Weak consistency notions for all the CSPs of bounded width [*]

Marcin Kozik

Theoretical Computer Science Department
The Faculty of Mathematics and Computer Science
Jagiellonian University
marcin.kozik@uj.edu.pl

Abstract

The characterization of all the Constraint Satisfaction Problems of bounded width, proposed by Feder and Vardi [SICOMP'98], was confirmed in [Bulatov'09] and independently in [FOCS'09, JACM'14]. Both proofs are based on the (2,3)-consistency (using Prague consistency in [FOCS'09], directly in [Bulatov'09]) which is costly to verify.

We introduce a new consistency notion, Singleton Linear Arc Consistency (SLAC), and show that it solves the same family of problems. SLAC is weaker than Singleton Arc Consistency (SAC) and thus the result answers the question from [JLC'13] by showing that SAC solves all the problems of bounded width. At the same time the problem of verifying weaker consistency (even SAC) offers significant computational advantages over the problem of verifying (2,3)-consistency which improves the algorithms solving the CSPs of bounded width.

Categories and Subject Descriptors F.4.1 [*Mathematical logic and formal languages*]: Mathematical Logic—Logic and constraint programming

Keywords Constraint Satisfaction Problem, DATALOG, local consistency checking algorithm

1. Introduction

An instance of the Constraint Satisfaction Problem consists of variables and constraints. In the decision version of CSP the question is whether the variables can be evaluated in such a way that all the constraints, often described as a relation constraining a sequence of variables, are satisfied.

In a seminal paper (Feder and Vardi 1998) Feder and Vardi proposed to parametrize the problem by restricting the constraining relations allowed in instances. More formally, for every finite relational structure **A** (called in this context *a template* or *a language*) the CSP parametrized by **A**, CSP(**A**), is the CSP restricted to instances with all the constraint relations taken from **A**.

Clearly, for any **A**, the problem CSP(**A**) is in NP and it is quite easy to construct relational structures which define NP-complete CSPs or CSPs solvable in polynomial time. One of the main problems in the area is *the CSP Dichotomy Conjecture* (Feder and Vardi 1998), which postulates that for every **A** the problem CSP(**A**) is NP-complete or solvable in a polynomial time. The CSP Dichotomy Conjecture remains open.

The class of problems which can be expressed as a CSP(**A**) is very rich; it is easy to construct relational structure **A** such that CSP(**A**) is 2-colorability of graphs, 3-SAT, 3-Horn-SAT, or a problem of solving systems of linear equations in $\mathbb{Z}_2$. The last problem on this list is a canonical example of a CSP with an *ability to count*. Feder and Vardi conjectured that all the CSPs which do not have the ability to count are solvable by local consistency checking. Local consistency checking algorithms operate by constructing a family of local solutions of an instance and enforcing some form of consistency on it. The CSPs solvable by such algorithms are *CSPs of bounded width*[1], and the conjecture of Feder and Vardi is the *the bounded width conjecture*.[2]

A breakthrough in the research on the parametrized CSPs appeared with an introduction of the *algebraic approach* (Bulatov et al. 2000, 2005). This approach is based on the Galois correspondence between relational structures and algebras (Bodnarčuk et al. 1969). At it's heart lies a method of associating algebras to templates in such a way that the computational properties of CSP (parametrized by the template) correspond to high-level algebraic properties of the algebra.

An algebraic approach allowed to restate and formalize (Larose and Zádori 2007) the bounded width conjecture. The restated conjecture postulated that a template has bounded with if and only if the associated algebra generates a variety which is congruence meet semi-distributive. This conjecture, and consequently the bounded width conjecture, was confirmed by two independent algebraic proofs (Barto and Kozik 2009, 2014; Bulatov 2009).

CSPs of bounded width form a big class of problems and appear naturally in many areas. To name one such connection: Guruswami and Zhou conjectured (Guruswami and Zhou 2011) that the class of CSPs which admit *robust*

[*] Research partially supported by National Science Center under grant DEC-2011/01/B/ST6/01006.

[1] In this paper we use phrases "CSPs solvable by local consistency checking" and "CSPs of bounded width" interchangeably.

[2] Not to be confused with the dichotomy conjecture of Feder and Vardi.

LICS '16, July 05-08, 2016, New York, NY, USA
Copyright © 2016 ACM 978-1-4503-4391-6/16/07. . . $15.00
DOI: http://dx.doi.org/10.1145/2933575.2934510

approximating algorithm (i.e. a poly-time algorithm that, if $1-\varepsilon$ fraction of constraints can be satisfied in an instance, provides a "solution" satisfying $1-g(\varepsilon)$ fraction of constraints and $g(\varepsilon) \to 0$ as $\varepsilon \to 0$) coincides with the class of CSPs of bounded width. This conjecture was confirmed in (Dalmau and Krokhin 2013; Barto and Kozik 2012).

The study of consistency notions for CSP predates the parametrized approach of Feder and Vardi. In many practical applications a consistency checking algorithm is used to quickly rule out some instances with no solutions with less regard for the existence of solution if the instance cannot be ruled out. Considered instances are usually large and very often sparse and thus require algorithms which work fast and do not disturb the structure of the instance. For an overview of consistency notions and algorithms we refer the reader to e.g. (Debruyne and Bessiere 1997; Bodirsky and Chen 2010; Chen et al. 2013).

The proof (Barto and Kozik 2009, 2014) of the bounded width conjecture required (k, l)-consistency (compare Definition 3.1 in (Barto 2014)) with k, l dependent on the maximal arity of relations in the template. The main theorem of (Barto 2014) shows that $(2, 3)$ minimality suffices for all the templates. On the other hand both proofs rely on a technical notion of Prague instance, which can be viewed as another consistency notion. The proof of (Barto and Kozik 2012) required another consistency definition: a definition of a weak Prague instance for templates with binary constraints (Definition 3.3 in (Barto and Kozik 2012)). All these results, as well as (Bulatov 2009), even in restriction to templates with binary constraints require some version of $(2, 3)$-consistency which is often discredited in practical applications.

Singleton arc consistency is, on the other hand, a well established consistency notion easier to verify than $(2, 3)$-consistency. Moreover an algorithm can verify SAC without distorting the structure of the instance (unlike the algorithms verifying $(2, 3)$-consistency). In (Chen et al. 2013) authors discuss applicability of SAC to CSPs of bounded width and ask if every CSP of bounded width is solvable by SAC.

We answer this question in positive by introducing a new consistency notion, Singleton Linear Arc Consistency, which is weaker than SAC and showing that it solves all the CSPs solvable by local consistency checking. The notion is not weaker than the one used in (Barto and Kozik 2012) to establish the conjecture of Guruswami and Zhou, but, at the cost of complicating proofs, it could be adjusted to imply that result as well. The definition of SLAC is motivated by the work in (Dalmau and Krokhin 2008; Bodirsky and Chen 2010) and provides a new insight into bounded width templates which can be important when attempting to construct algorithms solving wider classes of CSPs (with the dichotomy conjecture as a long term goal); or attempting a fine-grained classification of CSPs along the lines of e.g. (Dalmau and Krokhin 2008; Barto et al. 2012).

The paper is organized as follows. In the next section we describe a number of consistency notions and define the Singleton Linear Arc Consistency. In section 3 we state the main theorem of the paper and provide a rough sketch of the proof. Section 4 contains definitions of algebraic notions and collects established tools in the algebraic approach to CSP. In section 5 we rephrase the consistency notions from section 3 into a language which allows us to work with the instances of CSP. In section 6 we return to the main theorem of the paper and present a more in-depth sketch of the proof. Finally in section 7 we give a taste of the real proof by deriving a result which is responsible for one of the two cases in the proof of the main theorem. The last section contains acknowledgments.

2. Local consistency notions for CSPs

The purpose of local consistency checking is to eliminate, at a low computational cost, instances with no solutions. However if an algorithm checking a local consistency notion stops without deriving a contradiction the instance does not need to have a solution. In the parametrized approach to CSP the template has bounded width if some local consistency notion derives a contradiction for every non-solvable instance.

Formally *a template* of a CSP is a fixed, finite relational structure usually denoted by $\mathbf{A}$, and *an instance of CSP over template* $\mathbf{A}$ consists of a set of variables and a set of constraints which are pairs: $((x_1, \dots, x_n), R)$ where x_i is a variable and R is an n-ary relation in $\mathbf{A}$. *A solution of an instance* is a function f sending variables to the universe of $\mathbf{A}$, usually denoted by A, in such a way that for every constraint $((x_1, \dots, x_n), R)$ we have $(f(x_1), \dots, f(x_n)) \in R$.

There exists an extensive literature on various local consistency notions used in CSP (Debruyne and Bessiere 1997; Bodirsky and Chen 2010; Chen et al. 2013) etc. In the remaining part of this section we present only these which are directly pertinent to our approach. This is done using the notion of a DATALOG program.

2.1 DATALOG programs

A DATALOG program derives new facts about a relational structure using a set of rules. It operates in the languages of this structure (the relations/predicates[3] of that language are called *extensional database* or EDBs) enhanced by a number of auxiliary predicates/relations (called *intensional database* or IDBs).

A DATALOG rule has a *head* which is a single IDB on an appropriate number of variables and the *body* which is a sequence on IDBs and EDBs. During the execution of the program the head IDB is updated whenever the body of the rule is satisfied. The computation ends when no relation can be updated, or when the goal predicate is reached.

Example. The following DATALOG program operates on digraphs. The edge relation of a digraph (denoted by E) is the single binary EDB in the program. The program uses two IDBs: ODD and $GOAL$ (where $GOAL$ is the goal predicate) and verifies whether the digraph has a directed cycle of odd length.

$$ODD(x, y) :- E(x, y)$$
$$ODD(x, v) :- E(x, y) \wedge E(y, z) \wedge ODD(z, v)$$
$$GOAL :- ODD(x, x)$$

The program computes the relation ODD and fires the $GOAL$ predicate when a directed circle of odd length is found.

2.2 Arc consistency (1-consistency)

Arc consistency is one of the most basic local consistency notions. For the purpose of this paper we present it in two ways: using a DATALOG program and defining an iterative procedure. The notion is sometimes (Dalmau and Pearson 1999) called *generalized arc consistency* as it is adjusted to work with relations of all arities.

A DATALOG program verifying arc consistency for a template $\mathbf{A}$ has an IDB for each subset of the universe of

[3] We abuse the notation by identifying predicates with the relations in the structure.

A. Whenever $B(x)$ (for a variable x and $B \subseteq A$) is derived we understand that the program established that in every potential solution the variable x is evaluated into B.

The derivation itself proceeds in steps: in each step we focus a single constraint $((x_1, \dots, x_n), R)$ and use already established facts (i.e. IDBs) about the variables

$$A_1'(x_1), A_1''(x_1), \dots, A_2'(x_2), \dots$$

to derive that x_i needs to be evaluated in the appropriate projection of[4]

$$R \cap \Big(\bigcap_j A_1^{(j)} \times \dots \times \bigcap_j A_n^{(j)}\Big). \qquad (\ddagger)$$

A DATALOG rule realizing this derivation is

$$\begin{aligned} B(x_i) :- R(&x_1, \dots, x_n) \wedge \\ &A_1'(x_1) \wedge A_1''(x_1) \wedge \dots \wedge \\ &A_2'(x_2) \wedge A_2''(x_2) \wedge \dots \wedge \\ &\dots \\ &A_n'(x_n) \wedge A_n''(x_n) \wedge \dots \end{aligned}$$

where B is a projection of ($\ddagger$) to the i–th coordinate. To construct a DATALOG program verifying arc consistency for a fixed template **A** we collect all such rules, and set the goal predicate to be the predicate associated with an empty set. Whenever this goal predicate is reached we will say that the program *derived a contradiction*.

Alternatively arc consistency can be described be a procedure updating, for every variable x, a special unary constraint C_x. The idea behind these constraints is to store information derived about the variable. More formally:

```
for every variable $x$ in $\mathcal{I}$ do
    add to $\mathcal{I}$ a new special constraint $C_x := (x, A)$
end for
repeat
    for every variable $x$ in $\mathcal{I}$ do
        $B := A$
        for every value $a \in A$ do
            for each constraint $((x_1, \dots, x_n), R)$ with $x_i = x$ do
                let $R' \subseteq R$ contain all tuples respecting $C_{x_j}$'s
                if no $(a_1, \dots, a_n)$ in $R'$ has $a_i = a$ then
                    remove $a$ from $B$
                end if
            end for
        end for
        $C_x := (x, B)$
    end for
until none of the $C_x$ changed
```

For a constraint $((x_1, \dots, x_n), R)$ a tuple $(a_1, \dots, a_n) \in R$ respects $C_{x_j} = (x_j, A_{x_j})$ if $a_j \in A_{x_j}$. The algorithm *derives a contradiction* when one of the special constraints is of the form $C_x = (x, \emptyset)$ i.e. has an empty constraint relation.

Note that if the instance has a solution sending variable x to $a \in A$ and the Datalog program derives $B(x)$ then we have $a \in B$, and in the procedural version if $C_x = (x, B)$ at some step, then $a \in B$. Therefore deriving an empty predicate on a variable (or a special constraint with an empty set) implies that the variable has no possible values and that the instance has no solution.

[4] Note that a variable need not appear in any of the IDBs, to address this purely technical problem we set the intersection of the empty family of sets to be A.

Example. Let the template **A** have universe $A = \{0, 1\}$ and a single binary relation $\neq$. To avoid confusion we denote the IDB associated with $\{0\}$ subset (resp. $\{1\}$ subset) by $\cdot = 0$ (resp. $\cdot = 1$). The following DATALOG program verifies 1-consistency for **A**:

$$\begin{aligned} \emptyset(x) &:- x = 0 \wedge x = 1, \\ x = 0 &:- x \neq y \wedge y = 1, \\ y = 0 &:- x \neq y \wedge x = 1, \\ x = 1 &:- x \neq y \wedge y = 0, \\ y = 1 &:- x \neq y \wedge x = 0. \end{aligned}$$

Note that all the rules are necessary as a Datalog program disregards commutativity of $\neq$.

On the instance with variables x, y, z and constraints $x \neq y, y \neq z, z \neq x$, this program derives no contradiction despite the fact that the instance has no solution.

This example shows that the template $(\{0, 1\}, \neq)$ is not solvable by arc consistency. As it is solvable by a different local consistency notion we conclude that the arc consistency is not strong enough to verify all the CSPs of bounded width. In fact all the CSPs solvable by arc consistency are characterized in (Dalmau and Pearson 1999).

Note that, for a fixed template, arc-consistency can be computed quite quickly i.e. in the time linear with respect to the number of constrains.

2.3 Path consistency ($(2, 3)$-consistency)

The results establishing (Bulatov 2009; Barto and Kozik 2009, 2014) the bounded width conjecture of Feder and Vardi (Feder and Vardi 1998) were using higher consistency notions. The paper of Barto (Barto 2014) improves these results using slightly different concepts. We will not define these notions since they are not directly connected to this paper. We just mention that the result (Barto 2014) showed that if the template has bounded width then every $(2, 3)$-minimal instance has a solution.

When restricted to templates with binary relations all the above results (Bulatov 2009; Barto and Kozik 2009, 2014; Barto 2014) require some form of $(2, 3)$-consistency also known as path-consistency. A DATALOG program verifying $(2, 3)$-consistency, for a binary template, is constructed in a way similar to the program for arc consistency, except for two differences:

- it introduces a binary IDB for each subset of A^2 (instead of unary ones for subsets of A), and
- it allows derivation rules with body on three variables (arc-consistency on binary relations would have at most two variables in the body of every DATALOG rule).

Even when restricted to binary constraints SLAC, or SAC, is strictly weaker than $(2, 3)$-consistency. For constraints of arbitrary arities both notions are strictly weaker than $(2, 3)$ minimality.

Verifying $(2, 3)$-consistency is inefficient for many reasons: the computational complexity is cubic with respect to the number of variables (Debruyne and Bessiere 1997). More importantly the instance of the constraint satisfaction program is loosing its structure. In many practical applications the CSP instances are sparse — establishing arc consistency do not change the structure of an instance, however establishing $(2, 3)$-consistency on any (even sparse) instance would essentially add a constraint for every pair of variables loosing a significant computational advantage.

The proofs in (Barto and Kozik 2009, 2014; Barto 2014) were using yet another, technical, consistency notion: Prague consistency. This notion, although interesting theoretically, is not easy to verify and did not lead to algorithms more efficient then those verifying $(2, 3)$-consistency.

2.4 Singleton Arc Consistency (SAC)

Singleton arc-consistency is a notion stronger than arc consistency, but weaker than path consistency. That means that an instance which is path consistent is necessarily SAC, and that every SAC instance is arc consistent. The reverse implications do not hold.

Let us fix an instance $\mathcal{I}$. SAC is defined using a procedure similar to the one defining AC. This time the constraints C_x are updated by running arc consistency with *the value of x fixed to an arbitrary a*. Formally:

```
for every variable $x$ in $\mathcal{I}$ do
    add to $\mathcal{I}$ a new special constraint $C_x := (x, A)$
end for
repeat
    for every variable $x$ in $\mathcal{I}$ do
        $B := A$
        for every value $a \in A$ do
            run AC on $\mathcal{I}$ restricted by $C_y$'s and $(x, \{a\})$
            if AC derived a contradiction then
                remove $a$ from $B$
            end if
        end for
        $C_x := (x, B)$
    end for
until none of the $C_x$ changed
```

Restricting $\mathcal{I}$ by C_y's means substituting, in every constraint $((x_1, \ldots, x_n), R)$, the relation R with $R \cap \prod_{i=1}^{n} A_{x_i}$ where the A_{x_i}'s are taken from special constraints i.e. $C_{x_i} = (x_i, A_{x_i})$. The additional constraint $(x, \{a\})$ fixes value of x to a. The algorithm derives a contradiction in the same way the algorithm for AC did.

The algorithm above, while conceptually simple, does not provide the best possible running time. Using the fact that the arc consistency, in the inner most loop, is computed multiple times on more or less the same instance it is possible to verify SAC in time bounded by a constant times the number of constraints times the number of variables (Bessiere and Debruyne 2008). More importantly SAC does not alter the structure of the instance and therefore runs efficiently on sparse instances with many variables.

SAC has been studied (Chen et al. 2013) but there was no characterization of the set of templates solvable by SAC. As Singleton Linear Arc Consistency is weaker then SAC the main theorem of this paper shows that SAC solves all the CSPs of bounded width. SLAC offers some computational advantages over SAC, however it is not clear if it allows to construct an algorithm with better worst-case time complexity than the algorithm mentioned above.

2.5 Linear Arc Consistency (LAC)

This consistency notion is a weaker version of arc consistency and originates in (Bodirsky and Chen 2010; Dalmau and Krokhin 2008).

In order to define the notion we take the DATALOG program for arc consistency from section 2.2 and remove from it all the rules with more than one IDB in the body. Linear Arc Consistency is the consistency verified by this program. As the program has fewer derivation rules than the original one it computes less information about the instance.

On the other hand LAC can be verified in NL (as it essentially reduces to reachability for directed graphs), while AC solves some of the P-complete CSPs (e.g. Horn-SAT) and we cannot expect to put it into a low complexity class.

2.6 Singleton Linear Arc Consistency (SLAC)

This is the main consistency notion of this paper. It is stronger than AC, but weaker than SAC. In Theorem 3.3 we show that SLAC solves all the constraint satisfaction problems of bounded width.

The Singleton Linear Arc Consistency is defined by an algorithm almost identical to the algorithm for SAC in section 2.4; then only difference lies in the inner most loop where (for SLAC) we evaluate LAC instead of AC. Note that it is important (unlike in the case of SAC) that the LAC is evaluated in a restricted instance. Still the cost of excluding a single candidate for a variable is (unlike for SAC) in NL.

3. Parametrized CSP and the main theorem

The following corollary, of a more technical Theorem 3.3, is the easiest way to state the main result of the paper.

Corollary 3.1. SLAC derives a contradiction *on every* unsolvable instance over a template of bounded width.

As SLAC cannot derive a contradiction on a solvable instances we immediately get that SLAC solves the CSP for every template of bounded width.

However, in order to prove Theorem 3.3 and thus the corollary, we use algebraic characterizations which require a number of standard reductions. First reduction, already present in (Feder and Vardi 1998), allows us to add to $\mathbf{A}$ relations pp-definable in $\mathbf{A}$. A relation is *pp-definable* in $\mathbf{A}$ if it can be defined using relations in $\mathbf{A}$, conjunction and existential quantifiers. Adding a pp-definable relation to a template does not increase the computation complexity of the corresponding CSP. More importantly other properties, like being solvable by local consistency checking, are preserved under this construction. Throughout the paper we assume that all the pp-definable unary relations are already present in $\mathbf{A}$.

Further standard reductions allow us to focus on $\mathbf{A}$ which are *cores* that is relational structures such that every endomorphism of $\mathbf{A}$ (i.e. a homomorphism from $\mathbf{A}$ to $\mathbf{A}$) is a bijection. The final standard reduction allows us consider only *rigid cores* i.e. relational structures for which identity is the only endomorphism (Feder and Vardi 1998; Bulatov et al. 2000, 2005) .

For technical reasons the instances we consider throughout the paper have, for every variable x, a constraint of the form (x, A_x) where A_x is a subset of A pp-definable in $\mathbf{A}$. For historical reasons the sets A_x are sometimes called potatoes. Note that every instance can be turned into such an instance by adding dummy constraints of the form (x, A).

3.1 1-consistent instances

Definition 3.2. *An instance is* 1*-consistent if, for every constraint $((x_1, \ldots, x_n), R)$ in the instance, the projection of R to the i-th coordinate is equal to A_{x_i}.*

Note that the Datalog program for arc consistency run on a 1-consistent instance would immediately derive $A_x(x)$ for every variable x and would not derive any further information. Moreover, for any instance $\mathcal{I}$, we can use AC to construct a 1-consistent instance. If AC stops without deriving a contradiction then, for every variable x of $\mathcal{I}$ we have a special constraint of the form $C_x = (x, A_x)$ for some non-empty A_x.

After restricting the instance by C_x's (equivalently to A_x's), as in section 2.4, we obtain a new instance $\mathcal{I}'$. The instance $\mathcal{I}'$ has exactly the same set of solutions as $\mathcal{I}$ and is 1-consistent (since the derivation of AC stopped). Moreover the new relations appearing in the constraints are pp-definable in the relational structure. We sometimes call such $\mathcal{I}'$ *a* 1-*consistent subinstance of* $\mathcal{I}$.

3.2 SLAC instances and the main theorem

Comparing the procedural version of AC and SLAC we immediately obtain that the SLAC verifies (among other things) arc consistency of the instance. If the SLAC algorithm stops, on an instance $\mathcal{I}$, with special constraints of the form $C_x = (x, A_x)$ we can restrict $\mathcal{I}$ by C_x's – a result of such a restriction is a *SLAC instance* .

This instance has the same set of solutions as the original instance, it is 1-consistent and SLAC run on such an instance would immediately derive $C_x = (x, A_x)$ and would not derive any further information (this property is sometimes used as an equivalent definition of a SLAC instance).

The main theorem of this paper claims solutions for SLAC instances:

Theorem 3.3. *Let* $\mathbf{A}$ *be a template which is a rigid core. If* CSP($\mathbf{A}$) *has bounded width, then every SLAC instance in* CSP($\mathbf{A}$) *has a solution.*

Proof of Corollary 3.1. Let $\mathcal{I}$ be an instance over $\mathbf{A}$ (of bounded width) such that the SLAC algorithm does not derive a contradiction on $\mathcal{I}$.

Restricting by C_x's, derived by SLAC, we obtain a SLAC instance over $\mathbf{A}$. Let $\mathbf{A}'$ be a core of $\mathbf{A}$ via a homomorphism h. Applying h to $\mathcal{I}'$ we obtain a SLAC instance over $\mathbf{A}'$, which is also an instance over a rigid core of $\mathbf{A}$.

Since $\mathbf{A}$ had bounded width so does the rigid core of $\mathbf{A}$ and we can use Theorem 3.3 to establish a solution to the image of $\mathcal{I}'$ under h. Since h is an endomorphism of $\mathbf{A}$ this solution works for $\mathcal{I}$ as well. ◻

We present a short sketch of the proof of Theorem 3.3; this sketch is extended in section 6.

The general idea of the proof is to start with an arbitrary SLAC instance $\mathcal{I}$ and show that, choosing appropriate pp-definable $A'_x \subseteq A_x$, the restriction of $\mathcal{I}$ to A'_x is a proper SLAC subinstance of $\mathcal{I}$. If this can be accomplished then, in a finite number of steps, we arrive at a SLAC instance such that every A_x has one element, and such an instance clearly defines a solution.

Finding proper A'_x for an instance splits into two cases depending on the algebraic structure of the instance. Either we can find A'_x which *absorbs* (compare section 4.1.1) A_x, or the lack of absorption implies enough "rectangularity" of the constraints that A'_x can be chosen more arbitrarily.

The next section contains definitions of algebraic concepts necessary to make these statements precise.

4. Algebraic notions and tools

First we introduce general algebraic notions which allow us to define the Galois correspondence.

4.1 Basic algebraic notions

An algebra, usually denoted by $\mathbb{A}$, is a set A (the *universe* of the algebra) together with a list of functions of arbitrary (but finite) arity from A to A (the *operations* of the algebra). *A signature* of an algebra is a list of arities of its operations.

Let $\mathbb{A}$ be an algebra, a set $B \subseteq A$ is *a subuniverse* of $\mathbb{A}$ if it is closed under every operation of $\mathbb{A}$. In such a case the algebra with universe B and operations obtained from the operations of $\mathbb{A}$ by restricting to arguments from B is a *subalgebra* of $\mathbb{A}$ denoted $\mathbb{B} \leq \mathbb{A}$.

Let $\mathbb{A}_i$ be a list (or, more generally, an indexed family) of algebras in the same signature, the algebra $\prod_i \mathbb{A}_i$ is the algebra in the same signature; its universe is a cartesian product of universes of $\mathbb{A}_i$'s and the operations are evaluated coordinatewise. If $\mathbb{A}$ is an algebra then $\mathbb{A}^n$ is the n-th cartesian power of $\mathbb{A}$ i.e. a product of n copies of $\mathbb{A}$.

An equivalence relation α on the universe of an algebra $\mathbb{A}$ is *a congruence* if it is a subalgebra of $\mathbb{A}^2$. In such a case the algebra $\mathbb{A}/\alpha$ is well defined i.e. the operations in the quotient do not depend on the choice of representatives of the equivalence class. For any $a \in \mathbb{A}$ by a/α we mean the equivalence block of α containing a.

An algebra $\mathbb{A}$ is *simple* if its only congruences are $0_{\mathbb{A}} = \{(a, a) : a \in \mathbb{A}\}$ and $1_{\mathbb{A}} = \{(a, b) : a, b \in \mathbb{A}\}$. If α and β are congruences on $\mathbb{B}$ then the largest equivalence contained in both is a congruence denoted by $\alpha \wedge \beta$ and the smallest congruence containing α and β is denoted by $\alpha \vee \beta$. In general, for any binary relation on an algebra, the congruence *generated* by this relation is the unique smallest congruence containing this relation.

If $\mathbb{A} \leq \prod_{i \in I} \mathbb{A}_i$ then π_i is the congruence identifying tuples of $\mathbb{A}$ with the same element on the i-th coordinate; and for any $J \subseteq I$ the algebra $\mathrm{proj}_J(\mathbb{A})$ is obtained from $\mathbb{A}$ by taking tuples in $\mathbb{A}$ and projecting them to the coordinates from J. If $\mathbb{A} \leq \prod_i \mathbb{A}_i$ and for every i $\mathrm{proj}_i(\mathbb{A}) = \mathbb{A}_i$ then $\mathbb{A}$ is *subdirect* and we denote this fact by $\mathbb{A} \leq_{\mathrm{sub}} \prod_i \mathbb{A}_i$.

If $\mathbb{C} \leq \mathbb{A}^2$ then by $\mathbb{C}^{(n)}$ we denote the subalgebra of $\mathbb{A}^2$ which is obtained by composing $\mathbb{C}$ with itself n-times. If $\mathbb{C} \leq \mathbb{A} \times \mathbb{B}$ and $\mathbb{A}' \leq \mathbb{A}$ then $\mathbb{A}' + \mathbb{C}$ is a subalgebra of $\mathbb{B}$ containing b iff $\exists\, a \in \mathbb{A}' : (a, b) \in \mathbb{C}$. For $\mathbb{B}' \leq \mathbb{B}$ we define $\mathbb{B}' - \mathbb{C}$ analogously.

A homomorphism from $\mathbb{A}$ to $\mathbb{B}$ (where $\mathbb{A}$ and $\mathbb{B}$ have the same signature) is a map from A to B which commutes with operations of $\mathbb{A}$ and $\mathbb{B}$. A bijective homomorphism is *an isomorphism*.

A term in a given signature is a formal description of a composition of operations of an algebra in this signature. For every algebra in this signature a term defines a *term operation*.

A class of algebras in the same signature closed under taking products, homomorphic images and subalgebras is called *a variety*. The smallest variety containing an algebra $\mathbb{A}$ is the variety *generated* by $\mathbb{A}$.

An algebra $\mathbb{A}$ is idempotent if every one-element subset of its universe is a subalgebra. In such an algebra a congruence block is a subalgebra; moreover every algebra in a variety generated by an idempotent algebra is idempotent.

4.1.1 Absorption

The notion of *absorption* and *absorbing subalgebras* appeared first in (Barto and Kozik 2009) and plays a crucial role in many recent developments in the algebraic approach to CSP.

Let $\mathbb{B} \leq \mathbb{A}$ be idempotent algebras, we say that $\mathbb{B}$ *absorbs* $\mathbb{A}$ (denoted $\mathbb{B} \trianglelefteq \mathbb{A}$) if there exists a term operation t in $\mathbb{A}$ such that

$$t(a_1, \ldots, a_n) \in \mathbb{B} \text{ whenever } |\{i : a_i \notin \mathbb{B}\}| \leq 1.$$

The following easy consequences of the definition will be useful:

- if $\mathbb{B} \trianglelefteq \mathbb{A}$ and $\mathbb{C} \leq \mathbb{A}$ then $(\mathbb{B} \cap \mathbb{C}) \trianglelefteq \mathbb{C}$;
- if $\mathbb{C} \leq_{\text{sub}} \mathbb{A} \times \mathbb{B}$ and $\mathbb{A}' \trianglelefteq \mathbb{A}$ then $\mathbb{A}' + \mathbb{C} \trianglelefteq \mathbb{B}$;
- if $\mathbb{B} \trianglelefteq \mathbb{A}, \mathbb{B}' \trianglelefteq \mathbb{A}'$ are algebras in the same signature, then both absorptions can be witnessed by a common term.

4.2 The Galois correspondence and basic reductions

At the heart of the algebraic approach to CSP lies a correspondence (Bodnarčuk et al. 1969; Bulatov et al. 2000, 2005) between relational structures and algebras.

To each template $\mathbf{A}$ we associate an algebra $\mathbb{A}$: a k-ary function f is an operation of $\mathbb{A}$ if and only if every relation in $\mathbf{A}$ is a subuniverse of appropriate power of the algebra (A, f). Such an operation is called a *polymorphism* of $\mathbf{A}$.

For such an $\mathbb{A}$ all the finite subpowers of $\mathbb{A}$ are exactly the relations pp-definable in $\mathbf{A}$ which indicates why $\mathbb{A}$ captures the complexity of CSP defined by $\mathbf{A}$. In the remaining part of the paper, all the relations appearing in constraints are, at the same time, subalgebras of powers of $\mathbb{A}$. Note that, for any instance, the set of solutions of this instance also forms a subalgebra of a power of $\mathbb{A}$.

By standard reductions listed in section 3 we restrict our reasoning to templates which are rigid cores. The algebras associated to such templates by the Galois correspondence are idempotent.

By results of (Barto and Kozik 2009, 2014; Bulatov 2009) a rigid core template $\mathbf{A}$ defines a CSP solvable by local consistency checking if and only if the associated algebra $\mathbb{A}$ generates a variety such that for any algebra $\mathbb{B}$ in this variety and any α, β, γ congruences of $\mathbb{B}$ if $\alpha \wedge \beta = \alpha \wedge \gamma$ then $\alpha \wedge \beta = \alpha \wedge (\beta \vee \gamma)$. We call such an algebra $\mathbb{A}$ an SD($\wedge$) algebra, and we call a rigid core template an SD($\wedge$) template if the algebra associated to it is SD($\wedge$).

A notion of a Taylor algebra (which appears in Theorem 4.2) is not required in this paper, all that we need to know is that every SD($\wedge$) algebra is Taylor.

4.3 Algebraic tools

In this section we cite some established algebraic tools which were developed with the algebraic approach to CSP in mind.

From this point on we assume that all the algebras are idempotent and all the templates are rigid cores.

Fact 4.1. Let $\mathbb{C} \leq_{\text{sub}} \mathbb{A} \times \mathbb{B}$:

- if α is a transitive closure of the relation on $\mathbb{A}$ containing all pairs
$$(a, a') : \exists\, b \in \mathbb{B}\, (a, b), (a', b) \in \mathbb{C}$$
then α is a congruence on $\mathbb{A}$;
- similarly for β defined dually on $\mathbb{B}$;
- if $\mathbb{C}' \leq_{\text{sub}} \mathbb{A} \times \mathbb{B}$ and $\mathbb{C}' \trianglelefteq \mathbb{C} \leq_{\text{sub}} \mathbb{A} \times \mathbb{B}$ then α's and β's defined by $\mathbb{C}$ and $\mathbb{C}'$ coincide.[5]

In fact, we call $\mathbb{C} \leq_{\text{sub}} \mathbb{A} \times \mathbb{B}$ *linked* if $\pi_1 \vee \pi_2 = 1_{\mathbb{C}}$ or, equivalently α (or β) defined for $\mathbb{C}$ in Fact 4.1 are $1_{\mathbb{A}}$ ($1_{\mathbb{B}}$ respectively).

Note that if $\mathbb{C} \leq_{\text{sub}} \mathbb{A} \times \mathbb{B}$ is a graph of a bijection (equivalently α, β in Fact 4.1 are $0_{\mathbb{A}}, 0_{\mathbb{B}}$ respectively) then $\mathbb{C}$ is actually a graph of an isomorphism between $\mathbb{A}$ and $\mathbb{B}$.

The following theorem first appeared as Theorem III.6 (Barto and Kozik 2010)

Theorem 4.2. *Let $\mathbb{C} \leq_{sub} \mathbb{A} \times \mathbb{B}$ be a Taylor algebra. If $\mathbb{C}$ is linked then*

- $\mathbb{C} = \mathbb{A} \times \mathbb{B}$, *or*
- $\mathbb{A}$ *has a proper absorbing subalgebra or*
- $\mathbb{B}$ *has a proper absorbing subalgebra.*

And the following corollary easily follows from it:

Corollary 4.3. Under the assumptions of Theorem 4.2 there exist $\mathbb{A}' \trianglelefteq \mathbb{A}, \mathbb{B}' \trianglelefteq \mathbb{B}$ such that $\mathbb{A}' \times \mathbb{B}' \leq \mathbb{C}$.

The following proposition is, in essence, Theorem 6 of (Barto et al. 2012).

Proposition 4.4. Let $\mathbb{A}$ be an algebra and $\mathbb{R}, \mathbb{S} \leq_{\text{sub}} \mathbb{A}^n$ such that $\mathbb{R} \trianglelefteq \mathbb{S}$, and for every $a \in \mathbb{A}$ the constant tuple $(a, \ldots, a)$ belongs to $\mathbb{S}$. Then $\mathbb{R}$ contains at least one constant tuple.

5. Instances and patterns

In order to apply the algebraic tools to SLAC, or for that matter 1-consistent, instances we need a better way to handle such instances. In the following section we introduce patterns (which are essentially CSP instances with added information) and a natural notion of a homomorphism between instances in order to capture these consistency notions.

Throughout the paper we work with instances of CSP over a fixed, finite template. Every instance of CSP over such a template can be equivalently viewed as a relational structure in the language of the template. This was already the case when we considered a DATALOG program computing on an instance of a CSP.

Having two instances of the same CSP, say $\mathcal{I}$ and $\mathcal{J}$, and φ a function mapping variables of $\mathcal{I}$ to variables of $\mathcal{J}$ we say φ is a homomorphism if for every constraint $((x_1, \ldots, x_n), \mathbb{C})$ in $\mathcal{I}$ there is a constraint $((\varphi(x_1), \ldots, \varphi(x_n)), \mathbb{C})$ in $\mathcal{J}$. This is exactly the same as saying that φ is a homomorphism between relational structures which $\mathcal{I}$ and $\mathcal{J}$ essentially are.

5.1 Path patterns

We begin with a basic definition.

Definition 5.1. A step *is an instance of a CSP with a single constraint (in which every variable appears once) with two selected variables. We denote a step by $(i, ((x_1, \ldots, x_n), \mathbb{C}), j)$ where $((x_1, \ldots, x_n), \mathbb{C})$ is the constraint and the selected variables are x_i* the beginning *of the step and x_j* the end.

We define path-patterns next.

Definition 5.2. A path-pattern *is an instance of CSP constructed from a sequence of steps (on pairwise disjoint sets of variables) by identifying each step's end variable with next step's beginning variable. The beginning variable of the path-pattern is the beginning variable of the first step, and the end variable of the path-pattern is the end variable of the last step.*

A subpattern of a path-pattern *is a path-pattern defined by a substring of the sequence of steps.*

A path-pattern p is in the instance $\mathcal{I}$ *if there exists a homomorphism (usually denoted by φ) from the instance of p to $\mathcal{I}$.*

A path-pattern p is a cycle (at x) in $\mathcal{I}$ *if this homomorphism can be chosen so that the beginning and end variables are mapped to the same element (to x) by φ.*

Note that the path-patterns allow us to capture the notions of linear arc consistency and singleton linear arc consistency:

- LAC algorithm does not derive a contradiction on the instance $\mathcal{I}$ if and only if every path-pattern in $\mathcal{I}$ has a solution;[6]

[5] The property is proved by a standard absorption argument found in e.g. (Barto and Kozik 2009).

[6] By a solution of a pattern we mean, of course, a solution of the underlying instance.

- an instance $\mathcal{I}$ is a SLAC instance if and only if for every variable x, every $a \in A_x$ and any path pattern p which is a cycle at x in $\mathcal{I}$ p has a solution sending beginning and end variables of p to a.

Indeed, a LAC algorithm derivation defines a path pattern (and vice versa), and the path pattern has a solution if the derived set is non-empty. In the case of SLAC the algorithm does not shrink A_x if fixing x to $a \in A_x$ and running LAC does not produce an empty set. This is equivalent to the condition about path-patterns which are cycles at x.

Throughout the paper we often create patterns by merging other patterns (just like in the definition above). In such situations we assume that the variable sets of these patterns are disjoint and all the identifications of the variables are explicitly stated. E.g. if p and q are path-patterns on disjoint sets of variables we define $p+q$ as a path-pattern created by identifying the end variable of p with the beginning variable of q. The beginning variable of the sum is the beginning variable of p and the end variable of the sum is the end variable of q. A pattern $-p$ is obtained from pattern p by exchanging the functions of beginning and end variables.

5.2 Tree patterns

To define a tree-pattern we introduce a notion of an adjacency multigraph of an instance. The vertex set of the adjacency multigraph of an instance consists of all the variables and all the constraints of this instance, and we introduce one edge between a variable vertex and a constraint vertex for every time the variable appears in the constraint.

Definition 5.3. *An instance is* a tree (forest) instance *if it's adjacency multigraph is a tree (forest) i.e. has no multiple edges and no cycles.*

A tree-pattern *is a tree-instance with a selected variable called* the root, *and a selected set of variables of degree one in the adjacency multigraph called* leaves.

A tree-pattern p is in the instance $\mathcal{I}$ *if there exists a homomorphism (usually denoted by φ) from the instance of p to $\mathcal{I}$.*

Tree patterns are to arc consistency as path patterns to linear arc consistency i.e.

- AC checking algorithm does not derive a contradiction on an instance if and only if every tree pattern in this instance has a solution.

5.3 Propagation via patterns

For a path-pattern (tree-pattern) p and a set $B \subseteq A$ we put $B+p$ to be the set consisting of all the values given to the end of p (resp. root of p) by solutions of p sending the beginning (resp. leaf) variables into B.

Additionally, for path-patterns, by $B+p+q$ we mean $(B+p)+q$ (equivalently $B+(p+q)$) and by $B-p$ we mean $B+(-p)$.

5.4 Universal covering tree instances

In order to capture the property "AC does not derive a contradiction on $\mathcal{I}$" we construct an auxiliary instance UCT($\mathcal{I}$). We define it for connected instances first.

We say that an instance is connected if it's adjacency multigraph is connected. For a connected instance $\mathcal{I}$ the tree instances in $\mathcal{I}$ form a Fraïssé family and we define the instance UCT($\mathcal{I}$) be its limit and φ to be the natural homomorphism from UCT($\mathcal{I}$) to $\mathcal{I}$.

Equivalently we can define UCT($\mathcal{I}$) to be the smallest (usually countably infinite) instance such that:

- it is a tree instance,
- it maps homomorphically onto $\mathcal{I}$ via a map denoted by φ,
- for every $(\mathbf{x}, \mathbb{C})$ constraint of $\mathcal{I}$ there is $(\mathbf{v}, \mathbb{C})$ constraint of UCT($\mathcal{I}$) such that $\varphi(v_i) = x_i$ for every i,
- for every two variables u, v of UCT($\mathcal{I}$) if $\varphi(u) = \varphi(v)$ there is an automorphism ψ of UCT($\mathcal{I}$) sending u to v and such that $\varphi(\psi(w)) = \varphi(w)$ for every w.

Note that each tree-pattern in $\mathcal{I}$ maps homomorphically to UCT($\mathcal{I}$) and the following conditions are equivalent:

- every tree-pattern in $\mathcal{I}$ has a solution;
- UCT($\mathcal{I}$) has a solution;
- AC checking algorithm does not derive a contradiction on the instance $\mathcal{I}$.

If $\mathcal{I}$ is disconnected we take UCT($\mathcal{I}$) to be the union of UCT's of its connected components.

6. Main theorem and sketch of the proof

In this section we restate Theorem 3.3 and provide a more in-depth sketch of its proof using the algebraic notions from the previous sections.

Recall that all the templates are rigid cores, and that all the instances have associated special constraints of the form (x, A_x). In order to use the algebraic properties of the template we restate Theorem 3.3 as follows:

Theorem 6.1. *Every SLAC instance over an SD($\wedge$) template has a solution.*

We start with a SLAC instance $\mathcal{I}$ over an SD($\wedge$) template $\mathbf{A}$. As described in section 3.2 our goal is to find algebras $\mathbb{A}'_x \leq \mathbb{A}_x$ for every x and show that $\mathcal{I}$ restricted to $\mathbb{A}'_x$ is a SLAC instance.

6.1 There is no absorption in the instance

The first case we consider is when none of the algebras $\mathbb{A}_x$ has an absorbing subuniverse. In this case we choose an arbitrary x with a non-trivial $\mathbb{A}_x$ and choose a congruence α_x such that $\mathbb{A}_x/\alpha_x$ is simple. We pick an arbitrary block of this congruence to be $\mathbb{A}'_x$.

Take any path-pattern p in $\mathcal{I}$ such that φ maps beginning of p to x and let w be the end variable of p. All the solutions to p, projected on the beginning and the end of p, produce an algebra $\mathbb{R} \leq_{\text{sub}} \mathbb{A}_x \times \mathbb{A}_{\varphi(w)}$ and we can quote $\mathbb{R}$ on the first coordinate by α_x to obtain $\mathbb{R}'$. If $\mathbb{R}'$ is not-linked then it defines congruence $\alpha_{\varphi(w)}$ on $\mathbb{A}_{\varphi(w)}$ and an isomorphism between $\mathbb{A}_x/\alpha_x$ and $\mathbb{A}_{\varphi(w)}/\alpha_{\varphi(w)}$ which forces the equivalence class of $\alpha_{\varphi(w)}$ that needs to be chosen as $\mathbb{A}'_{\varphi(w)}$ to be $\mathbb{A}'_x + p$.

One needs to show that the congruence and the isomorphism is independent on the choice of p; and define $\mathbb{A}'_y = \mathbb{A}_y$ for y's such that every p defines a linked $\mathbb{R}'$ (note that in this case, by Theorem 4.2, $\mathbb{R}' = \mathbb{A}_x/\alpha_x \times \mathbb{A}_{\varphi(w)}$). It remains to show that the restriction of $\mathcal{I}$ to $\mathbb{A}'_x$ is 1-consistent and finally a SLAC instance.

6.2 There is absorption in the instance

In this case some algebra $\mathbb{A}_{x'}$ has a non-trivial absorbing subuniverse, say $\mathbb{A}'$. For every tree pattern p in $\mathcal{I}$ with all the leaf variables sent to x' the algebra $\mathbb{A}' + p$ is either empty or an absorbing subuniverse of $\mathbb{A}_y$ where y is the variable of $\mathcal{I}$ to which φ sends the root of p. Using appropriate tree patterns we are able to define $\mathbb{A}'_x \trianglelefteq \mathbb{A}_x$ for every x such that the restriction of $\mathcal{I}$ to $\mathbb{A}'_x$ is 1-consistent.

Next, using Theorem 7.1, taking for $\mathcal{I}$ instances witnessing big chunks of singleton linear arc consistency we construct even smaller $\mathbb{A}'_x$ such that the original instance restricted to these algebras is a SLAC instance.

7. Tools: absorbing subinstances

In this section we prove a theorem which plays a crucial role in the reduction in the case with absorption. It does not require the template to be SD($\wedge$), in fact it requires no algebraic structure apart from the explicitly stated absorption. In our opinion this theorem is of independent interest and should find applications not connected to this paper.

In essence the theorem states that if a consistency can be found everywhere in the instance, then it can be found in any 1-consistent absorbing subinstance.

Theorem 7.1. *Let $\mathcal{I}$ be an instance such that any $a \in \mathbb{A}_x$ extends to a solution of $\mathcal{I}$. Let, for every variable x, $\mathbb{A}'_x$ be an algebra such that*

1. $\mathbb{A}'_x \trianglelefteq \mathbb{A}_x$;
2. $\mathrm{UCT}(\mathcal{I})$ *can be solved with each variable v in $\mathbb{A}'_{\varphi(v)}$.*

Then there exists a solution to $\mathcal{I}$ with every variable x evaluated into $\mathbb{A}'_x$.

We proceed to prove Theorem 7.1. Let a solution to $\mathrm{UCT}(\mathcal{I})$ be called *prime* if it sends every variable v into $\mathbb{A}'_{\varphi(v)}$. For every variable x of $\mathcal{I}$ we choose an arbitrary v of $\mathrm{UCT}(\mathcal{I})$ such that $\varphi(v) = x$ and redefine $\mathbb{A}'_x$ to consist of all the values v can take in prime solutions. Note that:

- the choice of v (for a fixed x) does not matter (as $\mathrm{UCT}(\mathcal{I})$ has automorphisms moving potential candidates to each other) and
- the assumptions of the theorem still hold:
  1. Fix an x and v such that $\varphi(v) = x$. For every $a \in \mathbb{A}_x$ there is a solution e of $\mathcal{I}$ sending x to a, therefore $e \circ \varphi$ is a solution of $\mathrm{UCT}(\mathcal{I})$ sending v to a. The algebra of prime solutions of $\mathrm{UCT}(\mathcal{I})$ absorbs the algebra of all solutions of $\mathrm{UCT}(\mathcal{I})$ and therefore the new $\mathbb{A}'_x$ absorbs $\mathbb{A}_x$.
  2. Fix any prime solution of $\mathrm{UCT}(\mathcal{I})$; if a variable v is mapped to a then a belongs to the new $\mathbb{A}'_{\varphi(v)}$ which implies that the solution is, at every coordinate, in the new $\mathbb{A}'_x$'s.

After such a change the restriction of $\mathcal{I}$ to $\mathbb{A}'_x$ is 1-consistent.

Note that to prove the theorem it suffices to show that there exists a prime solution to $\mathrm{UCT}(\mathcal{I})$ constant on $\varphi^{-1}(x)$ for every variable x of $\mathcal{I}$. For induction assume that there is a prime solution to $\mathrm{UCT}(\mathcal{I})$ constant on each of the sets $\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_k)$ and let x be another variable.

Let $\mathbb{E} \leq \prod_{v \in \mathrm{UCT}(\mathcal{I})} \mathbb{A}_{\varphi(v)}$ be the algebra of all the solutions to $\mathrm{UCT}(\mathcal{I})$; for every $x, a \in \mathbb{A}_x$ there is a solution e such that $e(x) = a$; then $e \circ \varphi$ sends $\varphi^{-1}(x)$ to a and we proved that $\mathbb{E}$ is subdirect in $\prod_{v \in \mathrm{UCT}(\mathcal{I})} \mathbb{A}_{\varphi(v)}$.

Let $\mathbb{F} \leq_{\mathrm{sub}} \prod_{v \in \mathrm{UCT}(\mathcal{I})} \mathbb{A}'_{\varphi(v)}$ be the algebra of prime solutions to $\mathrm{UCT}(\mathcal{I})$ which is non-empty by the assumptions of our theorem. Clearly $\mathbb{F} \trianglelefteq \mathbb{E}$ and we define $\mathbb{E}', \mathbb{F}'$ to be the subalgebras of $\mathbb{E}, \mathbb{F}$ respectively consisting of solutions constant on each of $\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_k)$.

Since all the solutions to $\mathrm{UCT}(\mathcal{I})$ of the form $e \circ \varphi$ (where e is a solution to $\mathcal{I}$) are constant on all the $\varphi^{-1}(x)$'s the algebra $\mathbb{E}'$ is still subdirect in $\prod_{v \in \mathrm{UCT}(\mathcal{I})} \mathbb{A}_{\varphi(v)}$. The algebra $\mathbb{F}'$ is non-empty by the inductive assumption and, clearly $\mathbb{F}' \trianglelefteq \mathbb{E}'$.

Claim. For any two variable v, w of $\mathrm{UCT}(\mathcal{I})$ if $\varphi(v) = \varphi(w) = x$ then $\mathrm{proj}_v(\mathbb{F}') = \mathrm{proj}_w(\mathbb{F}')$.

Proof. Take any $a \in \mathrm{proj}_v(\mathbb{F}')$ given by a solution e and let ψ be the automorphism of $\mathrm{UCT}(\mathcal{I})$ mapping w to v. The solution $e \circ \psi$ is constant on $\varphi^{-1}(x_i)$ by properties of ψ and guarantees that $a \in \mathrm{proj}_w(\mathbb{F}')$. □

Claim. Let W be a finite set of variables of $\mathrm{UCT}(\mathcal{I})$ such that $\varphi(W) = \{x\}$. There exists a tuple in $\mathbb{F}'$ constant on W; equivalently there exists a prime solution to $\mathrm{UCT}(\mathcal{I})$ constant on $\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_k)$ and W.

Proof. By the previous claim $\mathbb{F}'$ projects to the same algebra (call it $\mathbb{B}$) on every variable in W. Therefore $\mathrm{proj}_W(\mathbb{F}') \leq_{\mathrm{sub}} \mathbb{B}^W$ and $\mathrm{proj}_W(\mathbb{F}') \trianglelefteq (\mathrm{proj}_W(\mathbb{E}') \cap \mathbb{B}^W)$ and the last algebra contains all the constants. By Proposition 4.4 $\mathbb{F}'$ contains a constant tuple. □

Since, for every finite $W \subseteq \varphi^{-1}(x)$, we have a prime solution constant on $\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_k)$ as well as W, the compactness argument provides a prime solution constant on all of the sets $\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_k), \varphi^{-1}(x)$. This finishes the proof of an inductive step and of Theorem 7.1 as well.

Acknowledgments

We thank Libor Barto and Jakub Bulin for many fruitful discussions and simplifications of some of the proofs in the paper. We thank anonymous reviewers for pointing out some inconsistencies in the presentation and other comments on improving readability of the paper.

References

L. Barto. The collapse of the bounded width hierarchy. *Journal of Logic and Computation*, 2014. doi: 10.1093/logcom/exu070.

L. Barto and M. Kozik. Constraint Satisfaction Problems of bounded width. In *Foundations of Computer Science, 2009. FOCS '09. 50th Annual IEEE Symposium on*, pages 595–603, Oct 2009. doi: 10.1109/FOCS.2009.32.

L. Barto and M. Kozik. New conditions for Taylor varieties and CSP. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 100–109, July 2010. doi: 10.1109/LICS.2010.34.

L. Barto and M. Kozik. Robust satisfiability of Constraint Satisfaction Problems. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 931–940, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1245-5. doi: 10.1145/2213977.2214061.

L. Barto and M. Kozik. Constraint Satisfaction Problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, Jan. 2014. ISSN 0004-5411. doi: 10.1145/2556646.

L. Barto, M. Kozik, and R. Willard. Near unanimity constraints have bounded pathwidth duality. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS '12, pages 125–134, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4769-5. doi: 10.1109/LICS.2012.24.

C. Bessiere and R. Debruyne. Theoretical analysis of singleton arc consistency and its extensions. *Artificial Intelligence*, 172(1):29 – 41, 2008. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2007.09.001.

M. Bodirsky and H. Chen. Peek arc consistency. *Theoretical Computer Science*, 411(2):445 – 453, 2010. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2009.07.059.

V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I, II. *Kibernetika (Kiev)*, (3):1–10; ibid. 1969, no. 5, 1–9, 1969. ISSN 0023-1274.

A. Bulatov. Bounded relational width. 2009. manuscript.

A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34:720–742, March 2005. ISSN 0097-5397. doi: http://dx.doi.org/10.1137/S0097539700376676.

A. A. Bulatov, A. A. Krokhin, and P. Jeavons. Constraint satisfaction problems and finite algebras. In *Automata, languages and programming (Geneva, 2000)*, volume 1853 of *Lecture Notes in Comput. Sci.*, pages 272–282. Springer, Berlin, 2000.

H. Chen, V. Dalmau, and B. Grußien. Arc consistency and friends. *Journal of Logic and Computation*, 23(1):87–108, 2013. doi: 10.1093/logcom/exr039.

V. Dalmau and A. Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821 – 837, 2008. ISSN 0195-6698. doi: http://dx.doi.org/10.1016/j.ejc.2007.11.020. Homomorphisms: Structure and Highlights Homomorphisms: Structure and Highlights.

V. Dalmau and A. Krokhin. Robust satisfiability for CSPs: Hardness and algorithmic results. *ACM Trans. Comput. Theory*, 5(4):15:1–15:25, Nov. 2013. ISSN 1942-3454. doi: 10.1145/2540090.

V. Dalmau and J. Pearson. Closure functions and width 1 problems. In J. Jaffar, editor, *Principles and Practice of Constraint Programming - CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 159–173. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66626-4. doi: 10.1007/978-3-540-48085-3_12.

R. Debruyne and C. Bessiere. Some practicable filtering techniques for the Constraint Satisfaction Problem. In *In Proceedings of IJCAI'97*, pages 412–417, 1997.

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(1):57–104, 1998. doi: 10.1137/S0097539794266766.

V. Guruswami and Y. Zhou. Tight bounds on the approximability of almost-satisfiable Horn SAT and exact hitting set. In D. Randall, editor, *SODA*, pages 1574–1589. SIAM, 2011.

B. Larose and L. Zádori. Bounded width problems and algebras. *Algebra universalis*, 56(3-4):439–466, 2007. ISSN 0002-5240. doi: 10.1007/s00012-007-2012-6.