composition, intersection, and inversion. Note that these three operations can each be represented by using conjunction and existential quantification, so, for any subset Δ of Γ_{AIA}, the subalgebra Δ' generated by Δ has the property that $\Delta' \subseteq \langle\Delta\rangle$. It follows from Theorem 16 that $\mathrm{CSP}(\Delta)$

[7] This example is from [3], where it is credited to F. Börner.

Basic relation		Example	Endpoints
I precedes J	p	III	$I^+ < J^-$
J preceded by I	p^{-1}	JJJ	
I meets J	m	IIII	$I^+ = J^-$
J met by I	m^{-1}	JJJJ	
I overlaps J	o	IIII	$I^- < J^- < I^+$,
J overl. by I	o^{-1}	JJJJ	$I^+ < J^+$
I during J	d	III	$I^- > J^-$,
J includes I	d^{-1}	JJJJJJJ	$I^+ < J^+$
I starts J	s	III	$I^- = J^-$,
J started by I	s^{-1}	JJJJJJJ	$I^+ < J^+$
I finishes J	f	III	$I^+ = J^+$,
J finished by I	f^{-1}	JJJJJJJ	$I^- > J^-$
I equals J	$\equiv$	IIII	$I^- = J^-$,
		JJJJ	$I^+ = J^+$

Table 6.1: The 13 basic relations in Allen's Interval Algebra.

and $\mathrm{CSP}(\Delta')$ are polynomial-time equivalent. Hence it is sufficient to classify all subsets of Γ_{AIA} which are *subalgebras* of Allen's Interval Algebra.

Theorem 57 ([63]). *For any constraint language $\Gamma \subseteq \Gamma_{\mathrm{AIA}}$, if Γ is contained in one of the eighteen subalgebras listed in Table 6.2, then it is tractable; otherwise it is NP-complete.*

The domain for Allen's Interval Algebra can be taken to be the countably infinite set of intervals with rational endpoints. It was noted in [4] that the relational structure associated with Allen's Interval Algebra (without its operations) is ω-categorical. Therefore, by Theorem 56, the complexity classification problem for subsets of Γ_{AIA} can be tackled using polymorphisms. Such an approach might provide a route to simplifying the involved classification proof given in [63].

6.7 Multi-Sorted Constraint Languages

In practical constraint programming it is often the case that different variables have different domains. So far in this chapter we have considered a simplified situation in which all of the variables are assumed to have the same domain. This apparently minor simplification can have serious consequences for the analysis of the complexity of different forms of constraint; it can in fact mask the difference between tractability and NP-completeness for some languages, as we will demonstrate in this section.

The algebraic approach described in Section 6.4 has been extended to deal with the case when different variables have different domains [10], and we will now present the main results of the extended theory.

Definition 58. *For any collection of sets $\mathfrak{D} = \{D_i \mid i \in I\}$, and any list of indices $\langle i_1, i_2, \ldots, i_n \rangle \in I^n$, a subset of $D_{i_1} \times D_{i_2} \times \cdots \times D_{i_n}$, together with the list $\langle i_1, i_2, \ldots, i_n \rangle$, will be called a **multi-sorted relation** over $\mathfrak{D}$ with arity n and **signature** $\langle i_1, i_2, \ldots, i_n \rangle$.*

$$\mathcal{S}_{\mathsf{p}} = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{f}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{p})^{\pm 1} \subseteq r\}$$
$$\mathcal{S}_{\mathsf{d}} = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{f}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{d}^{-1})^{\pm 1} \subseteq r\}$$
$$\mathcal{S}_{\mathsf{o}} = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{f}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{o})^{\pm 1} \subseteq r\}$$
$$\mathcal{A}_1 = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{f}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{s}^{-1})^{\pm 1} \subseteq r\}$$
$$\mathcal{A}_2 = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{f}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{s})^{\pm 1} \subseteq r\}$$
$$\mathcal{A}_3 = \{r \mid r \cap (\mathsf{pmodf})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{s})^{\pm 1} \subseteq r\}$$
$$\mathcal{A}_4 = \{r \mid r \cap (\mathsf{pmodf}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{s})^{\pm 1} \subseteq r\}$$

$$\mathcal{E}_{\mathsf{p}} = \{r \mid r \cap (\mathsf{pmods})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{p})^{\pm 1} \subseteq r\}$$
$$\mathcal{E}_{\mathsf{d}} = \{r \mid r \cap (\mathsf{pmods})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{d})^{\pm 1} \subseteq r\}$$
$$\mathcal{E}_{\mathsf{o}} = \{r \mid r \cap (\mathsf{pmods})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{o})^{\pm 1} \subseteq r\}$$
$$\mathcal{B}_1 = \{r \mid r \cap (\mathsf{pmods})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{f}^{-1})^{\pm 1} \subseteq r\}$$
$$\mathcal{B}_2 = \{r \mid r \cap (\mathsf{pmods})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{f})^{\pm 1} \subseteq r\}$$
$$\mathcal{B}_3 = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{s}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{f}^{-1})^{\pm 1} \subseteq r\}$$
$$\mathcal{B}_4 = \{r \mid r \cap (\mathsf{pmod}^{-1}\mathsf{s})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{f}^{-1})^{\pm 1} \subseteq r\}$$

$$\mathcal{E}^* = \left\{ r \;\middle|\; \begin{array}{l} 1)\ r \cap (\mathsf{pmod})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{s})^{\pm 1} \subseteq r, \text{ and} \\ 2)\ r \cap (\mathsf{ff}^{-1}) \neq \emptyset \Rightarrow (\equiv) \subseteq r \end{array} \right\}$$

$$\mathcal{S}^* = \left\{ r \;\middle|\; \begin{array}{l} 1)\ r \cap (\mathsf{pmod}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{f}^{-1})^{\pm 1} \subseteq r, \text{ and} \\ 2)\ r \cap (\mathsf{ss}^{-1}) \neq \emptyset \Rightarrow (\equiv) \subseteq r \end{array} \right\}$$

$$\mathcal{H} = \left\{ r \;\middle|\; \begin{array}{l} 1)\ r \cap (\mathsf{os})^{\pm 1} \neq \emptyset \;\&\; r \cap (\mathsf{o}^{-1}\mathsf{f})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{d})^{\pm 1} \subseteq r, \text{ and} \\ 2)\ r \cap (\mathsf{ds})^{\pm 1} \neq \emptyset \;\&\; r \cap (\mathsf{d}^{-1}\mathsf{f}^{-1})^{\pm 1} \neq \emptyset \Rightarrow (\mathsf{o})^{\pm 1} \subseteq r, \text{ and} \\ 3)\ r \cap (\mathsf{pm})^{\pm 1} \neq \emptyset \;\&\; r \not\subseteq (\mathsf{pm})^{\pm 1} \Rightarrow (\mathsf{o})^{\pm 1} \subseteq r \end{array} \right\}$$

$$\mathcal{A}_{\equiv} = \{r \mid r \neq \emptyset \Rightarrow (\equiv) \subseteq r\}$$

For the sake of brevity, relations are written as collections of basic relations. So, for instance, we write (pmod) instead of $\mathsf{p} \cup \mathsf{m} \cup \mathsf{o} \cup \mathsf{d}$. We also use the symbol $\pm$, which should be interpreted as follows: a condition involving $\pm$ means the conjunction of two conditions, one corresponding to $+$ and one corresponding to $-$. For example, the condition $(\mathsf{o})^{\pm 1} \subseteq r \Leftrightarrow (\mathsf{d})^{\pm 1} \subseteq r$ means that both $(\mathsf{o}) \subseteq r \Leftrightarrow (\mathsf{d}) \subseteq r$ and $(\mathsf{o}^{-1}) \subseteq r \Leftrightarrow (\mathsf{d}^{-1}) \subseteq r$.

Table 6.2: The 18 maximal tractable subalgebras of Allen's Interval Algebra

For any multi-sorted relation R, the signature of R will be denoted $\sigma(R)$.

In the special case where $\mathfrak{D}$ contains just a single set D we will call a multi-sorted relation over $\mathfrak{D}$ a *one-sorted* relation over D.

Example 59. Let R be a 5-ary relation with 17 tuples defined as follows:

$$\begin{aligned} R = \{ &\langle 3,1,2,c,b\rangle, \langle 3,3,2,c,b\rangle, \langle 1,0,2,c,b\rangle, \langle 1,2,2,c,b\rangle, \\ &\langle 1,1,0,c,b\rangle, \langle 1,3,0,c,b\rangle, \langle 3,0,0,c,b\rangle, \langle 3,2,0,c,b\rangle, \\ &\langle 3,1,2,c,a\rangle, \langle 3,3,2,c,a\rangle, \langle 1,0,2,c,a\rangle, \langle 1,2,2,c,a\rangle, \\ &\langle 3,1,2,a,b\rangle, \langle 3,3,2,a,b\rangle, \langle 1,1,0,a,b\rangle, \langle 1,3,0,a,b\rangle, \langle 3,3,2,a,a\rangle \} \end{aligned}$$

This relation can be considered in the usual way as a one-sorted relation over the set $D = \{0,1,2,3,a,b,c\}$. Alternatively, it can be seen as a multi-sorted relation with signature $\langle 1,1,1,2,2\rangle$ over the collection of sets $\mathfrak{D} = \langle D_1, D_2\rangle$, where $D_1 = \{0,1,2,3\}$ and $D_2 = \{a,b,c\}$. □

Given any set of multi-sorted relations, we can define a corresponding class of multi-sorted constraint satisfaction problems, in the following way.

Definition 60. *Let Γ be a set of multi-sorted relations over a collection of sets $\mathfrak{D} = \{D_i \mid i \in I\}$. The **multi-sorted constraint satisfaction problem** over Γ, denoted $\mathrm{MCSP}(\Gamma)$, is defined to be the decision problem with*

Instance: *A quadruple $\langle V, \mathfrak{D}, \delta, \mathcal{C}\rangle$, where*

- *V is a finite set of variables;*
- *δ is a mapping from V to I called the **domain function**;*
- *$\mathcal{C}$ is a set of constraints, where each constraint $C \in \mathcal{C}$ is a pair $\langle s, R\rangle$ such that*
  - *s, is a tuple of variables of length n_C called the constraint scope, and*
  - *R is an element of Γ with arity n_C and signature $\langle \delta(s[1]), \ldots, \delta(s[n_C])\rangle$ called the constraint relation.*

Question: *Does there exist a solution, that is a function φ from V to $\bigcup_{i\in I} D_i$ such that, for each variable $v \in V, \varphi(v) \in D_{\delta(v)}$, and for each constraint $\langle s, R\rangle \in \mathcal{C}$ with $s = \langle v_1, \ldots, v_n\rangle$, the tuple $\langle \varphi(v_1), \ldots, \varphi(v_n)\rangle$ belongs to the multi-sorted relation R.*

It might be tempting to assume that the complexity of a set of multi-sorted relations could be determined by considering each of the domains involved separately; in other words, by separating the relations into a number of one-sorted relations, and analysing the complexity of each of these. However, in general this simple approach does not work, as the next example demonstrates.

Example 61. Consider the sets $D_1 = \{0,1\}$ and $D_2 = \{a,b,c\}$, and the multi-sorted relations R_1, R_2, R_3 over $\mathfrak{D} = \{D_1, D_2\}$, each with signature $\langle 1,2\rangle$, where

$$\begin{aligned} R_1 = \{ &\langle 1,a\rangle, \\ &\langle 0,b\rangle, \\ &\langle 0,c\rangle\} \end{aligned} \qquad \begin{aligned} R_2 = \{ &\langle 0,a\rangle, \\ &\langle 1,b\rangle, \\ &\langle 0,c\rangle\} \end{aligned} \qquad \begin{aligned} R_3 = \{ &\langle 0,a\rangle, \\ &\langle 0,b\rangle, \\ &\langle 1,c\rangle\}. \end{aligned}$$

If we divide each of these multi-sorted relations into two separate one-sorted relations, then we obtain just the unary relations $\{0,1\}$ and $\{a,b,c\}$ over the sets D_1 and D_2 respectively. Each of these unary relations individually is clearly tractable.

However, by establishing a reduction from the NP-complete problem ONE-IN-THREE SATISFIABILITY (see Example 6), it can be shown that the set of multi-sorted relations $\Gamma = \{R_1, R_2, R_3\}$ is NP-complete. (Details of this reduction are given in [10].) □

It is often desirable to convert a multi-sorted constraint satisfaction problem into a one-sorted problem. The most straightforward way to do this for a given multi-sorted problem instance $\langle V, \mathfrak{D}, \delta, \mathcal{C}\rangle$, is to take $\overline{D} = \bigcup_{D_i \in \mathfrak{D}} D_i$, and replace each constraint relation with a one-sorted relation over $\overline{D}$ containing exactly the same tuples.

However, this straightforward conversion method does *not* necessarily preserve the tractability of a multi-sorted constraint language Γ, as the next example indicates.

Example 62. Let D_1 and D_2 be two distinct supersets of a set D_0, and let Γ be the constraint language containing the single binary disequality relation $\neq_{D_0}$, as defined in Example 11, but now considered as a multi-sorted relation over $\{D_1, D_2\}$ with signature $\langle 1, 2\rangle$.

Because of the signature, this constraint can only be imposed between two variables when one of them has domain D_1 and the other has domain D_2. Hence, in this case $\mathrm{MCSP}(\Gamma)$ corresponds to the problem of colouring a *bipartite graph* with $|D_0|$ colours, which is clearly tractable for any set D_0. Note that the tractability is entirely due to the signature of the relation rather than the tuples it contains.

If we convert Γ to a one-sorted constraint language by considering the relation $\neq_{D_0}$ as a one-sorted relation over the set $\overline{D} = D_1 \cup D_2$, then we obtain the usual disequality relation over D_0, which for $|D_0| > 2$ is NP-complete (see Example 11). □

To ensure that we do preserve tractability when converting a multi-sorted constraint language to a one-sorted constraint language, we make use of a more sophisticated conversion technique, based on the following definition.

Definition 63. *Let $\mathfrak{D} = \{D_1, \ldots, D_p\}$ be a finite collection of sets, and define $D^* = D_1 \times D_2 \times \cdots \times D_p$.*

For any n-ary relation R over $\mathfrak{D}$ with signature $\sigma(R) = \langle i_1, \ldots, i_n\rangle$, we define the one-sorted n-ary relation $\chi(R)$ over D^* as follows:

$$\chi(R) = \{\langle t_1, t_2, \ldots, t_n\rangle \in (D^*)^n \mid \langle t_1[i_1], t_2[i_2], \ldots, t_m[i_m]\rangle \in R\}.$$

Note that for any one-sorted relation R, we have $\chi(R) = R$.

Example 64. Let R be the binary disequality relation $\neq_{D_0}$ over $\{D_1, D_2\}$ with signature $\langle 1, 2\rangle$, as in Example 62. In this case $\chi(R)$ is the relation consisting of all pairs $\langle\langle a, a'\rangle, \langle b, b'\rangle\rangle \in (D_1 \times D_2) \times (D_1 \times D_2)$ such that $a, b' \in D_0$ and $a \neq b'$. □

Proposition 65 ([10]). *Let Γ be a multi-sorted constraint language over a finite collection of finite sets. The language Γ is tractable if and only if the corresponding one-sorted constraint language $\{\chi(R) \mid R \in \Gamma\}$ is tractable.*

To extend the algebraic results of Section 6.4 to the multi-sorted case, we need to define a suitable extension of the notion of a polymorphism. As we have shown in Example 61, we cannot simply separate out different domains and consider polymorphisms on each one

separately; we must ensure that all of the domains are treated in a co-ordinated way. In the following definition, this is achieved by defining different *interpretations* for the same operation symbol applied to different sets.

Definition 66. *Let $\mathfrak{D} = \{D_i \mid i \in I\}$ be a collection of sets. A k-ary **multi-sorted operation** f on $\mathfrak{D}$ is defined by a collection of **interpretations** $\{f^{D_i} \mid i \in I\}$, where each f^{D_i} is a k-ary operation on the corresponding set D_i.*

For any multi-sorted relation R with signature $\langle i_1, \ldots, i_n \rangle$, and any collection of tuples $t_1, \ldots, t_k \in R$, define $f(t_1, \ldots, t_k)$ to be

$$\left\langle f^{D_{i_1}}(t_1[1], \ldots, t_k[1]), \ldots, f^{D_{i_n}}(t_1[n], \ldots, t_k[n]) \right\rangle.$$

Definition 67. *A k-ary multi-sorted operation f on $\mathfrak{D}$ is said to be a **multi-sorted polymorphism** of a multi-sorted relation R over $\mathfrak{D}$ if $f(t_1, \ldots, t_k) \in R$ for all choices of $t_1, \ldots, t_k \in R$.*

For any given multi-sorted constraint language Γ, the set of all multi-sorted polymorphisms of *every* relation in Γ is denoted $\mathrm{MPol}(\Gamma)$.

The next theorem is the main result of this section. It establishes the remarkable fact that many of the polymorphisms that ensure tractability in the one-sorted case can be combined in almost arbitrary ways to obtain new tractable multi-sorted constraint languages.

Note that a multi-sorted operation, f, is said to be *idempotent* if all of its interpretations f^D satisfy the identity $f^D(x, x, \ldots, x) = x$.

Theorem 68 ([10]). *Let Γ be a multi-sorted constraint language over a finite collection of finite sets $\mathfrak{D} = \{D_1, \ldots, D_n\}$.*

If, for each $D_i \in \mathfrak{D}$, $\mathrm{MPol}(\Gamma)$ contains a multi-sorted operation f_i such that

- *$f_i^{D_i}$ is a constant operation; or*
- *$f_i^{D_i}$ is a semilattice operation; or*
- *$f_i^{D_i}$ is a near-unanimity operation; or*
- *f_i is idempotent and $f_i^{D_i}$ is an affine operation,*

then $\mathrm{MCSP}(\Gamma)$ is tractable.

Example 69. Recall the relation R defined in Example 59.

If we consider R as a one-sorted relation over the domain $\{0, 1, 2, 3, a, b, c\}$, then it does not fall into any of the many known (one-sorted) tractable classes described in Section 6.5.2 above[8].

However if we consider R as a multi-sorted relation with signature $\langle 1, 1, 1, 2, 2 \rangle$ over the sets $D_1 = \{0, 1, 2, 3\}$ and $D_2 = \{a, b, c\}$, then we can use Theorem 68 to show that $\{R\}$ is tractable. To see this, it is sufficient to check that R has two multi-sorted polymorphisms $f(x, y, z)$ and $g(x, y)$, where

- f^{D_1} is the affine operation of the group $\mathbb{Z}_4$, and f^{D_2} is the (ternary) maximum operation on D_2, with respect to the order $a < b < c$ (which is idempotent).

[8] This was established by using the program *Polyanna* described in [39], which is available from http://www.comlab.ox.ac.uk/oucl/research/areas/constraints/software/.

- $g^{D_1}(x, y) = y$, and g^{D_2} is the (binary) maximum operation on D_2, with respect to the order $a < b < c$ (which is a semilattice operation).

□

Further developments in the algebraic approach to multi-sorted constraints, and applications to the standard one-sorted CSP where the constraints limit the domain of each variable, are given in [10].

6.8 Alternative Approaches

6.8.1 Homomorphism problems

An important reformulation of the CSP is the HOMOMORPHISM problem: the question of deciding whether there exists a homomorphism between two *relational structures* (see [3, 37, 41, 58]). Recall that a relational structure is simply a set, together with a list of relations over that set.

Definition 70. *Let $\mathcal{A}_1 = \langle D_1, R_1^1, R_2^1, \ldots, R_q^1 \rangle$ and $\mathcal{A}_2 = \langle D_2, R_1^2, R_2^2, \ldots, R_q^2 \rangle$ be relational structures where both R_i^1 and R_i^2 are n_i-ary, for all $i = 1, 2, \ldots, q$.*

*A mapping $\Phi : D_1 \to D_2$ is called a **homomorphism** from $\mathcal{A}_1$ to $\mathcal{A}_2$ if it has the property that $\langle \Phi(a_1), \ldots, \Phi(a_{n_i}) \rangle \in R_i^2$ whenever $\langle a_1, \ldots, a_{n_i} \rangle \in R_i^1$, for all $i = 1, 2, \ldots, q$.*

The HOMOMORPHISM PROBLEM *for $\langle \mathcal{A}_1, \mathcal{A}_2 \rangle$ is to decide whether there exists a homomorphism from $\mathcal{A}_1$ to $\mathcal{A}_2$*

To see that the HOMOMORPHISM PROBLEM is the same as the CSP, think of the elements in A_1 as variables, the elements in A_2 as values, tuples in the relations of $\mathcal{A}_1$ as constraint scopes, and the relations of $\mathcal{A}_2$ as constraint relations. With this correspondence, the solutions to this CSP instance are precisely the homomorphisms from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Example 71. A relational structure with a single binary relation $\langle V, E \rangle$ is usually known as a (directed) **graph**.

An instance of the GRAPH H-COLORING problem consists of a finite graph G. The question is whether there is a homomorphism from G to H. When H is the complete graph on k vertices, the GRAPH H-COLORING problem corresponds to the standard GRAPH COLORABILITY problem with k colours (see Example 11). For an arbitrary graph $H = \langle V, E \rangle$, the GRAPH H-COLORING problem precisely corresponds to the problem $\mathrm{CSP}(\{E\})$.

For *undirected* graphs H, where the edge relation E is symmetric, the complexity of GRAPH H-COLORING has been completely characterised [44]: it is tractable if H is bipartite or contains a loop; otherwise it is NP-complete. (Note that this characterisation also follows from Conjecture 52, see [7].) However, if we allow H and G to be directed graphs, then the complexity of GRAPH H-COLORING has not yet been fully characterised. Moreover, it was shown in [37] that every problem $\mathrm{CSP}(\Gamma)$ with finite Γ is polynomial-time equivalent to GRAPH H-COLORING for some suitable directed graph H. □

6.8.2 Constraint languages and logic

In the field of *descriptive complexity* [47] the computational complexity of a problem is investigated by studying the forms of *logic* which can be used to express that problem. The

use of descriptive complexity techniques to analyse the complexity of constraint languages was initiated by the pioneering work of Feder and Vardi [37].

As shown in Section 6.8.1, for any finite constraint language $\Gamma = \{R_1, \ldots, R_q\}$ over a set D, the problem $\mathrm{CSP}(\Gamma)$ can be represented as the problem of deciding whether a given relational structure has a homomorphism to the relational structure $\langle D, R_1, \ldots, R_q \rangle$, Hence the class of instances of $\mathrm{CSP}(\Gamma)$ which do have a solution can be viewed as a class of relational structures (sometimes called the "yes-instances"). If this class of relational structures can be characterised in some restricted logic, then this can sometimes be used to show that $\mathrm{CSP}(\Gamma)$ is tractable, as the following example illustrates.

Example 72. Recall from Example 11 that $\mathrm{CSP}(\{\neq_D\})$ is equivalent to the problem of colouring a graph with $|D|$ colours. The class of instances which have a solution is the class of $|D|$-colourable graphs, which is a class of relational structures with a single symmetric binary relation E (specifying which vertices are connected by edges).

Now assume that $D = \{0, 1\}$. It is well-known that a graph (V, E) is 2-colourable if and only if it does not have any odd-length cycles. The property of having an odd-length cycle can be expressed in the logic programming language **Datalog** [37] using the following set of rules:

$$\begin{aligned} P(x, y) &:- E(x, y) \\ P(x, y) &:- P(x, z) \wedge E(z, u) \wedge E(u, y) \\ Q &:- P(x, x) \end{aligned}$$

These rules give a recursive specification of two predicates, P and Q. Predicate $P(x, y)$ holds exactly when there exists an odd-length path in (V, E) from x to y. Predicate Q, which acts as goal predicate, holds if there exists any odd-length cycle.

Hence, the class of structures for which $\mathrm{CSP}(\{\neq_{\{0,1\}}\})$ has a solution can be characterised as the set of structures (V, E) for which the goal predicate in this Datalog program does not hold. It was shown in [37] that any CSP problem whose yes-instances can be characterised by a Datalog program in this way is tractable.

It has recently been shown that any CSP problem whose yes-instances can be characterised in first-order logic can be characterised by a Datalog program in this way [2]. □

The techniques of descriptive complexity can also be used to obtain a more refined description of the complexity of a constraint language. For example, Dalmau has shown [30] that if a finite constraint language Γ has a logical property which he calls "bounded path duality", then the problem $\mathrm{CSP}(\Gamma)$ is in the complexity class NL, and so can be solved very efficiently using parallel algorithms.

6.8.3 Disjunctive combinations of constraint languages

Another approach to the analysis of constraint languages has been to consider how they can be built up from combinations of simpler languages whose properties are more easily analyzed [25, 6]. This approach has successfully unified several important classes of tractable languages including five of the six tractable Boolean languages (Example 5), the max-closed constraints (Example 7), the 0/1/all constraints (Example 8), the connected row-convex constraints (Example 9) and the linear Horn constraints (Example 13). One

advantage of this constructive approach is that it works equally well for both finite and infinite domains.

The key step in this approach is to define how relations can be combined disjunctively.

Definition 73. *Let R_1 be an n-ary relation and R_2 an m-ary relation over a common set D. The **disjunction** of R_1 and R_2, denoted $R_1 \vee R_2$, is the relation of arity $n+m$ over D defined as follows:*

$$R_1 \vee R_2 = \langle \langle x_1, \ldots, x_{n+m} \rangle \mid (\langle x_1, \ldots, x_n \rangle \in R_1) \vee (\langle x_{n+1}, \ldots, x_{n+m} \rangle \in R_2) \} \rangle$$

This definition of disjunction can be extended to constraint languages as follows.

Definition 74. *For any two constraint languages Γ and Δ, over the same domain D, define the constraint language $\Gamma^{\veebar}\Delta$ as follows:*

$$\Gamma^{\veebar}\Delta = \Gamma \cup \Delta \cup \{R_1 \vee R_2 \mid R_1 \in \Gamma, R_2 \in \Delta\}$$

The constraint language $\Gamma^{\veebar}\Delta$ (pronounced Γ "or-times" Δ) contains all of the relations in Γ and Δ, together with the disjunction of each possible pair of relations from Γ and Δ.

The next example shows that when tractable constraint languages are combined using the disjunction operation defined in Definition 74 the resulting constraint language may or may not be tractable.

Example 75. Let Λ be the set of all relations over the domain $\{\text{TRUE}, \text{FALSE}\}$ which can be specified by a formula of propositional logic consisting of a single *literal* (where a literal is either a variable or a negated variable).

The constraint language Λ is clearly tractable, as it is straightforward to verify in linear time whether a collection of simultaneous single literals has a solution.

Now consider the constraint language $\Lambda^{\vee 2} = \Lambda^{\veebar}\Lambda$. This set contains all Boolean constraints specified by a disjunction of (at most) 2 literals. The problem $\mathrm{CSP}(\Lambda^{\vee 2})$ corresponds to the 2-SATISFIABILITY problem, which is well-known to be tractable [38] (see Example 42).

Finally, consider the constraint language $\Lambda^{\vee 3} = (\Lambda^{\vee 2})^{\veebar}\Lambda$. This set of relations contains all Boolean relations specified by a disjunction of (at most) 3 literals. The problem $\mathrm{CSP}(\Lambda^{\vee 3})$ corresponds to the 3-SATISFIABILITY problem, which is well-known to be NP-complete [38, 77]. □

Definition 76. *For any constraint language, Δ, define the set Δ^* as follows:*

$$\Delta^* = \bigcup_{i=1}^{\infty} \Delta^{\vee i}, \quad \textit{where}$$

$$\Delta^{\vee 1} = \Delta$$
$$\Delta^{\vee (i+1)} = (\Delta^{\vee i})^{\veebar}\Delta \ \textit{ for } \ i = 1, 2, \ldots$$

In the remainder of this section we identify a number of simple conditions on constraint languages Γ and Δ which are necessary and sufficient to ensure that various disjunctive combinations of Γ and Δ are tractable.

Definition 77. *For any constraint languages Γ and Δ over a common domain D, define $\mathrm{CSP}_{\Delta \leq k}(\Gamma \cup \Delta)$ to be the subproblem of $\mathrm{CSP}(\Gamma \cup \Delta)$ consisting of all instances containing at most k constraints whose relations are members of Δ.*

Using this definition, we now define what it means for one set of constraints to be 'k-independent' with respect to another.

Definition 78. *For any constraint languages Γ and Δ over a set D, we say that Δ is **k-independent with respect to** Γ if the following condition holds: any instance $\langle V, D, \mathcal{C}\rangle$ in $\mathrm{CSP}(\Gamma \cup \Delta)$ has a solution provided that any instance $\langle V, D, \mathcal{C}'\rangle$ in $\mathrm{CSP}_{\Delta \leq k}(\Gamma \cup \Delta)$ with $\mathcal{C}' \subseteq \mathcal{C}$ has a solution.*

The intuitive meaning of this definition is that the satisfiability of any set of constraints with relations chosen chosen from the set Δ can be determined by considering those constraints k at a time, even in the presence of arbitrary additional constraints from Γ.

Theorem 79 ([25, 6]). *Let Γ and Δ be constraint languages over a set D, such that $\mathrm{CSP}_{\Delta \leq 1}(\Gamma \cup \Delta)$ is tractable.*

The constraint language $\Gamma^{\times}_{\vee}\Delta^*$ is tractable if Δ is 1-independent with respect to Γ. Otherwise it is NP-complete.

A polynomial-time algorithm for solving instances of $\mathrm{CSP}(\Gamma^{\times}_{\vee}\Delta^*)$, for any constraint languages Γ and Δ satisfying the conditions of Theorem 79 is given in [25].

Example 80. Let D be the set of real numbers (or the rationals). Let Γ be the constraint language over D consisting of all constraints specified by a single (weak) linear inequality (e.g., $3x_1 + 2x_2 - x_3 \leq 6$). Let Δ be the constraint language over D consisting of all constraints specified by a single linear disequality (e.g., $x_1 + 4x_2 + x_3 \neq 0$).

To show that $\mathrm{CSP}_{\Delta \leq 1}(\Gamma \cup \Delta)$ is tractable, we note that the consistency of a set of inequalities, $\mathcal{C}$, can be decided in polynomial time, using Khachian's linear programming algorithm [56]. Furthermore, for any single disequality constraint, C, we can detect in polynomial time whether $\mathcal{C} \cup \{C\}$ is consistent by simply running Khachian's algorithm to determine whether $\mathcal{C}$ implies the negation of C.

To show that Δ is 1-independent with respect to Γ, we consider the geometrical interpretation of the constraints as half spaces and excluded hyperplanes in D^n (see [61]).

Hence, we can apply Theorem 79 and conclude that $\Gamma^{\times}_{\vee}\Delta^*$ is tractable. This set consists of the linear Horn realtions described in Example 13.

Note that the problem $\mathrm{CSP}(\Gamma \cup \Delta^*)$ is much simpler than $\mathrm{CSP}(\Gamma^{\times}_{\vee}\Delta^*)$ - it corresponds to deciding whether a convex polyhedron, possibly *minus* the union of a finite number of hyperplanes, is the empty set. This simpler problem was shown to be tractable in [68], using a more restrictive notion of independence which has been widely used in the development of consistency checking algorithms and canonical forms [67, 68]. However, the much larger set of linear Horn constraints is *not* independent in the sense defined in [68] (see [61]). □

Theorem 81 ([6]). *Let Γ and Δ be constraint languages over a set D, such that $\mathrm{CSP}(\Gamma \cup \Delta)$ is tractable.*

The constraint language $\Gamma \cup \Delta^{\vee 2}$ is tractable if Δ is 2-independent with respect to Γ. Otherwise it is NP-complete.

Note that Δ is 2-independent with respect to $\emptyset$ if and only if for every $\langle V, D, \mathcal{C}\rangle \in \mathrm{CSP}(\Delta)$ which has no solution, there exists a pair of (not necessarily distinct) constraints $C_i, C_j \in \mathcal{C}$ such that $\langle V, D, \{C_i, C_j\}\rangle$ has no solution.

A polynomial-time algorithm for solving instances of $\mathrm{CSP}(\Gamma \cup \Delta^{\vee 2})$, for any constraint languages Γ and Δ satisfying the conditions of Theorem 81 is given in [6].

Example 82. Consider the class of connected row-convex constraints over a set D described in Example 9. In this example we will show that the tractability of connected row-convex constraints is a simple consequence of Theorem 81. Furthermore, by using Theorem 81 we are able to generalise this result to obtain tractable constraints over infinite sets of values.

Note that the 0-1 matrices defining binary connected row-convex constraints have a very restricted structure. If we eliminate all rows and columns consisting entirely of zeros, and then consider any remaining zero in the matrix, all of the ones in the same row as the chosen zero must lie one side of it (because of the connectedness condition on the row). Similarly, all of the ones in the same column must lie on one side of the chosen zero. Hence there is a complete path of zeros from the chosen zero to the edge of the matrix along both the row and column in one direction. But this means there must be a complete rectangular sub-matrix of zeros extending from the chosen zero to one corner of the matrix (because of the connectedness condition).

This implies that the whole matrix can be obtained as the intersection (conjunction) of 0-1 matrices that contain all ones except for a submatrix of zeros in one corner (simply take one such matrix, obtained as above, for each zero in the matrix to be constructed).

There are four different forms of such matrices, depending on which corner submatrix is zero, and they correspond to constraints expressed by disjunctive expressions of the four following forms:

$$\begin{array}{lcl} (x_i \geq d_i) & \vee & (x_j \geq d_j) \\ (x_i \geq d_i) & \vee & (x_j \leq d_j) \\ (x_i \leq d_i) & \vee & (x_j \geq d_j) \\ (x_i \leq d_i) & \vee & (x_j \leq d_j) \end{array}$$

In these expressions x_i, x_j are variables and d_i, d_j are constants.

Finally, we note that a row or column consisting entirely of zeros corresponds to a constraint of the form $(x_i \leq d_1) \vee (x_i \geq d_2)$ for an appropriate choice of d_1 and d_2.

Hence, any connected row-convex constraint is equivalent to a conjunction of expressions of these forms.

Now define Δ to be the set of all unary constraints over D specified by a single inequality of the form $x_i \leq d_i$ or $x_i \geq d_i$, for some $d_i \in D$.

It is easily shown that Δ is 2-independent with respect to $\emptyset$ and $\mathrm{CSP}(\Delta)$ is tractable, since each instance consists of a conjunction of upper and lower bounds for individual variables. Hence, by Theorem 81, $\Delta^{\diamondsuit}\Delta$ is tractable. By the alternative characterisation described above, this establishes that connected row-convex constraints are tractable.

Unlike the arguments used previously to establish that connected row-convex constraints are tractable [35, 50], the argument above can still be applied when the set of values D is infinite. □

Many further examples of constraint languages over both finite and infinite domains which can be shown to be tractable by constructing them from simpler languages are given in [25].

Disjunctive combinations of constraint languages over *different* domains are discussed in [24, 13]. These papers make use of the algebraic methods discussed in Section 6.4 above.

6.9 Future Directions

We have shown in this chapter that considerable progress has been made in analysing the complexity of constraint problems with specified constraint languages. The algebraic approach described in Section 6.4 has led to a complete classification for many special cases of constraint languages, and has prompted the conjecture that *all* constraint languages can be classified as either tractable or NP-complete on the basis of their algebraic properties (Conjecture 52).

Even greater progress has been made in analysing the complexity of constraint problems with specified structure, where the constraint language is unrestricted. A number of powerful structural decomposition algorithms have been developed for constraint satisfaction problems, often based on ideas from relational database theory [40]. A complete classification of the complexity of constraint satisfaction problems where the structure of the constraints is fixed but the constraint relations are unrestricted is given in [42].

However, there is currently very little analytical work which combines these two approaches. The most promising result of this kind shows that a certain level of local consistency (see Chapter 3), which depends on the constraint *tightness* and the maximum constraint arity, is sufficient to ensure global consistency [88]. In general, enforcing the required level of local consistency will increase the constraint arity, and so increase the required level of consistency still further, which means that this result can only be used to establish the tractability of classes of problems involving particular languages applied on particular restricted structures [88]. Other "hybrid" results of this kind, involving both structural and language properties, are discussed in [78] and in Chapter 12 of [32].

In many practical problems it will be the case that some constraints fall into one tractable class and some fall into another. Can this fact be exploited to obtain an efficient solution strategy? Does this depend on the structural way in which the different forms of constraint overlap? There is currently no suitable theoretical framework to address this question. One promising approach would be to incorporate ideas of space complexity, as well as time complexity. The ability to construct solutions using only a limited amount of working space and stored information seems to be a unifying principle between many disparate techniques in constraint programming such as bucket elimination [32], hypertree decomposition [40], and several forms of tractable constraint language [52].

Another direction of future work is to extend the analysis presented here to other forms of constraint problem, such as *quantified* constraint problems, *soft* constraint problems, *overconstrained* problems, or problems where we wish to *count* the number of solutions [62]. There has been considerable progress in analysing variations of this kind for Boolean constraint problems [27]. For larger finite domains there have been some initial studies of the complexity of quantified constraint problems [5] and counting constraint problems [9] based on extensions to the algebraic theory described in this chapter: for example, it has been shown that for both of these problems the complexity of a constraint language is determined by its polymorphisms [5, 9].

A rather more substantial extension of the algebraic theory presented here is required to analyse the complexity of soft constraints, because in this form of problem the constraints are represented as functions from tuples of domain values to some measure of desirability (see Chapter 9, "Soft Constraints"). Many forms of combinatorial optimisation problems can be represented in this very general framework [27, 57]. An initial approach to analysing

the complexity of such problems using algebraic techniques is developed in [21, 22] and a tractable soft constraint language is presented in [23].

Bibliography

[1] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] A. Atserias. On digraph coloring problems and treewidth duality. In *Proceedings 20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 106–115, 2005.

[3] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. To appear in the *Journal of Logic and Computation*.

[4] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. In *Proceedings of Computer Science Logic and the 8th Kurt Gödel Colloquium*, volume 2803 of *Lecture Notes in Computer Science*, pages 44–57. Springer-Verlag, 2003.

[5] F. Boerner, A. Bulatov, P. Jeavons, and A. Krokhin. Quantified constraints: Algorithms and complexity. In *Proceedings of Computer Science Logic and the 8th Kurt Gödel Colloquium*, volume 2803 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2003.

[6] M. Broxvall, P. Jonsson, and J. Renz. Disjunctions, independence, refinements. *Artificial Intelligence*, 140(1-2):153–173, 2002.

[7] A. Bulatov. H-coloring dichotomy revisited. *Theoretical Computer Science*, 349(1):31–39, 2005.

[8] A. Bulatov and V. Dalmau. Mal'tsev constraints are tractable. *SIAM Journal on Computing*. (To appear).

[9] A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *Proceedings 44th Symposium on Foundations of Computer Science (FOCS 2003)*, pages 562–573. IEEE Computer Society, 2003.

[10] A. Bulatov and P. Jeavons. An algebraic approach to multi-sorted constraints. In *Proceedings 9th International Conference on Constraint Programming—CP'03 (Kinsale, September 2003)*, volume 2833 of *Lecture Notes in Computer Science*, pages 183–198. Springer-Verlag, 2003.

[11] A. Bulatov, A. Krokhin, and P. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

[12] A. Bulatov, Jeavons P., and M. Volkov. Finite semigroups imposing tractable constraints. In *Proceedings of the School on Algorithmic Aspects of the Theory of Semigroups and its Applications, Coimbra, Portugal, 2001*, pages 313–329. World Scientific, 2002.

[13] A. Bulatov and E. Skvortsov. Amalgams of constraint satisfaction problems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 197–202. Morgan Kaufmann, 2003.

[14] A.A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proceedings 43rd IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 649–658, Vancouver, Canada, 2002.

[15] A.A. Bulatov. Mal'tsev constraints are tractable. Technical Report PRG-RR-02-05, Computing Laboratory, University of Oxford, Oxford, UK, 2002.

[16] A.A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 321–330, Ottawa, Canada, 2003. IEEE Press.

[17] A.A. Bulatov and P.G. Jeavons. Tractable constraints closed under a binary operation. Technical Report PRG-TR-12-00, Computing Laboratory, University of Oxford, Oxford, UK, 2002.

[18] A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *Lecture Notes in Computer Science*, pages 272–282. Springer-Verlag, 2000.

[19] A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. The complexity of maximal constraint languages. In *Proceedings 33rd ACM Symposium on Theory of Computing (STOC'01)*, pages 667–674, 2001.

[20] D. Cohen. Tractable decision for a constraint language implies tractable search. *Constraints*, 9:219–229, 2004.

[21] D. Cohen, M. Cooper, and P. Jeavons. A complete characterization of complexity for Boolean constraint optimization problems. In *Proceedings 10th International Conference on Constraint Programming—CP'04*, volume 3258 of *Lecture Notes in Computer Science*, pages 212–226. Springer-Verlag, 2004.

[22] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. Soft constraints: Complexity and multimorphisms. In *Proceedings 9th International Conference on Constraint Programming—CP'03 (Kinsale, September 2003)*, volume 2833 of *Lecture Notes in Computer Science*, pages 244–258. Springer-Verlag, 2003.

[23] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. A maximal tractable class of soft constraints. *Journal of Artificial Intelligence Research (JAIR)*, 22:1–22, 2004.

[24] D.A. Cohen, P.G. Jeavons, and R.L. Gault. New tractable classes from old. *Constraints*, 8:263–282, 2003.

[25] D.A. Cohen, P.G. Jeavons, P. Jonsson, and M. Koubarakis. Building tractable disjunctive constraints. *Journal of the ACM*, 47:826–853, 2000.

[26] M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65:347–361, 1994.

[27] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA., 2001.

[28] V. Dalmau. A new tractable class of constraint satisfaction problems. In *Proceedings 6th International Symposium on Artificial Intelligence and Mathematics*, 2000.

[29] V. Dalmau. Generalized majority-minority operations are tractable. In *Proceedings 20th IEEE Symposium on Logic in Computer Science, (LICS 2005)*, pages 438–447. IEEE Computer Society, 2005.

[30] V. Dalmau. Linear datalog and bounded path duality of relational structures. *Logical Methods in Computer Science*, 1:1–32, 2005.

[31] V. Dalmau, R. Gavaldà, P. Tesson, and D. Thérien. Tractable clones of polynomials over semigroups. In *Proceedings 11th International Conference on Constraint Programming—CP'05 (Sitges, October 2005)*, volume 3709 of *Lecture Notes in*

Computer Science, pages 196–210. Springer-Verlag, 2005.

[32] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[33] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[34] K. Denecke and S.L. Wismath. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall/CRC Press, 2002.

[35] Y. Deville, O. Barette, and P. van Hentenryck. Constraint satisfaction over connected row convex constraints. In *Proceeedings of IJCAI'97*, pages 405–411, 1997.

[36] T. Drakengren and P Jonsson. A complete classification of tractability in Allen's algebra relative to subsets of basic relations. *Artificial Intelligence*, 106:205–219, 1998.

[37] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.

[38] M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA., 1979.

[39] R.L Gault and P. Jeavons. Implementing a test for tractability. *Constraints*, 9:139–160, 2004.

[40] G. Gottlob, L. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124:243–282, 2000.

[41] G. Gottlob, L. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[42] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. In *Proceedings 44th Annual IEEE Symposium on Foundations of Computer Science, (FOCS'03)*, pages 552–561, 2003.

[43] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, 1994.

[44] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Ser.B*, 48:92–110, 1990.

[45] D. Hobby and R.N. McKenzie. *The Structure of Finite Algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, R.I., 1988.

[46] W. Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

[47] N. Immerman. *Descriptive Complexity*. Texts in Computer Science. Springer-Verlag, 1998.

[48] P.G. Jeavons. Constructing constraints. In *Proceedings 4th International Conference on Constraint Programming—CP'98 (Pisa, October 1998)*, volume 1520 of *Lecture Notes in Computer Science*, pages 2–16. Springer-Verlag, 1998.

[49] P.G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[50] P.G. Jeavons, D.A. Cohen, and M.C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1–2):251–265, 1998.

[51] P.G. Jeavons, D.A. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *Proceedings 1st International Conference on Constraint Programming, CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 1995.

[52] P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.

[53] P.G. Jeavons, D.A. Cohen, and M. Gyssens. How to determine the expressive power of constraints. *Constraints*, 4:113–131, 1999.

[54] P.G. Jeavons and M.C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

[55] P. Jonsson and C. Bäckström. A unifying approach to temporal constraint reasoning. *Artificial Intelligence*, 102:143–155, 1998.

[56] L.G. Khachian. A polynomial time algorithm for linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979.

[57] S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.

[58] Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.

[59] M. Koubarakis. Dense time and temporal constraints with $\neq$. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92)*, pages 24–35, San Mateo, CA, 1992. Morgan Kaufmann.

[60] M. Koubarakis. From local to global consistency in temporal constraint networks. In *Proceedings 1st International Conference on Constraint Programming—CP'95 (Cassis, France, September 1995)*, volume 976 of *Lecture Notes in Computer Science*, pages 53–69. Springer-Verlag, 1995.

[61] M. Koubarakis. Tractable disjunctions of linear constraints. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 297–307. Springer-Verlag, 1996.

[62] A. Krokhin, A. Bulatov, and P. Jeavons. Functions of multiple-valued logic and the complexity of constraint satisfaction: A short survey. In *Proceedings 33rd IEEE International Symposium on Multiple-Valued Logic (ISMVL 2003)*, pages 343–351. IEEE Computer Society, 2003.

[63] A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *Journal of the ACM*, 50:591–640, 2003.

[64] A. Krokhin, P. Jeavons, and P. Jonsson. Constraint satisfaction problems on intervals and lengths. *SIAM Journal on Discrete Mathematics*, 17:453–477, 2004.

[65] P.B. Ladkin and R.D. Maddux. On binary constraint problems. *Journal of the ACM*, 41:435–469, 1994.

[66] R.E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.

[67] J-L. Lassez and K. McAloon. A constraint sequent calculus. In *Constraint Logic Programming, Selected Research*, pages 33–43. MIT Press, 1991.

[68] J-L. Lassez and K. McAloon. A canonical form for generalized linear constraints. *Journal of Symbolic Computation*, 13:1–24, 1992.

[69] D. Lesaint, N. Azarmi, R. Laithwaite, and P. Walker. Engineering dynamic scheduler for Work Manager. *BT Technology Journal*, 16:16–29, 1998.

[70] T. Łuczak and J. Nešetřil. A probabilistic approach to the dychotomy problem. Technical Report 2003-640, KAM-DIMATIA Series, Charles University, Prague, 2003.

[71] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[72] A.K. Mackworth. Constraint satisfaction. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 285–293. Wiley Interscience, 1992.

[73] A.K. Mackworth and E.C. Freuder. The complexity of constraint satisfaction revisited. *Artificial Intelligence*, 59:57–62, 1993.

[74] R.N. McKenzie, G.F. McNulty, and W.F. Taylor. *Algebras, Lattices and Varieties*, volume I. Wadsworth and Brooks, California, 1987.

[75] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[76] B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42:43–66, 1995.

[77] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[78] J.K. Pearson and P.G. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London, July 1997.

[79] N. Pippenger. *Theories of Computability*. Cambridge University Press, Cambridge, 1997.

[80] R. Pöschel and L.A. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.

[81] E.L. Post. *The two-valued iterative systems of mathematical logic*, volume 5 of *Annals Mathematical Studies*. Princeton University Press, 1941.

[82] L. Purvis and P. Jeavons. Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proceedings of 1st International Conference on The Practical Application of Constraint Technologies and Logic Programming - PACLP'99*, pages 63–79. Practical Applications Company, 1999.

[83] J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the Region Connection Calculus. *Artificial Intelligence*, 108:69–123, 1999.

[84] I.G. Rosenberg. Minimal clones I: the five types. In *Lectures in Universal Algebra (Proc. Conf. Szeged 1983)*, volume 43 of *Colloq. Math. Soc. Janos Bolyai*, pages 405–427. North-Holland, 1986.

[85] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing, STOC'78*, pages 216–226, 1978.

[86] E. Schwalb and L. Vila. Temporal constraints: a survey. *Constraints*, 3(2–3):129–149, 1998.

[87] A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures*. University of Montreal, 1986.

[88] P. van Beek and R. Dechter. Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, 44:549–566, 1997.

[89] P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[90] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In D.S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–281. Morgan Kaufmann, 1989.