On Singleton Arc Consistency for CSPs Defined by Monotone Patterns*

Clément Carbonnel[1], David A. Cohen[2], Martin C. Cooper[3], and Stanislav Živný[1]

1 **Department of Computer Science, University of Oxford, UK**
{clement.carbonnel,standa.zivny}@cs.ox.ac.uk
2 **Royal Holloway, University of London, UK**
d.cohen@rhul.ac.uk
3 **IRIT, University of Toulouse III, France**
cooper@irit.fr

Abstract

Singleton arc consistency is an important type of local consistency which has been recently shown to solve all constraint satisfaction problems (CSPs) over constraint languages of bounded width. We aim to characterise all classes of CSPs defined by a forbidden pattern that are solved by singleton arc consistency and closed under removing constraints. We identify five new patterns whose absence ensures solvability by singleton arc consistency, four of which are provably maximal and three of which generalise 2-SAT. Combined with simple counter-examples for other patterns, we make significant progress towards a complete classification.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases constraint satisfaction problems, forbidden patterns, singleton arc consistency

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.20

1 Introduction

The constraint satisfaction problem (CSP) is a declarative paradigm for expressing computational problems. An instance of the CSP consists of a number of variables to which we need to assign values from some domain. Some subsets of the variables are constrained in that they are not permitted to take all values in the product of their domains. The scope of a constraint is the set of variables whose values are limited by the constraint, and the constraint relation is the set of permitted assignments to the variables of the scope. A solution to a CSP instance is an assignment of values to variables in such a way that every constraint is satisfied, i.e. every scope is assigned to an element of the constraint relation.

The CSP has proved to be a useful technique for modelling in many important application areas from manufacturing to process optimisation, for example planning and scheduling optimisation [31], resource allocation [29], job shop problems [14] and workflow management [32]. Hence much work has been done on describing useful classes of constraints [3] and

* The authors were supported by EPSRC grant EP/L021226/1 and ANR-11-LABX-0040-CIMI within the program ANR-11-IDEX-0002-02. Stanislav Živný was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

implementing efficient algorithms for processing constraints [7]. Many constraint solvers use a form of backtracking where successive variables are assigned values that satisfy all constraints. In order to mitigate the exponential complexity of backtracking some form of pre-processing is always performed. These pre-processing techniques identify values that cannot be part of any solution in an effective way and then propagate the effects of removing these values throughout the problem instance. Of key importance amongst these pre-processing algorithms are the relatives of arc consistency propagation including generalised arc consistency (GAC) and singleton arc consistency (SAC). Surprisingly there are large classes [16, 23, 13, 28] of the CSP for which GAC or SAC are decision procedures: after establishing consistency if every variable still has a non-empty domain then the instance has a solution.

More generally, these results fit into the wider area of research aiming to identify sub-problems of the CSP for which certain polynomial-time algorithms are decision procedures. Perhaps the most natural ways to restrict the CSP is to limit the constraint relations that we allow or to limit the structure of (the hypergraph of) interactions of the constraint scopes. A set of allowed constraint relations is called a constraint language. A subset of the CSP defined by limiting the scope interactions is called a structural class.

There has been considerable success in identifying tractable constraint languages, recently yielding a full classification of the complexity of finite constraint languages [11, 33]. Techniques from universal algebra have been essential in this work as the complexity of a constraint language is characterised by a particular algebraic structure [10]. The two most important algorithms for solving the CSP over tractable constraint languages are local consistency and the few subpowers algorithm [9, 27], which generalises ideas from group theory. A necessary and sufficient condition for solvability by the few subpowers algorithm was identified in [27, 4]. The set of all constraint languages decided by local consistency was later described by Barto and Kozik [2] and independently by Bulatov [8]. Surprisingly, all such languages are in fact decided by establishing singleton arc consistency [28].

A necessary condition for the tractability of a structural class with bounded arity is that it has bounded treewidth modulo homomorphic equivalence [26]. In all such cases we decide an instance by establishing k-consistency, where k is the treewidth of the core. It was later shown that the converse holds: if a class of structures does not have treewidth k modulo homomorphic equivalence then it is not solved by k-consistency [1], thus fully characterising the strength of consistency algorithms for structural restrictions. Both language-restricted CSPs and CSPs of bounded treewidth are *monotone* in the sense that we can relax (remove constraints from) any CSP instance without affecting its membership in such a class.

Since our understanding of consistency algorithms for language and structural classes is so well advanced there is now much interest in so called hybrid classes, which are neither definable by restricting the language nor by limiting the structure. For the binary CSP, one popular mechanism for defining hybrid classes follows the considerable success of mapping the complexity landscape for graph problems in the absence of certain induced subgraphs or graph minors. Here, hybrid (binary) CSP problems are defined by forbidding a fixed set of substructures (*patterns*) from occurring in the instance [17]. This framework is particularly useful in algorithm analysis, since it allows us to identify precisely the local properties of a CSP instance that make it impossible to solve via a given polynomial-time algorithm. This approach has recently been used to obtain a pattern-based characterisation of solvability by arc consistency [23], a detailed analysis of variable elimination rules [15] and various novel tractable classes of CSP [20, 19].

Singleton arc consistency is a prime candidate to study in this framework since it is one of the most prominent incomplete polynomial-time algorithms for CSP and the highest level of

consistency (among commonly studied consistencies) that operates only by removing values from domains. This property ensures that enforcing SAC cannot introduce new patterns, which greatly facilitates the analysis. It is therefore natural to ask for which patterns, forbidding their occurrence ensures that SAC is a sound decision procedure. In this paper we make a significant contribution towards this objective by identifying five patterns which define classes of CSPs decidable by SAC. All five classes are monotone, and we show that only a handful of open cases separates us from an essentially full characterisation of monotone CSP classes decidable by SAC and definable by a forbidden pattern. Some of our results rely on a novel proof technique which follows the *trace* of a successful run of the SAC algorithm to dynamically identify redundant substructures in the instance and construct a solution.

The structure of the paper is as follows. In Section 2 we provide essential definitions and background theory. In Section 3 we state the main results. The rest of the paper includes some of the proofs. All remaining proofs are provided in the long version [12].

2 Preliminaries

CSP A *binary CSP instance* is a triple $I = (X, D, C)$, where X is a finite set of variables, D is a finite domain, each variable $x \in X$ has its own domain of possible values $D(x) \subseteq D$, and $C = \{R(x, y) \mid x, y \in X, x \neq y\}$, where $R(x, y) \subseteq D^2$, is the set of constraints. We assume, without loss of generality, that each pair of variables $x, y \in X$ is constrained by a constraint $R(x, y)$. (Otherwise we set $R(x, y) = D(x) \times D(y)$.) We also assume that $(a, b) \in R(x, y)$ if and only if $(b, a) \in R(y, x)$. A constraint is *trivial* if it contains the Cartesian product of the domains of the two variables. By *deleting* a constraint we mean replacing it with a trivial constraint. The *projection* $I[X']$ of a binary CSP instance I on $X' \subseteq X$ is obtained by removing all variables in $X \backslash X'$ and all constraints $R(x, y)$ with $\{x, y\} \not\subseteq X'$. A *partial solution* to a binary CSP instance on $X' \subseteq X$ is an assignment s of values to variables in X' such that $s(x) \in D(x)$ for all $x \in X'$ and $(s(x), s(y)) \in R(x, y)$ for all constraints $R(x, y)$ with $x, y \in X'$. A *solution* to a binary CSP instance is a partial solution on X.

An assignment (x, a) is called a *point*. If $(a, b) \in R(x, y)$, we say that the assignments $(x, a), (y, b)$ (or more simply a, b) are *compatible* and that ab is a *positive edge*, otherwise a, b are *incompatible* and ab is a *negative edge*. For simplicity of notation we can assume that variable domains are disjoint, so that using a as a shorthand for (x, a) is unambiguous. We say that $a \in D(x)$ has a *support* at variable y if $\exists b \in D(y)$ such that ab is a positive edge.

The constraint graph of a CSP instance with variables X is the graph $G = (X, E)$ such that $(x, y) \in E$ if $R(x, y)$ is non-trivial. The *degree* of a variable x in a CSP instance is the degree of x in the constraint graph of the instance.

Arc Consistency A domain value $a \in D(x)$ is *arc consistent* if it has a support at every other variable. A CSP instance is *arc consistent* (AC) if every domain value is arc consistent.

Singleton Arc Consistency Singleton arc consistency is stronger than arc consistency (but weaker than strong path consistency [30]). A domain value $a \in D(x)$ in a CSP instance I is *singleton arc consistent* if the instance obtained from I by removing all domain values $b \in D(x)$ with $a \neq b$ can be made arc consistent without emptying any domain. A CSP instance is *singleton arc consistent* (SAC) if every domain value is singleton arc consistent.

Establishing Consistency Domain values that are not arc consistent or not singleton arc consistent cannot be part of a solution so can safely be removed. For a binary CSP instance

with domain size d, n variables and e non-trivial constraints there are $O(ed^2)$ algorithms for establishing arc consistency [6] and $O(ned^3)$ algorithms for establishing singleton arc consistency [5]. These algorithms repeatedly remove inconsistent values from domains.

SAC *decides* a CSP instance if, after establishing singleton arc consistency, non-empty domains for all variables guarantee the existence of a solution. SAC decides a class of CSP instances if SAC decides every instance from the class.

Neighbourhood Substitutability If $a, b \in D(x)$, then a is *neighbourhood substitutable* by b if there is no c such that ac is a positive edge and bc a negative edge: such values a can be deleted from $D(x)$ without changing the satisfiability of the instance since a can be replaced by b in any solution [25]. Similarly, removing neighbourhood substitutable values cannot destroy (singleton) arc consistency.

Patterns In a binary CSP instance each constraint decides, for each pair of values in D, whether it is allowed. Hence a binary CSP can also be defined as a set of points $X \times D$ together with a compatibility function that maps each edge, $((x, a), (y, b))$ with $x \neq y$, into the set $\{\text{negative}, \text{positive}\}$. A *pattern* extends the notion of a binary CSP instance by allowing the compatibility function to be partial. A pattern P *occurs* (as a subpattern) in an instance I if there is mapping from the points of P to the points of I which respects variables (two points are mapped to points of the same variable in I if and only if they belong to the same variable in P) and maps negative edges to negative edges, and positive edges to positive edges. A set of patterns occurs in an instance I if at least one pattern in the set occurs in I.

We use the notation $\mathrm{CSP}(\overline{P})$ for the set of binary instances in which P does not occur as a subpattern. A pattern P is *SAC-solvable* if SAC decides $\mathrm{CSP}(\overline{P})$. It is worth observing that $\mathrm{CSP}(\overline{P})$ is closed under the operation of establishing (singleton) arc consistency. A pattern P is *tractable* if $\mathrm{CSP}(\overline{P})$ can be solved in polynomial time.

Points (x, a) and (x, b) in a pattern are *mergeable* if there is no point (y, c) such that ac is positive and bc is negative or vice versa. For each set of patterns there is an equivalent set of patterns without mergeable points which occur in the same set of instances.

A point (x, a) in a pattern is called *dangling* if there is at most one b such that ab is a positive edge and no c such that ac is a negative edge. Dangling points are redundant when considering the occurrence of a pattern in an arc consistent CSP instance.

A pattern is called *irreducible* if it has no dangling points and no mergeable points [20]. When studying algorithms that are at least as strong as arc consistency, a classification with respect to forbidden *sets* of irreducible patterns is equivalent to a classification with respect to all forbidden sets of patterns. For this reason classifications are often established with respect to irreducible patterns even if only classes definable by forbidding a single pattern are considered [20, 23], as we do in the present paper.

3 Results

Call a class $\mathcal{C}$ of CSP instances *monotone* if deleting any constraint from an instance $I \in \mathcal{C}$ produces another instance in $\mathcal{C}$. For example, language classes and bounded treewidth classes are monotone. An interesting research direction is to study those monotone classes defined by a forbidden pattern which are solved by singleton arc consistency, both in order to uncover new tractable classes and to better understand the strength of SAC.

We call a pattern *monotone* if when forbidden it defines a monotone class. Monotone patterns can easily be seen to correspond to exactly those patterns in which positive edges

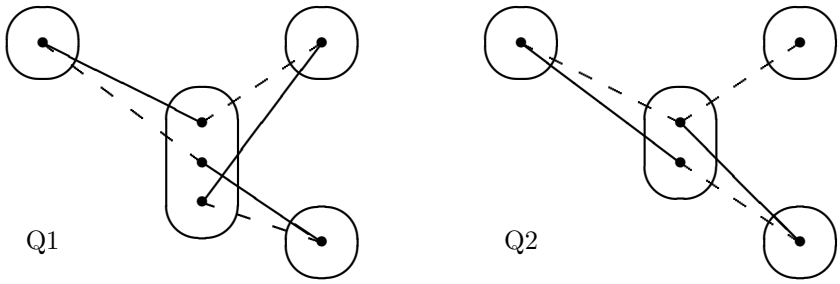

Figure 1 All degree-3 irreducible monotone patterns solved by SAC must occur in at least one of these patterns.

only occur in constraints which have at least one negative edge.

Consider the monotone patterns Q1 and Q2 shown in Figure 1, patterns R5, R8 shown in Figure 2, and pattern R7- shown in Figure 3.

▶ **Theorem** (Main). *The patterns Q1, Q2, R5, R8, and R7- are SAC-solvable.*

In order to prove the SAC-solvability of Q1, R8 and R7- we use the same idea of following the trace of arc consistency and argue that the resulting instance is not too complicated. While the same idea is behind the proofs of all three patterns, the technical details differ.

In the remaining two cases we identify an operation that preserves SAC and satisfiability, does not introduce the pattern and after repeated application necessarily produces an equivalent instance which is solved by SAC. In the case of R5, the operation is simply removing any constraint. In the case of Q2, the operation is BTP-merging [19].

▶ Remark. The full version of this paper [12] tells us that any *monotone* and *irreducible* pattern solvable by SAC must occur in at least one of the patterns shown in Figures 1 and 2. By this analysis, we have managed to reduce the number of remaining cases to a handful. Our main result shows that some of these are SAC-solvable. In particular, the patterns Q1, Q2, R5, and R8 are maximal in the sense that adding anything to them would give a pattern that is either non-monotone or not solved by SAC.

▶ Remark. We point out that certain interesting forbidden patterns, such as BTP [21], NegTrans [22], and EMC [23] are not monotone. On the other hand, the patterns T1, . . . ,T5 shown in Figure 4 *are* monotone. Patterns T1, . . . ,T5 were identified in [20] as the maximal irreducible tractable patterns on two connected constraints. We show in [12] that T1 is not solved by SAC. Our main result implies (since R8 contains T4 and T5) that both T4 and T5 are solved by SAC. It can easily be shown, from Lemma 12 and [20, Lemma 25], that T2 is solved by SAC, and we provide in [12] a simple proof that T3 is solved by SAC as well. Hence, we have characterised all 2-constraint irreducible patterns solvable by SAC.

▶ Remark. Observe that Q1 does not occur in any binary CSP instance in which all degree 3 or more variables are Boolean. This shows that 2-SAT is a strict subset of CSP($\overline{\text{Q1}}$). This class is incomparable with language-based generalisations of 2-SAT, such as the class ZOA [18], since in CSP($\overline{\text{Q1}}$) degree-2 variables can be constrained by arbitrary constraints. Indeed, instances in CSP($\overline{\text{Q1}}$) can have an arbitrary constraint on the pair of variables x, y, where x is of arbitrary degree and of arbitrary domain size if for all variables $z \notin \{x, y\}$, the constraint on the pair of variables x, z is of the form $(x \in S) \vee (z \in T_z)$ where S is fixed (i.e. independent of z) but T_z is arbitrary. R8 and R7- generalise T4 and CSP($\overline{\text{T4}}$) generalises ZOA [20], so CSP($\overline{\text{R8}}$) and CSP($\overline{\text{R7-}}$) are strict generalisations of ZOA.

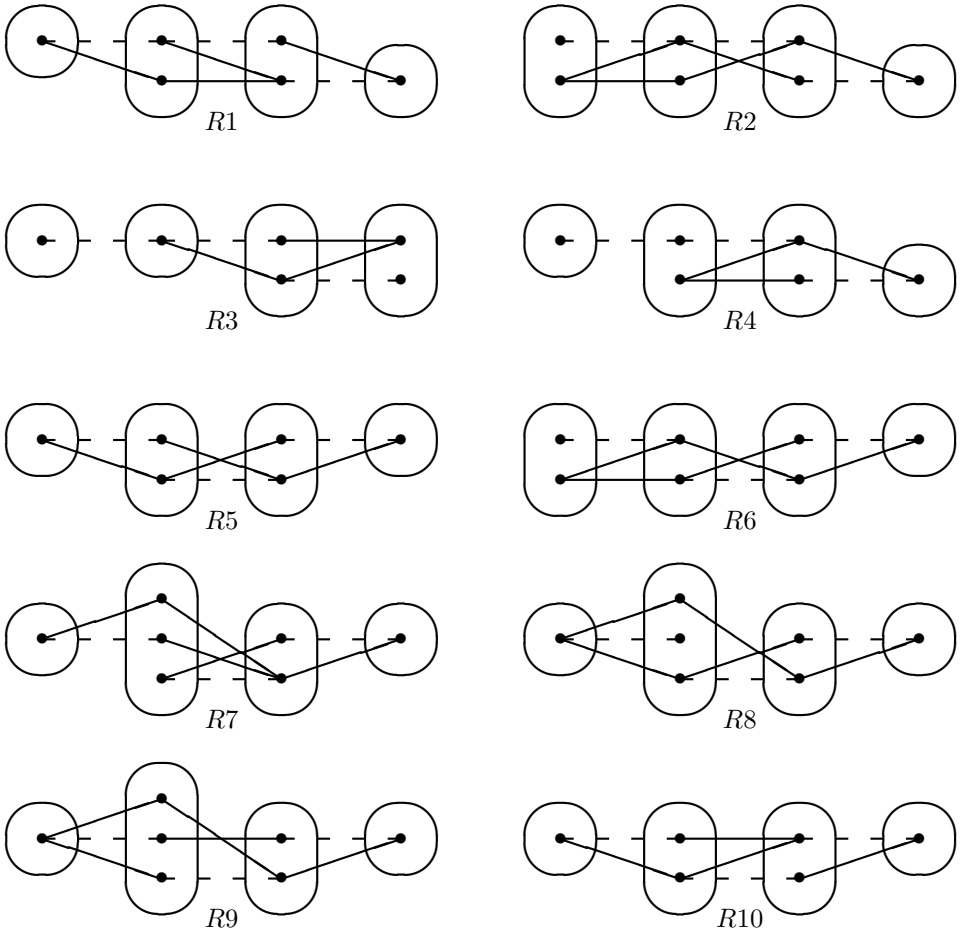

Figure 2 All degree-2 irreducible monotone patterns solved by SAC must occur in at least one of these patterns.

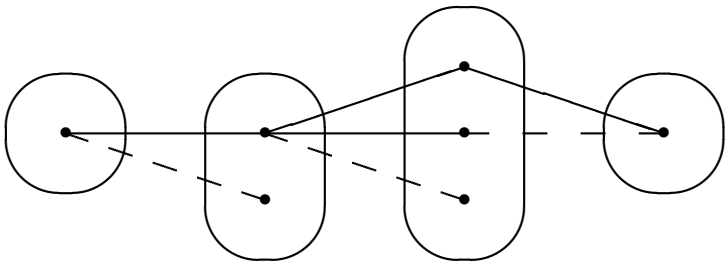

Figure 3 The pattern R7-, a subpattern of R7.

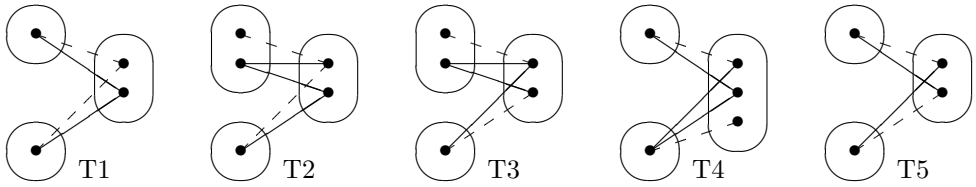

Figure 4 The set of tractable 2-constraint irreducible patterns.

4 Notation for the Trace Technique

Given a singleton arc consistent instance I, a variable x and a value $v \in D(x)$, we denote by I_{xv} the instance obtained by assigning x to v (that is, setting $D(x) = \{v\}$) and enforcing arc consistency. To avoid confusion with the original domains, we will use $D_{xv}(y)$ to denote the domain of the variable y in I_{xv}. For our proofs we will assume that arc consistency has been enforced using a straightforward algorithm that examines the constraints one at a time and removes the points that do not have a support until a fixed point is reached. We will be interested in the *trace* of this algorithm, given as a chain of propagations:

$$(P_{xv}) : (x \to y_0), (x_1 \to y_1), (x_2 \to y_2), \ldots, (x_p \to y_p)$$

where $x_i \to y_i$ means that the algorithm has inferred a change in the domain of y_i when examining the constraint $R(x_i, y_i)$. We define a map $\rho : (P_{xv}) \mapsto 2^D$ that maps each $(x_i \to y_i) \in (P_{xv})$ to the set of values that were removed from $D_{xv}(y_i)$ at this step. Without loss of generality, we assume that the steps $(x_i \to y_i)$ such that the pruning of $\rho(x_i \to y_i)$ from $D_{xv}(y_i)$ does not incur further propagation are performed last.

We denote by $S_{(P_{xv})}$ the set of variables that appear in (P_{xv}). Because I was (singleton) arc consistent before x was assigned, we have $S_{(P_{xv})} = \{x\} \cup \{y_i \mid i \geq 0\}$. We rename the elements of $S_{(P_{xv})}$ as $\{p_i \mid i \geq 0\}$ where the index i denotes the order of first appearance in (P_{xv}). Finally, we use $S^I_{(P_{xv})}$ to denote the set of *inner* variables, that is, the set of all variables $p_j \in S_{(P_{xv})}$ for which there exists $p_r \in S_{(P_{xv})}$ such that $(p_j \to p_r) \in (P_{xv})$.

5 Tractability of Q1

Consider the pattern Q1 shown in Figure 1. Let $I \in \mathrm{CSP}(\overline{\mathrm{Q1}})$ be a singleton arc consistent instance, x be any variable and v be any value in the domain of x. Our proof of the SAC-decidability of $\mathrm{CSP}(\overline{\mathrm{Q1}})$ uses the trace of the arc consistency algorithm to determine a subset of variables in the vicinity of x such that (i) the projection of I_{xv} to this particular subset is satisfiable, (ii) those variables do not interact too much with the rest of the instance and (iii) the projections of I_{xv} and I on the other variables are almost the same. We then use these three properties to show that the satisfiability of I is equivalent to that of an instance with fewer variables, and we repeat the operation until the smaller instance is trivially satisfiable.

The following lemma describes the particular structure of I_{xv} around the variables whose domain has been reduced by arc consistency. Note that a non-trivial constraint in I can be trivial in I_{xv} because of domain changes.

▶ **Lemma 1.** *Consider the instance I_{xv}. Every variable $p_i \in S^I_{(P_{xv})}$ is in the scope of at most two non-trivial constraints, which must be of the form $R(p_j, p_i)$ and $R(p_i, p_r)$ with $j < i$, $(p_j \to p_i) \in (P_{xv})$ and $(p_i \to p_r) \in (P_{xv})$.*

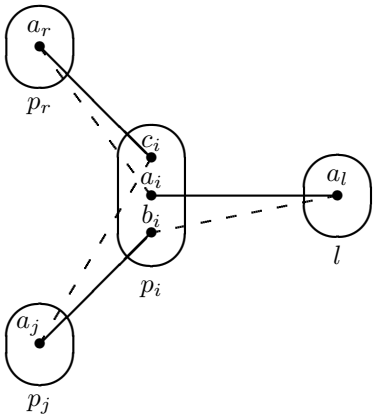

Figure 5 The occurence of Q1 in the proof of Lemma 1.

Proof. The claim is true for $p_0 = x$ as every constraint incident to x is trivial. Otherwise, let $p_i \in S^I_{(P_{xv})}$ be such that $p_i \neq x$. Let p_j, $j < i$ be such that $(p_j \to p_i)$ occurs first in (P_{xv}). Because $p_i \in S^I_{(P_{xv})}$ and we assumed that the arc consistency algorithm performs the pruning that do not incur further propagation last, we know that there exists $c_i \in \rho(p_j \to p_i)$ and $p_r \in S_{(P_{xv})}$ with $(p_i \to p_r) \in (P_{xv})$ such that the pruning of c_i from $D_{xv}(p_i)$ allows the pruning of some $a_r \in \rho(p_i \to p_r)$ from the domain of p_r. It follows that $(c_i, a_r) \in R(p_i, p_r)$, $(v_i, a_r) \notin R(p_i, p_r)$ for any $v_i \in D_{xv}(p_i)$ and $(v_j, c_i) \notin R(p_j, p_i)$ for any $v_j \in D_{xv}(p_j)$. Moreover, a_r was a support for c_i at p_r when c_i was pruned so we know that $p_j \neq p_r$.

For the sake of contradiction, let us assume that there exists a constraint $R(p_i, l)$ with $l \notin \{p_j, p_r\}$ that is not trivial. In particular, there exist $a_i, b_i \in D_{xv}(p_i)$ and $a_l \in D_{xv}(l)$ such that $(a_i, a_l) \in R(p_i, l)$ but $(b_i, a_l) \notin R(p_i, l)$. Furthermore, a_r was removed by arc consistency when inspecting the constraint $R(p_i, p_r)$ so $(a_i, a_r) \notin R(p_i, p_r)$. I_{xv} is arc consistent so there exists some $a_j \in D_{xv}(p_j)$ such that $(a_j, b_i) \in R(p_j, p_i)$, and since $c_i \in \rho(p_j \to p_i)$ we have $(a_j, c_i) \notin R(p_j, p_i)$. At this point we have reached the desired contradiction as Q1 occurs on (p_i, p_j, p_r, l) with p_i being the middle variable (see Figure 5). ◀

Given a subset S of variables, an *S-path* between two variables y_1 and y_2 is a path $R(y_1, x_2), R(x_2, x_3), \ldots, R(x_k, y_2)$ of non-trivial constraints with $k \geq 2$ and $x_2, \ldots, x_k \in S$.

▶ **Lemma 2.** *Consider the instance I_{xv}. There is no $(S^I_{(P_{xv})})$-path between two variables in $X \backslash S^I_{(P_{xv})}$ and there is no cycle of non-trivial constraints in $I_{xv}[S^I_{(P_{xv})}]$.*

Proof. Let $y_1, y_2 \in X \backslash S^I_{(P_{xv})}$ and assume for the sake of contradiction that a $(S^I_{(P_{xv})})$-path $R(y_1, x_2), R(x_2, x_3), \ldots, R(x_{k-1}, y_2)$ exists. Let $p_i \in \{x_2, \ldots, x_{k-1}\}$ be such that i is minimum. Since p_i is in the scope of two non-trivial constraints in this path, it follows from Lemma 1 that p_i is in the scope of exactly two non-trivial constraints, one of which is of the form $R(p_j, p_i)$ with $j < i$ and $(p_j \to p_i) \in (P_{xv})$. It follows from $(p_j \to p_i) \in (P_{xv})$ that $p_j \in S^I_{(P_{xv})}$ and hence p_j is not an endpoint of the path, and then $j < i$ contradicts the minimality of i. The second part of the claim follows from the same argument, by considering a cycle as a $(S^I_{(P_{xv})})$-path $R(x_1, x_2), R(x_2, x_3), \ldots, R(x_{k-1}, x_1)$ with $x_1 \in (S^I_{(P_{xv})})$ and defining p_i as the variable among $\{x_1, \ldots, x_{k-1}\}$ with minimum index. ◀

▶ **Lemma 3.** *I_{xv} has a solution if and only if $I_{xv}[X \backslash S^I_{(P_{xv})}]$ has a solution.*

Proof. The "only if" implication is trivial, so we focus on the other direction. Suppose that there exists a solution ϕ to $I_{xv}[X \backslash S^I_{(P_{xv})}]$. Let Y be a set of variables initialized to $X \backslash S^I_{(P_{xv})}$.

We will grow Y with the invariants that (i) we know a solution ϕ to $I_{xv}[Y]$, and (ii) there is no $(X\backslash Y)$-path between two variables in Y (which is true at the initial state by Lemma 2).

If there is no non-trivial constraint between $X\backslash Y$ and Y then I_{xv} is satisfiable if and only if $I_{xv}[X\backslash Y]$ is. By construction $X\backslash Y \subseteq S^I_{(P_{xv})}$ and by Lemma 2 we know that $I_{xv}[X\backslash Y]$ has no cycle of non-trivial constraints. Because $I_{xv}[X\backslash Y]$ is arc consistent and acyclic it has a solution [24], and we can conclude that in this case I_{xv} has a solution.

Otherwise, let $p_i \in X\backslash Y$ be such that there exists a non-trivial constraint between p_i and some variable $p_r \in Y$. By (ii), this non-trivial constraint must be unique (with respect to p_i) as otherwise we would have a $(X\backslash Y)$-path between two variables in Y. By arc consistency, there exists $a_i \in D_{xv}(p_i)$ such that $(a_i, \phi(p_r)) \in R(p_i, p_r)$; because this non-trivial constraint is unique, setting $\phi(p_i) = a_i$ yields a solution to $I_{xv}[Y \cup \{p_i\}]$. Because any $(X\backslash(Y \cup \{p_i\}))$-path between two variables in $Y \cup \{p_i\}$ would extend to a $(X\backslash Y)$-path between Y variables by going through p_i, we know that no such path exists. Then $Y \leftarrow Y \cup \{p_i\}$ satisfies both invariants, so we can repeat the operation until we have a solution to the whole instance or all constraints between Y and $X\backslash Y$ are trivial. In both cases I_{xv} has a solution. ◀

▶ **Lemma 4.** *I has a solution if and only if $I[X\backslash S^I_{(P_{xv})}]$ has a solution.*

Proof. Again the "only if" implication is trivial so we focus on the other direction. Let us assume for the sake of contradiction that $I[X\backslash S^I_{(P_{xv})}]$ has a solution but I does not. In particular this implies that I_{xv} does not have a solution, and then by Lemma 3 we know that $I_{xv}[X\backslash S^I_{(P_{xv})}]$ has no solution either. We define Z as a subset of $X\backslash S^I_{(P_{xv})}$ of minimum size such that $I_{xv}[Z]$ has no solution. Observe that $I_{xv}[Z]$ can only differ from $I[Z]$ by having fewer values in the domain of the variables in $S_{(P_{xv})}$. Let ϕ be a solution to $I[Z]$ such that $\phi(y) \in D_{xv}(y)$ for as many variables y as possible. Because ϕ is not a solution to $I_{xv}[Z]$, there exists $p_r \in Z \cap S_{(P_{xv})}$ and $p_j \in S^I_{(P_{xv})}$ such that $(p_j \to p_r) \in (P_{xv})$ and $\phi(p_r) \in \rho(p_j \to p_r)$ (recall that $\rho(p_j \to p_r)$ is the set of points removed by the AC algorithm in the domain of p_r at step $(p_j \to p_r)$). By construction, $p_j \notin Z$.

First, let us assume that there exists a variable $y \in Z$, $y \neq p_r$ such that there is no $a_r \in D_{xv}(p_r)$ with $(\phi(y), a_r) \in R(y, p_r)$. This implies, in particular, that $\phi(y) \notin D_{xv}(y)$. We first prove that $R(y, p_r)$ and $R(p_j, p_r)$ are the only possible non-trivial constraints involving p_r in I_{xv}. If there exists a fourth variable z such that $R(p_r, z)$ is non-trivial in I_{xv}, then there exist $a_r, b_r \in D_{xv}(p_r)$ and $a_z \in D_{xv}(z)$ such that $(a_r, a_z) \in R(p_r, z)$ but $(b_r, a_z) \notin R(p_r, z)$. By assumption we have $(\phi(y), a_r) \notin R(y, p_r)$ and $(\phi(y), \phi(p_r)) \in R(y, p_r)$. Finally, b_r has a support $a_j \in D_{xv}(p_j)$ and $\phi(p_r) \in \rho(p_j \to p_r)$ so we have $(a_j, a_r) \in R(p_j, p_r)$ but $(a_j, \phi(p_r)) \notin R(p_j, p_r)$. This produces Q1 on (p_r, y, p_j, z) with p_r being the middle variable. Therefore, we know that $R(y, p_r)$ and $R(p_j, p_r)$ are the only possible non-trivial constraints involving p_r in I_{xv}. However, in this case the variable p_r has only one incident non-trivial constraint in $I_{xv}[Z]$, and hence $I_{xv}[Z]$ has a solution if and only if $I_{xv}[Z\backslash p_r]$ has one. This contradicts the minimality of Z, and for the rest of the proof we can assume that for every $y \in Z$ there exists some $a_r \neq \phi(p_r)$ such that $a_r \in D_{xv}(p_r)$ and $(\phi(y), a_r) \in R(y, p_r)$.

Now, let $y \in Z$ be such that $y \neq p_r$ and $|\{b \in D_{xv}(p_r) \mid (\phi(y), b) \in R(y, p_r)\}|$ is minimum. By the argument above, there exists $a_r \in D_{xv}(p_r)$ such that $(\phi(y), a_r) \in R(y, p_r)$ and $a_r \neq \phi(p_r)$. By hypothesis setting $\phi(p_r) = a_r$ would violate at least one constraint in $I[Z]$, so there exists some variable $z \in Z$, $z \neq y$ such that $(\phi(z), a_r) \notin R(z, p_r)$. Furthermore, by arc consistency of I_{xv} there exists $a_j \in D_{xv}(p_j)$ such that $(a_j, a_r) \in R(p_j, p_r)$. Recall that we picked p_j in such a way that $\phi(p_r) \in \rho(p_j \to p_r)$, and so we have $(a_j, \phi(p_r)) \notin R(p_j, p_r)$. We summarize what we have in Figure 6. Observe that unless Q1 occurs, for every $b_r \in D_{xv}(p_r)$ such that $(\phi(y), b_r) \notin R(y, p_r)$ we also have $(\phi(z), b_r) \notin R(z, p_r)$. However, recall that

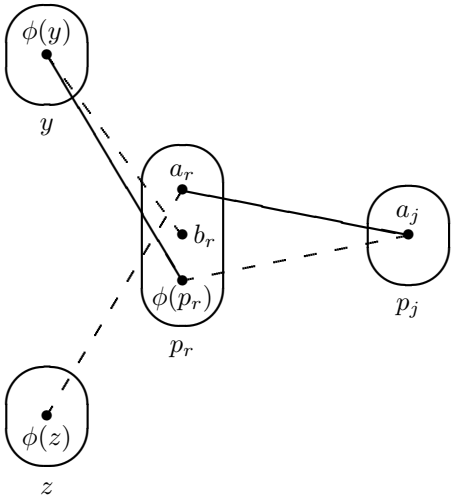

Figure 6 Some positive and negative edges between y, z, p_j and p_r. The positive edges $(\phi(y), a_r)$ and $(\phi(z), \phi(p_r))$ are omitted for clarity; b_r is any value in $D_{xv}(p_r)$ that is not compatible with $\phi(y)$.

$(\phi(y), a_r) \in R(y, p_r)$ so $\phi(z)$ is compatible with strictly fewer values in $D_{xv}(p_r)$ than $\phi(y)$. This contradicts the choice of y. It follows that setting $\phi(p_r) = a_r$ cannot violate any constraint in $I[Z]$, which is impossible by our choice of ϕ - a final contradiction. ◀

▶ **Theorem 5.** $\mathrm{CSP}(\overline{Q1})$ *is solved by singleton arc consistency.*

Proof. Let $I \in \mathrm{CSP}(\overline{\mathrm{Q1}})$ be singleton arc consistent. Pick any variable x and value $v \in D(x)$. By singleton arc consistency the instance I_{xv} does not have any empty domains. By Lemma 4, I has a solution if and only if $I[X \backslash S^I_{(P_{xv})}]$ has one. Because $I[X \backslash S^I_{(P_{xv})}]$ is singleton arc consistent as well and $S^I_{(P_{xv})} \neq \emptyset$ we can repeat the procedure until $X \backslash S^I_{(P_{xv})}$ is empty, at which point we may conclude that I has a solution. ◀

6 Tractability of R8 and R7-

Q1 and R8 (Figure 2) are structurally dissimilar, but the idea of using I_{xv} and the trace of the arc consistency algorithm to extract variables from I without altering satisfiability works in the case of R8 as well. We define a *star* to be a non-empty set of constraints whose scopes all intersect. The *centers* of a star are its variables of highest degree (every star with three or more variables has a unique center). The following lemma is the R8 analog of Lemma 1; the main differences are a slightly stronger prerequisite (no neighbourhood substitutable values) and that arc consistency leaves stars of non-trivial constraints instead of paths.

▶ **Lemma 6.** *Let $I = (X, D, C) \in \mathrm{CSP}(\overline{R8})$ be singleton arc consistent. Let $x \in X$, $v \in D(x)$ and consider the instance I_{xv}. After the removal of every neighbourhood substitutable value, every connected component of non-trivial constraints that intersect with $S_{(P_{xv})}$ is a star with a center in $S_{(P_{xv})}$.*

In the proof of SAC-solvability of Q1, only inner variables are extracted from the instance. The above lemma suggests that in the case of R8 it is more convenient to extract all variables in $S_{(P_{xv})}$, plus any variable that can be reached from those via a non-trivial constraint.

▶ **Lemma 7.** *Let $I = (X, D, C) \in \mathrm{CSP}(\overline{R8})$ be singleton arc consistent. Let $x \in X$, $v \in D(x)$ and consider the instance I_{xv}. There exists a partition (X_1, X_2) of X such that*

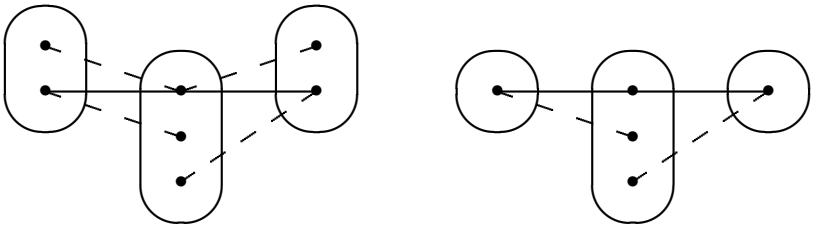

Figure 7 The patterns $\hat{\mathrm{M}}$ (left) and V_2 (right).

- $S_{(P_{xv})} \subseteq X_1$;
- $\forall (x, y) \in X_1 \times X_2$, $R(x, y)$ *is trivial;*
- *Every connected component of non-trivial constraints with scopes subsets of* X_1 *is a star.*

▶ **Theorem 8.** $\mathrm{CSP}(\overline{R8})$ *is solved by singleton arc consistency.*

Our proof of the SAC-solvability of R7- (Figure 3) follows a similar reasoning, with two main differences. First, branching on just any variable-value pair (as we did for Q1 and R8) may lead to a subproblem that is *not* solved by arc consistency. However, once the right assignment is made the reward is much greater as all constraints involving a variable whose domain has been reduced by arc consistency must become trivial *except at most one.*

Finding out which variable we should branch on is tricky. Our proof works by induction, and the ideal starting point is a substructure corresponding to a particular pattern $\hat{\mathrm{M}}$ (Figure 7). However, $\hat{\mathrm{M}}$ is an NP-hard pattern [17] so it may not occur at all in the instance. To handle this problem we define a weaker pattern V_2 (Figure 7), whose absence implies SAC-solvability (because it is a sub-pattern of T4), and we show that if the induction started from V_2 breaks then $\hat{\mathrm{M}}$ must occur somewhere - a win-win situation.

▶ **Lemma 9.** *Let* $I = (X, D, C) \in \mathrm{CSP}(\overline{R7\text{-}})$ *be singleton arc consistent. Let* $x \in X$ *be such that* $\hat{M}$ *occurs on* (y, x, z) *with* x *the middle variable and* v *be the value in* $D(x)$ *that is the meet point of the two positive edges. Then every constraint whose scope contains a variable in* $S_{(P_{xv})}$ *is trivial in* I_{xv}*, except possibly* $R(y, z)$*.*

▶ **Lemma 10.** *Let* $I = (X, D, C) \in \mathrm{CSP}(\overline{\hat{M}}) \cap \mathrm{CSP}(\overline{R7\text{-}})$ *be singleton arc consistent. Let* $x \in X$ *be such that* V_2 *occurs on* (y, x, z) *with* x *the middle variable and* v *be the value in* $D(x)$ *that is the meet point of the two positive edges. Then every constraint whose scope contains a variable in* $S_{(P_{xv})}$ *is trivial in* I_{xv}*, except possibly* $R(y, z)$*.*

▶ **Theorem 11.** $\mathrm{CSP}(\overline{R7\text{-}})$ *is solved by singleton arc consistency.*

7 Tractability of Q2 and R5

For our last two proofs of SAC-decidability, we depart from the trace technique. Our fundamental goal, however, remains the same: find an operation which shrinks the instance without altering satisfiability, introducing the pattern or losing singleton arc consistency. For Q2 this operation is BTP-merging [19] and for R5 it is removing constraints.

Consider the pattern V^- shown in Figure 8(a). We say that V^- occurs at point a or at variable x if $a \in D(x)$ is the central point of the pattern in the instance. The pattern V^- is known to be tractable since all instances in $\mathrm{CSP}(\overline{\mathrm{V}^-})$ satisfy the joint-winner property [22]. However, we show a slightly different result, namely that singleton arc consistency is sufficient to solve instances in which V^- only occurs at degree-2 variables.

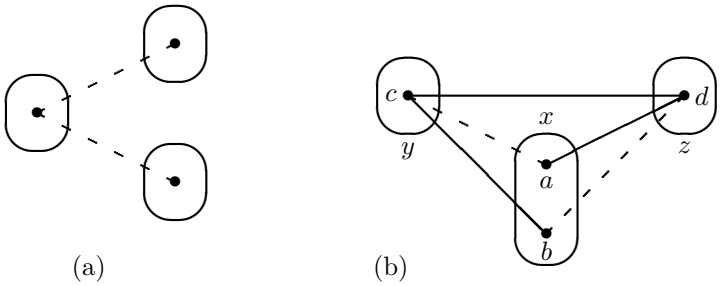

Figure 8 (a) The pattern V^- and (b) the associated broken-triangle pattern (BTP).

▶ **Lemma 12.** *Instances in which V^- only occurs at degree-2 variables are solved by singleton arc consistency.*

Two values $a, b \in D(x)$ are *BTP-mergeable* [19] if there are not two other distinct variables $y, z \neq x$ such that $\exists c \in D(y)$, $\exists d \in D(z)$ with ad, bc, cd positive edges and ac, bd negative edges as shown in Figure 8(b). The *BTP-merging* operation consists in merging two BTP-mergeable points $a, b \in D(x)$: the points a, b are replaced by a new point c in $D(x)$ such that for all other variables $w \neq x$ and for all $d \in D(w)$, cd is a positive edge if at least one of ad, bd was a positive edge (a negative edge otherwise). BTP-merging preserves satisfiability [19].

▶ **Lemma 13.** *Let P be a pattern in which no point occurs in more than one positive edge. Then the BTP-merging operation cannot introduce the pattern P in an instance $I \in \mathrm{CSP}(\overline{P})$.*

Since Q2 has no point which occurs in more than one positive edge, we can deduce from Lemma 13 that Q2 cannot be introduced by BTP-merging. We then combine this property with Lemma 12 by proving that V^- can only occur at degree-2 variables in any instance of $\mathrm{CSP}(\overline{\mathrm{Q2}})$ with no BTP-mergeable values.

▶ **Theorem 14.** $\mathrm{CSP}(\overline{Q2})$ *is solved by singleton arc consistency.*

That only leaves R5. Removing constraints cannot introduce R5 because it is a monotone pattern, so we can apply repeatedly the following lemma to obtain our last result.

▶ **Lemma 15.** *If the pattern R5 does not occur in a singleton arc consistent binary CSP instance I, then removing any constraint leaves the satisfiability of I invariant.*

▶ **Theorem 16.** $\mathrm{CSP}(\overline{R5})$ *is solved by singleton arc consistency.*

Note that Lemma 15 is technically true for all SAC-solvable patterns (not only R5); this is simply the only case where we are able to prove it directly.

8 Conclusion

We have established SAC-solvability of five novel classes of binary CSPs defined by a forbidden pattern, three of which are generalisations of 2SAT. For monotone patterns (defining classes of CSPs closed under removing constraints), there remains only a relatively small number of irreducible patterns whose SAC-solvability is still open. In addition to settling the remaining patterns, a possible line of future work is to study *sets* of patterns or partially-ordered patterns [23] that give rise to SAC-solvable (monotone) classes of CSPs.

References

1 Albert Atserias, Andrei Bulatov, and Victor Dalmau. On the power of k-consistency. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, pages 279–290, 2007. doi:10.1007/978-3-540-73420-8_26.

2 Libor Barto and Marcin Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *Journal of the ACM*, 61(1), 2014. Article No. 3. doi:10.1145/2556646.

3 Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. *Global Constraint Catalog*, 2017. http://sofdem.github.io/gccat/gccat/titlepage.html.

4 Joel Berman, Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Varieties with few subalgebras of powers. *Transactions of the American Mathematical Society*, 362(3):1445–1473, 2010. doi:10.1090/S0002-9947-09-04874-0.

5 Christian Bessiere and Romuald Debruyne. Theoretical analysis of singleton arc consistency and its extensions. *Artificial Intelligence*, 172(1):29–41, 2008. doi:10.1016/j.artint.2007.09.001.

6 Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005. doi:10.1016/j.artint.2005.02.004.

7 Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999. doi:10.1016/S0377-2217(98)00364-6.

8 Andrei Bulatov. Bounded relational width. 2009. Unpublished manuscript.

9 Andrei Bulatov and Víctor Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, July 2006. doi:10.1137/050628957.

10 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, March 2005. doi:10.1137/S0097539700376676.

11 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

12 Clément Carbonnel, David A. Cohen, Martin C. Cooper, and Stanislav Živný. On singleton arc consistency for CSPs defined by monotone patterns. Technical report, April 2017. URL: http://arxiv.org/abs/1704.06215.

13 Hubie Chen, Victor Dalmau, and Berit Grußien. Arc consistency and friends. *Journal of Logic and Computation*, 23(1):87, 2013. doi:10.1093/logcom/exr039.

14 Cheng-Chung Cheng and Stephen F. Smith. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research*, 70(0):327–357, 1997. doi:10.1023/A:1018934507395.

15 David A. Cohen, Martin C. Cooper, Guillaume Escamocher, and Stanislav Živný. Variable and value elimination in binary constraint satisfaction via forbidden patterns. *Journal of Computer and System Sciences*, 81(7):1127–1143, 2015. doi:10.1016/j.jcss.2015.02.001.

16 David A. Cohen and Peter G. Jeavons. The power of propagation: when gac is enough. *Constraints*, 22(1):3–23, Jan 2017. doi:10.1007/s10601-016-9251-0.

17 Martin C. Cooper, David A. Cohen, Páidí Creed, Dániel Marx, and András Z. Salamon. The tractability of CSP classes defined by forbidden patterns. *Journal of Artificial Intelligence Research*, 45:47–78, 2012. doi:10.1613/jair.3651.

18 Martin C. Cooper, David A. Cohen, and Peter G. Jeavons. Characterising Tractable Constraints. *Artificial Intelligence*, 65:347–361, 1994. doi:10.1016/0004-3702(94)90021-3.

19 Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, and Bruno Zanuttini. Broken triangles: From value merging to a tractable class

of general-arity constraint satisfaction problems. *Artificial Intelligence*, 234:196–218, 2016. doi:10.1016/j.artint.2016.02.001.

20 Martin C. Cooper and Guillaume Escamocher. Characterising the complexity of constraint satisfaction problems defined by 2-constraint forbidden patterns. *Discrete Applied Mathematics*, 184:89–113, 2015. doi:10.1016/j.dam.2014.10.035.

21 Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, 174(9–10):570–584, 2010. doi:10.1016/j.artint.2010.03.002.

22 Martin C. Cooper and Stanislav Živný. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, 175(9-10):1555–1569, 2011. doi:10.1016/j.artint.2011.02.003.

23 Martin C. Cooper and Stanislav Živný. The power of arc consistency for CSPs defined by partially-ordered forbidden patterns. In *Proceedings of the 31st Annual ACM/IEE Symposium on Logic in Computer Science (LICS'16)*, pages 652–661, 2016. doi:10.1145/2933575.2933587.

24 Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, January 1982. doi:10.1145/322290.322292.

25 Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proceedings of AAAI-91*, pages 227–233, 1991. URL: http://www.aaai.org/Library/AAAI/1991/aaai91-036.php.

26 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007. doi:10.1145/1206035.1206036.

27 Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, 2010. doi:10.1137/090775646.

28 Marcin Kozik. Weak consistency notions for all the CSPs of bounded width. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*, pages 633–641. ACM, 2016. doi:10.1145/2933575.2934510.

29 Norman Lim, Shikharesh Majumdar, and Peter Ashwood-Smith. A constraint programming-based resource management technique for processing MapReduce jobs with SLSs on clouds. In *Proceedings of the 43rd International Conference on Parallel Processing (ICPP'14)*, pages 411–421. IEEE, 2014. doi:10.1109/ICPP.2014.50.

30 Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–233, 1986. doi:10.1016/0004-3702(86)90083-4.

31 Cemalettin Ozturk and M. Arslan Ornek. Optimisation and constraint based heuristic methods for advanced planning and scheduling systems. *International Journal of Industrial Engineering: Theory, Applications and Practice*, 23(1):26–48, 2016. URL: http://journals.sfu.ca/ijietap/index.php/ijie/article/view/1930.

32 Pinar Senkul and Ismail H. Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, 30(5):399–422, 2005. doi:10.1016/j.is.2004.03.003.

33 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 331–342, 2017. doi:10.1109/FOCS.2017.38.