Exploiting fixable, removable, and implied values in Constraint Satisfaction Problems

Lucas Bordeaux, Marco Cadoli, Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
`bordeaux|cadoli|tmancini@dis.uniroma1.it`

Abstract. Complete algorithms for constraint solving typically exploit properties like (in)consistency or interchangeability, which they detect by means of incomplete yet effective algorithms and use to reduce the search space. In this paper, we study a wide range of properties which includes most of the ones used by existing CSP algorithms as well as some which have not yet been considered in the literature, and we investigate their use in CSP solving. We clarify the relationships between these notions and characterise the complexity of the problem of checking them. Following the CSP approach, we then determine a number of relaxations (for instance *local* versions) which provide sufficient conditions whose detection is tractable. This work is a first step towards a comprehensive framework for CSP properties, and it also shows that new notions still remain to be exploited.

1 Introduction

Many Constraint Satisfaction Problems (CSPs) which arise in the modelling of real-life problems exhibit “structural” properties that distinguish them from random instances. Detecting such properties has been widely recognised to be an effective way for improving the solving process. To this end, several of them have already been identified, and different techniques have been developed in order to exploit them, with the goal of reducing the search space to be explored. Good examples are value substitutability and interchangeability [11], more general forms of symmetries [6, 12], and functional dependencies among variables [18, 1].

Unfortunately, checking whether such properties hold, is (or is thought to be) often computationally hard. As an example, let us consider interchangeability. Value a is said to be interchangeable with value b for variable x if every solution which assigns a to x remains a solution if x is changed to b, and vice versa [11]. The problem of checking interchangeability is coNP-complete (*cf.* Proposition 2). Analogously, detecting some other forms of symmetry reduces to the graph automorphism problem [5] (for which there are no polynomial time algorithms, even if there is evidence that it is not NP-complete [16]).

To this end, in order to allow general algorithms to exploit such properties efficiently, different approaches can be followed. First of all, syntactic restrictions on the constraint languages can be enforced, in order to allow for the efficient verification of the properties of interest. Alternatively, "local" versions of such properties can be defined, that can be used to infer their global counterparts, and such that they can be verified in polynomial time. As an instance of this "local reasoning" approach, instead of checking whether a value is *fully* interchangeable for a variable, Freuder [11] proposes to check whether that value is *neighbourhood*, or k-interchangeable. This task involves considering only subsets of the constraints of bounded size, and hence can be performed in polynomial time. Neighbourhood and k-interchangeability are sufficient (but not necessary) conditions for full interchangeability, and have been proven to be highly effective in practice (*cf.*, *e.g.,* [4, 3]). Moreover, in some cases, the existence of some properties can also derive from intrinsic characteristics of the problem, or even from an explicit promise [9], *cf.* forthcoming Example 1.

In this paper we give a formal characterisation of several properties of CSPs which can be exploited in order to save search. Some of them are well-known, while some others are, to the best of our knowledge, original. All the presented definitions are then collected in a unified framework, and hierarchically classified in order to highlight the semantical connections that hold among them. Afterwards, we present a formal discussion of their computational properties, and show how some of them can be practically exploited by the solving engine in order to save search.

In general, all these properties can be detected either statically, during a preprocessing stage of the input CSP (*cf. e.g.,* [2]), dynamically, during search (since they may arise at any time), or explicitly "promised" by an external entity.

Example 1 (Factoring [17, 23]). This problem is a simplified version of one of the most important problems in public-key cryptography. Given a (large) positive integer Z, which is known to be the product of two *prime* numbers (different from 1), it amounts to find its factors X and Y.

An intuitive formulation of this problem as a CSP, in order to deal with arbitrarily large numbers, amounts to encode the combinatorial circuit of integer multiplication, and is as follows: assuming the input integer Z having n digits (in base b) $z_1, \ldots, z_n$, we consider $2n$ variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ one for each digit (in base b) of the two factors, X and Y (with x_1 and y_1 being the least significant ones). The domain for all these variables is $[0, b-1]$. In order to maintain information about the carries, $n+1$ additional variables $c_1, \ldots, c_{n+1}$ must be considered, with domain $[0..(b-1)^2 n/b]$.

As for the constraints (cf. Fig. 1 for the intuition), they are the following:

1. Constraints on factors:
   (a) Factors must be different from 1, or, equivalently, $X \neq Z$ and $Y \neq Z$ must hold;
   (b) For every digit $i \in [1, n]$: $z_i = c_i + \sum_{j,k \in [1,n] : j+k=i+1} (x_j * y_k \mod b)$;
2. Constraints on carries:
   (a) Carry on the least significant digit is 0: $c_1 = 0$;
   (b) Carries on other digits: $\forall i \in [2, n+1]$, $c_i = c_{i-1} + \sum_{j,k \in [1,n] : j+k=i} \frac{x_j * y_k}{b}$;
   (c) Carry on the most significant digit is 0: $c_{n+1} = 0$; □

				7	8	7	*
				7	9	7	=
0	6	13	18	12	4	0	
				49	56	49	
			63	72	63	–	
		49	56	49	–	–	
	6	2	7	2	3	9	

				x_3	x_2	x_1	*
				y_3	y_2	y_1	=
c_7	c_6	c_5	c_4	c_3	c_2	c_1	
				x_3y_1	x_2y_1	x_1y_1	
			x_3y_2	x_2y_2	x_1y_2	–	
		x_3y_3	x_2y_3	x_1y_3	–	–	
	z_6	z_5	z_4	z_3	z_2	z_1	

Fig. 1. Factoring instance 627239, $n = 6$, $b = 10$

It is worth noting that, when a guess on the two factors X and Y (i.e., on variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$) has been made, values for variables $c_1, \ldots, c_{n+1}$ are completely determined, since they follow from the semantics of the multiplication. Functional dependencies arise very often, *e.g.,*, in all problems for which an intermediate state has to be maintained, and their detection and exploitation has been recognized to be of great importance from an efficiency point of view, since it can lead to significant reductions of the search space (cf., e.g., [13, 1, 2]).

The presence of functional dependencies among variables of a CSP highlights an interesting problem, i.e., that of *computing* the values of dependent variables when a choice of the defining ones has been made. It is worth noting that this problem, always present as a subproblem of a CSP with dependencies, has exactly one solution. Hence, the knowledge of such a *promise* can be useful to the solver. It is worth noting that there are also problems which intrinsically exhibit promises. This is the case of, e.g., Factoring, where we additionally add the symmetry-breaking constraint forcing $x_1, \ldots, x_n$ to be lexicographically less than or equal to $y_1, \ldots, y_n$. This new formulation is guaranteed to have exactly one solution.

In what follows, we investigate the relations that hold among different concepts. In particular, we reconsider the notions of *inconsistency*, *substitutability* and *interchangeability*, and propose the concepts of *fixable*, *removable*, and *implied* value for a given variable, and those of *determined*, *dependent*, and *irrelevant* variable. These properties make it possible to transform a problem into a simpler one. Depending on the case, this transformation is guaranteed to preserve all solutions of the problem, or to preserve at least one if one exists.

In order to give the intuition of some of the properties we are going to define, let us reconsider the Factoring problem.

Example 2 (Factoring, Example 1 continued). Let us consider an instance such that Z is given in binary notation (*i.e.,* $b = 2$) and with the least significant digit, $z_1 = 1$. This implies that the last digit of both factors X and Y must be 1. Hence, we can say that value 1 is *implied* for variables x_1 and y_1, and that 0 is *removable* for them and, more precisely *inconsistent.* Moreover, for this problem, which, if the symmetry is broken, has a unique solution, we also know that all variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are *determined* (*cf.* forthcoming Definition 1), regardless of the instance, and because of the functional dependence already discussed in Example 1, we have that variables encoding carries, i.e., c_i ($i \in [1, n]$), are *dependent* on $\{x_1, \ldots x_n,\, y_1, \ldots, y_n\}$. $\square$

Unfortunately, solving problem instances with unique solutions is likely to remain intractable (*cf.*, *e.g.,* [22]). But this does not exclude, of course, the possibility to find good heuristics for instances with such a promise, or to look for other properties that are implied by the existence of unique solutions, that can be exploited in order to improve the search process. In particular, determined and implied values play an important role in this and other classes of problems. As the previous example shows, such problems arise frequently in practice, either as subproblems of other CSPs, as in presence of functional dependencies, or because of intrinsic characteristics of the problem at hand. In general, if a problem has a unique solution, all variables have a determined value.

Another central role is played by the removability property, that characterises precisely the case when a value can be safely removed from the domain of a variable, while preserving satisfiability. This property is of course weaker than inconsistency, (since some solutions may be lost), but can be safely used in place of it when we are interested in finding only a solution of the input CSP, if one exists, and not all of them.

Unfortunately, detecting the proposed properties is computationally hard in general. In particular, we show that these tasks are all coNP-complete. This holds also for Freuder's substitutability and interchangeability (this result is, to the best of our knowledge, original). Hence, in order to be able to practically make the relevant checks during preprocessing and search, we show how some of the proposed properties can be verified efficiently along two lines: by imposing restrictions on the constraint language, and by exploiting locality, *i.e.,* by checking them for single constraints.

The outline of the paper is as follows: after recalling some preliminaries, in Section 2 we formally define all the properties we are interested in, and discuss the semantical connections that hold among them. Then, in Section 3 we present the intractability results of checking such properties. Hence, in Section 4 we show some tractability results, investigating the two aforementioned approaches in order to be able to efficiently make the required reasoning: imposing restrictions on the constraint languages, and exploring locality. Finally, in Section 5 we draw conclusions and address future work.

2 A hierarchy of properties

2.1 Preliminaries

Let $\mathbb{D}$ be a finite set of size at least 2. A V-tuple t, where V represents a finite set of variables, is a mapping which associates a value $t_x \in \mathbb{D}$ to every $x \in V$. A V*-relation* is a set of V-tuples. A *Constraint Satisfaction Problem* (CSP) is a triple $\langle X, D, C\rangle$ where:

- X is a finite set of variables,
- D associates to every variable $x \in V$ a domain $D_x \subseteq \mathbb{D}$ and
- C is a finite set of constraints, each of which is a V-relation for some $V \subseteq X$.

Given a V-tuple t and a subset $U \subseteq V$ of its variables, we denote by $t|_U$ the *restriction* of t to U, which has the same value as t on the variables of U and is undefined elsewhere. The explicit assignment of the value of a V-tuple t on a variable $x \in V$ to value a is written $t[x := a]$. The relational operators of *selection*, *projection* and *complement* will be useful: given a V-relation c, a subset U of V and a value $a \in D_x$, we denote by $\sigma_{x=a}(c)$ (*resp.* $\sigma_{x\neq a}(c)$) the V-relation which contains the tuples of c whose value on x is a (*resp.* is different from a), by $\pi_U(c)$ the set of restrictions to U of tuples of c (*i.e.,* the set of U-tuples $\{t \mid \exists t' \in c \ (t = t'|_U)\}$) and by $\overline{c}$ the set of V-tuples $\{t \mid t \notin c\}$.

An X-tuple t *satisfies* a V-relation $c \in C$ if $t|_V \in c$. We denote by $Sol(c)$ the set of X-tuples which satisfy c. The set $\bigcap_{c\in C} Sol(c)$ of X-tuples which satisfy all the constraints is called the *solution space*, and denoted $Sol(C)$. The set of X-tuples t such that $t_x \in D_x$ for all variables x is called the *search space* and noted S_D, or simply S if the domain is implicit from the context. Note that $\sigma_{x=a}(S)$ denotes the search space obtained by fixing D_x to $\{a\}$ if $a \in D_x$ and is empty otherwise. For the sake of simplicity, the sets X and C will be considered as globally defined and shall therefore be omitted from the parameters of most definitions; only the search space will be explicitly mentioned.

2.2 Definitions

Definition 1. *The following properties are defined for a search space S, variables x and y, values a and b, and for a set of variables V:*

$$\mathit{fixable}(S,x,a) \equiv \quad \forall t \in S \ \ (t \in Sol(C) \ \rightarrow \ t[x := a] \in Sol(C))$$

$$\mathit{substitutable}(S,x,a,b) \equiv \quad \forall t \in S \ \begin{pmatrix} t_x = a \wedge t \in Sol(C) \ \rightarrow \\ \qquad t[x := b] \in Sol(C) \end{pmatrix}$$

$$\mathit{interchangeable}(S,x,a,b) \equiv \quad \begin{array}{l} \mathit{substitutable}(S,x,a,b) \wedge \\ \mathit{substitutable}(S,x,b,a) \end{array}$$

$$\mathit{removable}(S,x,a) \equiv \quad \forall t \in S \ \begin{pmatrix} t_x = a \wedge t \in Sol(C) \ \rightarrow \\ \qquad \exists b \neq a \ (t[x := b] \in Sol(C)) \end{pmatrix}$$

$$\mathit{inconsistent}(S,x,a) \equiv \quad \forall t \in S \ \big(t \in Sol(C) \ \rightarrow \ t_x \neq a)\big)$$

$$\mathit{implied}(S,x,a) \equiv \quad \forall t \in S \ \big(t \in Sol(C) \ \rightarrow \ t_x = a)\big)$$

$$\mathit{determined}(S,x) \equiv \quad \forall t \in S \ \begin{pmatrix} t \in Sol(C) \ \rightarrow \\ \qquad \forall b \neq t_x \ (t[x := b] \notin Sol(C)) \end{pmatrix}$$

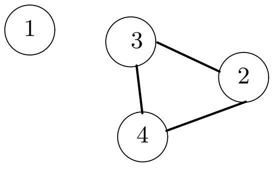
$$\mathit{dependent}(S,V,y) \equiv \quad \forall t,t' \in S \left(\begin{pmatrix} t \in Sol(C) \wedge \\ t' \in Sol(C) \wedge \\ \forall x \in V \ (t_x = t'_x) \end{pmatrix} \rightarrow t_y = t'_y \right)$$

$$\mathit{irrelevant}(S,x) \equiv \quad \forall t \in S \ \begin{pmatrix} t \in Sol(C) \ \rightarrow \\ \qquad \forall a \in D_x \ (t[x := a] \in Sol(C)) \end{pmatrix}$$

In the few cases where an ambiguity arises on the considered set of constraints, we will indicate it using subscript (*e.g.,* $\mathit{irrelevant}_C(S,x)$). Note that all the definitions but the last three ones are *value*-oriented, in that they are properties of particular values of the domain. On the contrary, dependency, irrelevance and determinacy are *variable-oriented* properties which do not directly express results on particular values of the domains but have important relations with the value-oriented notions.

The notion of consistency was proposed in [21, 19] and is one of the best-studied notions in CSP. Substitutability and interchangeability were introduced in [11]. Implied values, which are known as *backbones* in the literature, were seemingly first explicitly studied in [20]. To the best of our knowledge, the notion of removable and fixable values have on the contrary not been considered. Determined, irrelevant and dependent variables have been studied in a number of contexts but we are aware of little work concerning their application in the context of CSP. The following example illustrates some of the properties.

Example 3. Consider a CSP modeling the colouring problem for the graph below. Let $c1 \ldots c4$ denote the variables involved, and Σ denote the search space in which all four variables have domain $\{R, G, B\}$. We have:



– *fixable(*Σ*,*$c1$*,*R*)*,
– *substitutable(*Σ*,*$c1$*,*R*,*G*)*,
– *interchangeable(*Σ*,*$c1$*,*R*,*G*)*,
– *removable(*Σ*,*$c1$*,*G*)*,
– *irrelevant(*Σ*,*$c1$*)*.

Example 4. Consider a CSP over boolean variables a, b, and c, whose constraints are written below. Denoting as Ξ the search space in which all variables range over $\{\mathit{true}, \mathit{false}\}$. We have:

$$\begin{array}{ll} a & \wedge \\ a \rightarrow b & \wedge \\ (c \vee d) \leftrightarrow e & \end{array}$$

– *inconsistent(*Ξ*,*b*,false)*,
– *implied(*Ξ*,*b*,true)*,
– *determined(*Ξ*,*b*)*,
– *dependent(*Ξ*,*$\{c, d\}$*,*e*)*.

2.3 Semantical relations

The notions presented in Definition 1 are semantically connected, and we clarify here the main relationships that exist between them.

Proposition 1. *The relations shown in Figure 2 hold between the properties defined in Definition 1.*

Proof. (sketch)

dependence-determinacy: *we have dependent*$(S, \{x_1, \ldots, x_i\}, y)$ *iff any solution* t *has a value on* y *which is given by a function* f *of the values it assigns to* $x_1 \ldots x_i$*, iff in any search space* $\sigma_{x_1=a_1 \ldots x_i=a_i}(S)$ *(where all these variables receive a fixed value), all solutions assign the same value* $f(a_1, \ldots, a_n)$ *to* y*.*

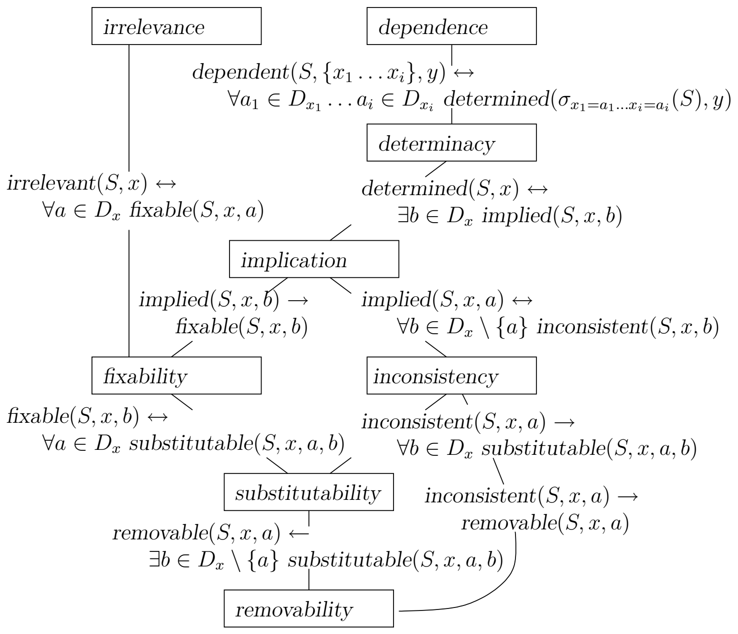

irrelevance
dependence
$dependent(S, \{x_1 \ldots x_i\}, y) \leftrightarrow \forall a_1 \in D_{x_1} \ldots a_i \in D_{x_i}\ determined(\sigma_{x_1=a_1..x_i=a_i}(S), y)$
determinacy
$irrelevant(S, x) \leftrightarrow \forall a \in D_x\ fixable(S, x, a)$
$determined(S, x) \leftrightarrow \exists b \in D_x\ implied(S, x, b)$
implication
$implied(S, x, b) \rightarrow fixable(S, x, b)$
$implied(S, x, a) \leftrightarrow \forall b \in D_x \setminus \{a\}\ inconsistent(S, x, b)$
fixability
inconsistency
$fixable(S, x, b) \leftrightarrow \forall a \in D_x\ substitutable(S, x, a, b)$
$inconsistent(S, x, a) \rightarrow \forall b \in D_x\ substitutable(S, x, a, b)$
substitutability
$inconsistent(S, x, a) \rightarrow removable(S, x, a)$
$removable(S, x, a) \leftarrow \exists b \in D_x \setminus \{a\}\ substitutable(S, x, a, b)$
removability

Fig. 2. Relations between the properties

irrelevance-fixability: *$t \in Sol(C) \rightarrow \forall a \in D_x(t[x := a] \in Sol(C))$ rewrites to $\forall a \in D_x(t \in Sol(C) \rightarrow t[x := a] \in Sol(C))$.*

determinacy-implication: *if we have $implied(S, x, b)$ for some b, then for any t and any $a \neq b$ we have $t[x := a] \notin Sol(C)$. If we have $determined(S, x)$ and $t \in Sol(C)$, then $implied(S, x, t_x)$ (no t' with $t'_x \neq t_x$ is in $Sol(C)$).*

implication-fixability: *$implied(S, x, b)$ means that every $t \in Sol(C)$ has $t_x = b$. Hence for every $t \in Sol(C)$, we have $t[x := b] = t \in Sol(C)$.*

implication-inconsistency: *$implied(S, x, a)$ holds iff $\forall t\ (t_x \neq a \rightarrow t \notin Sol(C))$, i.e., iff $\forall t\ \forall b \in D_x \setminus \{a\}\ (t_x = b \rightarrow t \notin Sol(C))$. This rewrites to $\forall b \in D_x \setminus \{a\}\ inconsistent(S, x, b)$.*

fixability-substitutability: *Let $D_x = \{a_1, .., a_d\}$. We have $\bigwedge_{i \in 1..d} substitutable(S, x, a_i, b)$ iff $\forall t\ ((t_x = a_1 \vee \cdots \vee t_x = a_d) \wedge t \in Sol(C) \rightarrow t[x := b] \in Sol(C))$, which rewrites to $fixable(S, x, v)$.*

inconsistency-substitutability: *suppose we have $inconsistent(S, x, a)$. No solution t with $t_x = a$ exists, hence the implication $t_x = a \wedge t \in Sol(C) \rightarrow \ldots$ is always valid.*

inconsistency-removability: *same argument as for inconsistency-substitutability.*

substitutability-removability: *suppose we have $substitutable(S, x, a, b)$ for some value b. This can be written $\exists b\ \forall t\ (t_x = a \wedge t \in Sol(C) \rightarrow t[x := b] \in Sol(C))$, which implies that $\forall t\ \exists b (t_x = a \wedge t \in Sol(C) \rightarrow t[x := b] \in Sol(C))$. The latter rewrites to $\forall t\ (t_x = a \wedge t \in Sol(C) \rightarrow \exists b\ t[x := b] \in Sol(C))$.*

Note also that determined values are strongly related to problems with a *unique solution*: if a problem has a unique solution, then all its variables have an implied value (*cf.* Example 1).

2.4 Exploiting properties in constraint solving

An important reason why the aforementioned properties are interesting is that, when detected, they allow us to reduce the search space by removing values. Two key notions here are inconsistency and removability:

- Suppressing a value a from the domain of a variable x preserves all solutions (*i.e.,* $\sigma_{x \neq a}(S) \cap Sol(C) = S \cap Sol(C)$) iff a is inconsistent for variable x.
- Suppressing a value a from the domain of variable x preserves the *satisfiability* of the problem (*i.e.,* $\sigma_{x \neq a}(S) \cap Sol(C) = \varnothing \leftrightarrow S \cap Sol(C) = \varnothing$) iff value a is removable from the domain of x.
- Instantiating a value a from the domain of variable x preserves the *satisfiability* of the problem (*i.e.,* $\sigma_{x=a}(S) \cap Sol(C) = \varnothing \leftrightarrow S \cap Sol(C) = \varnothing$) if value a is fixable for x.

The removability property is therefore weaker than the inconsistency one, and this shows an interesting benefit: in cases where we do not want to find all solutions of a problem but we simply want to find one, removability is the ideal property to use.

Some of these definitions can be used to construct *solution-preserving* mappings, *i.e.,* mappings which transform solutions into solutions.

Definition 2 (solution-preserving transformation). *A* solution-preserving transformation *is a total mapping τ from S to S such that*

$$\forall t \in S \ (t \in Sol(C) \rightarrow \tau(t) \in Sol(C))$$

To understand the connection between solution-preserving transformations and the aforementioned properties, consider the following mappings:

$$\tau_1(t) = t[x := a]$$

$$\tau_2(t) = \begin{cases} t[x := b] & \text{if } t_x = a \\ t & \text{otherwise} \end{cases} \qquad \tau_3(t) = \begin{cases} t[x := b] & \text{if } t_x = a \\ t[x := a] & \text{if } t_x = b \\ t & \text{otherwise} \end{cases}$$

Checking whether value a is fixable for variable x, whether value a is substitutable to value b for variable x, and whether values a and b are interchangeable for value x amounts to checking whether mappings τ_1, τ_2 and τ_3 (respectively) are solution-preserving.

3 Intractability results

In this section, we show that the problem of checking whether properties defined in Definition 1 hold is intractable. From now on, we assume that the input is given as a set of constraints C over a set of variables X. We also assume that the problem of checking whether $t \in Sol(C)$ is polynomial in the size of the representation of the input. Additionally, we assume that the size of D is fixed. Such properties hold for propositional logic and for CSPs, in the sense of [8].

We note that the problem of checking each property of Definition 1 is in coNP, because it can be done by guessing all tuples in S in non-deterministic polynomial time, and making the relevant tests in polynomial time (as for interchangeability, we note that the logical and of two properties in coNP is still in coNP). In the rest of this section, proofs are therefore restricted to the coNP-hardness part.

Proposition 2 (coNP-completeness of properties of Definition 1). *Given a CSP, the following tasks are coNP-complete:*

- *Checking whether value a is fixable, removable, inconsistent, implied, determined for variable x;*
- *Checking whether value a is substitutable to, or interchangeable with b for variable x;*
- *Checking whether variable y is dependent on variables in V;*
- *Checking whether variable x is irrelevant.*

Proof. For the sake of simplicity, we give the proofs for fixability and substitutability. The other proofs can be given in a similar way, by using also Proposition 1. To prove that checking fixability and substitutability are hard for coNP, we reduce a coNP-complete problem, i.e., that of checking that an arbitrary CSP is unsatisfiable, to fixability and substitutability. In particular, the proofs hold even if the domains are binary, in which case the CSP can be written as a propositional formula, e.g., in CNF.

Fixability. Let us consider an arbitrary propositional formula ϕ in CNF, over variables X, and a variable $x \notin X$. Let ψ be defined as $\phi \wedge \neg x$. We have that ψ is unsatisfiable if and only if ϕ is unsatisfiable.

We now show that ϕ is unsatisfiable if and only if value true *is fixable for x in formula ψ. Let us first assume that ϕ is unsatisfiable. It follows that* true *is fixable for x in ψ, because ψ has no models.*

As for the other direction, by Definition 1, if true *is fixable for x in ψ, then, every model of ψ remains a model if x is assigned to* true*. However, since, by construction, models of ψ never assign* true *to x, it follows that* true *is fixable for x in ψ only if no solutions to ψ exist, hence, only if ϕ is unsatisfiable.*

Substitutability. Let us consider an arbitrary propositional formula ϕ in CNF, over variables X, and a variable $x \notin X$. Let ψ be the defined as $\phi \wedge x$. We have that ψ is unsatisfiable if and only if ϕ is unsatisfiable.

We now show that ϕ is unsatisfiable if and only if value true *is substitutable to* false *for x in ψ. Let us first assume that ϕ is unsatisfiable. It follows that* true *is substitutable to* false *for x in ψ, because ψ has no models.*

As for the other direction, by Definition 1, if true *is substitutable to* false *for x in ψ, then, every model of ψ with x assigned to* true *remains a model if x is assigned to* false*. However, since, by construction, models of ψ never assign* false *to x, it follows that true is substitutable to false for x in ψ only if no solutions to ϕ exist.*

It is worth noting that the intractability of checking the above properties hold also for binary CSPs (*i.e.,* CSPs in which all constraints relate at most two variables). As an example, the following result holds.

Corollary 1 (coNP-completeness of fixability for binary constraints). *Given a CSP with only binary constraints, checking whether a value a is fixable for a variable x is coNP-complete.*

Proof. Let $\Phi = \langle X, D, C\rangle$ be a binary CSP. Consider an arbitrary variable $y \notin X$ and let a and b be arbitrary values. Let Ψ denote the CSP $\langle X', D', C'\rangle$ with $X' = X \cup \{y\}$, $D'_x = D_x$ forall $x \in X$, $D'_y = \{a, b\}$, and $C' = C \cup \{y \neq a\}$. Ψ is binary and, by using the same arguments of the proof of Proposition 2, it follows that Φ is unsatisfiable if and only if value a for variable y is fixable for Ψ. From the observation that a CSP encoding of the graph 3-colourability problem can be made using only binary constraints, the thesis follows, since checking unsatisfiability of this problem (which is coNP-hard) can be reduced into checking fixability in a binary CSP.

4 Tractability results

Since detecting any of the properties we are interested in in the paper is a computationally hard problem, a natural question is to determine special cases where this can be done efficiently. We investigate two approaches: we exhibit syntactical restrictions which make the problem tractable, and we study *local* relaxations of these definitions which are polynomial-time checkable, and which therefore provide incomplete algorithms for detecting the property.

4.1 Tractability for restricted constraint languages

A number of syntactical restrictions to the constraint satisfaction problem are known which make it tractable. For instance, in the case of boolean constraints, *i.e.,* propositional formulae, the satisfiability problem becomes tractable if the instance is expressed using only Horn clauses, only dual Horn clauses (*i.e.,* clauses with at most one negative literal), only clauses of size at most 2, or only affine constraints (*i.e.,* constraints built using XOR) [24]. It is natural to wonder if all the properties identified in Definition 1 are also easy to determine for these classes of formulae. This is indeed the case for most of them, and we give a more general condition under which tractable classes for the consistency property are also tractable for other properties of our framework. We note that a recent paper [15] gives a complete characterization of tractable cases for a related property.

We say that a language is *closed under instantiation* (*resp. under complementation*) if whenever a constraint c is expressible in the language, the relation $\pi_{X\setminus\{x\}}(\sigma_{x=a}(c))$ (*resp.* the complementation $\overline{c}$) is also representable by a conjunction of constraints of this language. For instance, taking a Horn clause, a dual Horn clause, a 2CNF clause or an affine constraint, we can express the relation obtained by instantiating a variable to a value or by complementing the constraint as a conjunction of constraints of the same type.

Proposition 3. *If the satisfiability problem for the language is tractable and if the language is closed under complementation and instantiation, then checking any property among fixability, substitutability, interchangeability, inconsistency, determinacy or irrelevance is tractable.*

Proof. We start by the substitutability property and note that value a is substitutable by b for variable x if

$$\pi_{X\setminus\{x\}}(\sigma_{x=a}(Sol(C))) \quad\subseteq\quad \pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(C)))$$

This inclusion holds iff $t_x = a \wedge t \in Sol(C) \rightarrow t[x := b] \in Sol(C)$. This inclusion is false, i.e., we do not *have substitutability if the set*

$$\pi_{X\setminus\{x\}}(\sigma_{x=a}(Sol(C))) \quad\cap\quad \overline{\pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(C)))} \tag{1}$$

is non empty. Since $\sigma_{x=a}(Sol(C)) = \sigma_{x=a}(\bigcap_{c\in C} Sol(c)) = \bigcap_{c\in C}(\sigma_{x=a}(Sol(c)))$, we have:

$$\pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(C))) \;=\; \pi_{X\setminus\{x\}}\left(\bigcap_{c\in C}(\sigma_{x=b}(Sol(c)))\right)$$

Although the projection of an intersection of relations is not equal to the intersection of their projections in general, the latter rewrites to:

$$\bigcap_{c\in C} \pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(c)))$$

This is due to the fact that we select on x before eliminating it by projection. We only prove the inclusion which does not hold in general: suppose we have $t \in \bigcap_{c\in C} \pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(c)))$. This means that $\forall c \in C$, there exists a tuple t^c such that $t^c|_{X\setminus\{x\}} = t$ and $t^c \in \sigma_{x=b}(Sol(c))$. It follows that $t^c_x = b$ and that we have indeed a unique t with $t_x = b$ and $t^c|_{X\setminus\{x\}} = t$ which is such that $\forall c \in C$ $(t \in \sigma_{x=b}(Sol(c)))$, i.e., $t \in \pi_{X\setminus\{x\}}(\bigcap_{c\in C}(\sigma_{x=b}(Sol(c))))$.

Equation (1) is therefore equivalent to:

$$\pi_{X\setminus\{x\}}(\sigma_{x=a}(Sol(c))) \quad\cap\quad \bigcup_{c\in C} \overline{\pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(c)))}$$

A solution exists (and we therefore do not have substitutability) if one of the sets

$$\pi_{X\setminus\{x\}}(\sigma_{x=a}(Sol(c))) \quad\cap\quad \overline{\pi_{X\setminus\{x\}}(\sigma_{x=b}(Sol(c)))}$$

obtained for every $c \in C$ has a solution. If the language is closed under instantiation and complement, we can express the new constraint $\overline{\pi_{X\setminus\{x\}}(\sigma_{x=a}(Sol(c)))}$ as a constraint c' of the language. Each of the sets has a solution iff the CSP $\langle X, D, \{\sigma_{x=a}(c) \mid c \in C\} \cup \{c'\}\rangle$ is satisfiable. We have reduced the substitutability testing problem to solving m instances of a constraint satisfaction problem whose constraints are all in the original language, which is tractable.

The result for the fixability, interchangeability and irrelevance properties follows directly. Consistency of value a for variable x can directly be expressed as the satisfiability of $\pi_{X\setminus\{x\}}(\sigma_{x=a}(Sol(C)))$, which can be expressed in the language since we assume closure under instantiation, and the proofs for the implication and determinacy properties follow from this result.

A slightly different closure property is needed for the removability of value a for variable x since it is expressed as $\pi_{X\setminus\{x\}}(Sol(C)) \;\subseteq\; \pi_{X\setminus\{x\}}(\sigma_{x\neq a}(Sol(C)))$.

Nevertheless, since on boolean domains a value v is *removable* if v is substitutable by $\neg v$, and from the remarks on the closure properties of Schaefer's classes, and the previous proposition, we obtain that:

Corollary 2. *Testing fixability, substitutability, interchangeability, inconsistency, determinacy, irrelevance and removability is tractable for a boolean CSP where constraints are either Horn clauses, dual Horn clauses, clauses of size at most two or affine constraints.*

4.2 Tractability through locality

An important class of incomplete criteria to determine in polynomial time whether a complex property holds are those based on *local* reasoning. This approach has proved extremely successful for consistency [19] and interchangeability [11] properties. We propose in this section a systematic investigation of whether a local approach can be used for value-based properties.

Verifying a property $P(C)$ of a set of constraints C *locally* means that we verify the property on a well-chosen number of sub-problems. We must ensure that this approach is sound for the considered property:

Definition 3 (soundness of local reasoning). *We say that local reasoning on a property P is* sound *if, for all subsets of constraints $C_1 \subseteq C, \ldots, C_k \subseteq C$ such that $\bigcup_{i\in 1..k} C_i = C$, we have*

$$\left(\bigwedge_{i\in 1..k} P(C_i)\right) \;\rightarrow\; P(C)$$

Note that if a property P satisfies this requirement, its negation satisfies a stronger soundness property: $\left(\bigvee_{i\in 1..k} \neg P(C_i)\right) \rightarrow \neg P(C)$. A typical choice of granularity is to simply consider that each C_i contains one of the constraints of C as is done, for instance, for arc-consistency. On the other extreme, if we take a unique $C_1 = C$, we have a global checking. Between these two extremes, a wide range of intermediate levels can be defined [10, 11].

Reasoning locally is typically tractable if we focus on a moderate number of subsets of C, and under the condition that we can bound the complexity of reasoning on each of these subsets. A typical assumption in CSP is that we can bound the arity of the constraints, and that every constraint is for instance binary. In this case, the cost of determining any property of the constraint is polynomial (here again we are indeed polynomial *in the domain size*, we therefore assume that the input is represented in a way polynomial in the domain size, for instance with the domains listed explicitly); and if we choose to reason locally by considering each constraint separately, or by taking groups of constraints of bounded size, then local checking is tractable.

Proposition 4. *Local reasoning is sound for the properties of substitutability, interchangeability, fixability, inconsistency and implication.*

Proof. *The result is well-known for* consistency *[19], substitutability and interchangeability [11]. Fixability of variable x to value b can be expressed as*

$$\forall a \neq b \ (\mathit{substitutable}_{C}(S, x, a, b))$$

Therefore, if we have $\bigwedge_{i\in 1..k} \mathrm{fixable}_{C_i}(S,x,b)$ (which is equivalent to $\bigwedge_i \bigwedge_{a\neq b}$ $\mathit{substitutable}_{C_i}(S,x,a,b)$ and to $\bigwedge_{a\neq b}\bigwedge_i$ $\mathit{substitutable}_{C_i}(S,x,a,b)$), then we have $\bigwedge_{a\neq b}$ $\mathit{substitutable}_{C}(S,x,a,b)$, which means $\mathrm{fixable}_{C}(S,x,b)$. The implication property satisfies the following, stronger property (which implies that local reasoning is sound):

$$\left(\bigvee_{i\in 1...m} \ \mathit{implied}_{C_i}(S,x,a)\right) \quad \rightarrow \quad \mathit{implied}_{C}(S,x,a)$$

In effect, if a value a is implied for variable x in any C_i, then all tuples t with $t_x \neq a$ violate the constraints of C_i and do a fortiori *not belong to $Sol(C)$.*

There is only one property, namely removability, for which the local approach is unfortunately not sound:

Proposition 5. *Local reasoning is* ***not*** *sound for the removability property.*

Proof. *Take $C = C_1 \wedge C_2$, where C_1 is defined as $x \leq y$ and C_2 as $x \geq y$. Suppose the domain has values $\{1, 2, 3\}$. Value 2 for x is removable from both constraints considered independently since, in both cases, we can change the value of any solution which assigns 2 to x to another value. Still, value 2 is not removable from their conjunction.*

Note that removing values which are shown to be removable only locally can even make a satisfiable problem unsatisfiable: if furthermore we add the constraints C_3, defined as $x \neq 1$ and C_4, defined as $x \neq 3$, then value 2 for x is removable in each constraint, while the only (global) solution actually has value 2 on x.

This proposition raises an interesting issue: does there exist new (*i.e.,* other than the special cases of substitutable and inconsistent values) properties for which local reasoning is sound and which imply removability?

We end this section by noting that the local version of the fixability property is indeed a generalisation for arbitrary domains of the *pure literal rule* [7] which is well-known in the case of boolean constraints in conjunctive normal form. The pure literal rule exploits the cases where no constraint (clause) of the problem has a positive (*resp.* negative) occurrence of some variable x. In this case, assigning value 0 (*resp.* 1) to x preserves the satisfiability of the problem: if a solution t with $t_x = 1$ exists, then $t[x := 0]$ will also be a solution since no clause constrains x to have value 1. It is clear that the pure literal rule is a rule to detect fixability based on a reasoning local to each clause (a variable x is fixable to, say, 1 in a clause iff this clause does not contain the literal $\neg x$, and the pure literal rule checks that this condition holds for every constraint).

5 Conclusions and perspectives

In this paper we focused on structural properties of CSPs that can be exploited by the solver in order to simplify search. Starting from the well-known notions of inconsistency and substitutability, we propose *removability* as a property which subsumes both of them, as well as several new others, *e.g.,* fixability, which are particularly interesting if we want to find just a solution of the input CSP, and not to compute all of them. By classifying these properties in a unified hierarchy, we investigated the semantical connections among them, and provided a first step towards a comprehensive framework. Note that our central definitions are *value-based* and that more general definitions inspired from the *tuple-based* notion of substitutability proposed in [14] could be considered in future work.

Then, we tackled the questions related to their automated detection and of their exploitation by the solving engine for simplifying problems. In particular, we showed how detecting all the proposed properties is generally intractable, but, for many of them, it becomes polynomial-time in two cases: by restricting the constraint languages, and by exploiting locality. Moreover, we discussed how in some cases such properties may arise from explicit promises made by users. This is the case of problems with properties such as functional dependencies and unique solutions.

Two of the perspectives raised by our work concern the new properties which have emerged from it. We have identified the removability property as an ideal characterisation of the values which can be removed while preserving satisfiability. Unfortunately, negative results (coNP-completeness of the detection of this property and impossibility of local reasoning) make it impossible to directly use the removability property in practice. This justifies the use of weaker notions (like inconsistency or substitutability) which imply the removability property, yet can be checked by tractable means (of course at the price of losing completeness). An interesting problem is to determine new cases where removability-checking is tractable. Lastly, the benefits of fixability have long been known in the boolean case, since this property has been used in the form of the pure literal rule in many SAT solvers. Its generalisation to CSPs has not yet been considered, and will be the subject of future work. Another issue we intend to explore is the application of the properties of Definition 1 to problems not in NP, *e.g.,* to model checking of formulae of temporal logic or quantified boolean formulae.

Acknowledgements. Work partially supported by project ASTRO funded by the Italian Ministry for Research under the FIRB framework (funds for basic research), and by a COFIN/PRIN project. We thank the reviewers for their careful reading which helped us improving the paper.

References

1. M. Cadoli and T. Mancini. Exploiting functional dependencies in declarative problem specifications. In *9th Euro. Conf. on Logic in Artificial Intelligence (JELIA)*, pages 628–640. Springer, 2004.
2. M. Cadoli and T. Mancini. Using a theorem prover for reasoning on constraint problems. In *3rd Int. CP Workshop on Modelling and Reformulating CSPs*, 2004.

3. B. Choueiry, A. Lal, and E. C. Freuder. Interchangeability and dynamic bundling for non-binary finite CSPs. In *Int. Workshop on Constraint Solving and Constraint Logic Programming (CSCLP)*, page To appear. Springer, 2004.
4. B. Choueiry and G. Noubir. On the computation of local interchangeability in discrete constraint satisfaction problems. In *Nat. (US) Conf. on Artificial Intelligence (AAAI)*, pages 326–333. AAAI, 1998.
5. J. M. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic (extended abstract). In *AAAI Workshop on Tractable Reasoning*, 1992.
6. J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 148–159. Morgan Kaufmann, 1996.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. of the ACM*, 7(3):201–215, 1960.
8. R. Dechter. Constraint networks (survey). In *Encyclopedia of Artificial Intelligence, 2nd edition*, pages 276–285. 1992.
9. S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.
10. E. C. Freuder. Synthesizing constraint expressions. *Comm. of the ACM*, 21(11):958–966, 1978.
11. E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Nat. (US) Conf. on Artificial Intelligence (AAAI)*, pages 227–233. AAAI Press, 1991.
12. I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *Euro. Conf. on Artificial Intelligence (ECAI)*, pages 599–603. IOS Press, 2000.
13. E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Nat. (US) Conf. on Artificial Intelligence (AAAI)*, pages 948–953. AAAI, 1998.
14. P. Jeavons, D. A. Cohen, and M. C. Cooper. A substitution operation for constraints. In *Int. Conf. on Principles and Practice of Constraint Programming (CP)*, pages 1–9. Springer, 1994.
15. P. Jonsson and A. Krokhin. Recognizing frozen variables in constraint satisfaction problems. *Theoretical Computer Science (TCS)*, 329(1-3):93–113, 2004.
16. J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its computational complexity.* Birkhauser, 1993.
17. A. Lenstra and H. W. Lenstra. Algorithms in number theory. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science, vol. 1: Algorithms and Complexity.* MIT Press, 1990.
18. C. M. Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Nat. (US) Conf. on Artificial Intelligence (AAAI)*, pages 291–296. AAAI press, 2000.
19. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
20. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.
21. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(2):85–132, 1974.
22. Ch. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.
23. T. Pyhälä. Factoring benchmarks for SAT solvers. Technical report, Helsinki university of technology, 2004.
24. T. J. Schaefer. The complexity of satisfiability problems. In *ACM Symp. on Theory of Computing (STOC)*, pages 216–226. ACM, 1978.