Beyond Consistency and Substitutability

Martin C. Cooper

Beyond Consistency and Substitutability*

Martin C. Cooper

IRIT, University of Toulouse III, 31062 Toulouse, France
cooper@irit.fr

Abstract. Elimination of inconsistent values in instances of the constraint satisfaction problem (CSP) conserves all solutions. Elimination of substitutable values conserves at least one solution. We show that certain values which are neither inconsistent nor substitutable can also be deleted while conserving at least one solution. This allows us to state novel rules for the elimination of values in a binary CSP. From a practical point of view, we show that one such rule can be applied in the same asymptotic time complexity as neighbourhood substitution but is strictly stronger.

An alternative to the elimination of values from domains is the elimination of variables. We give novel satisfiability-preserving variable elimination operations. In each case we show that if the instance is satisfiable, then a solution to the original instance can always be recovered in low-order polynomial time from a solution to the reduced instance.

1 Introduction

Operations to reduce the worst-case exponential time complexity of exhaustive search are essential for the efficient resolution of large-scale constraint satisfaction problems. Reduction operations are most effective at reducing search space size when applied during search, but if this is too computationally expensive they can still be usefully applied just once during a preprocessing phase. Most previous research in this domain has concentrated on domain-filtering operations based on various forms of consistency: a value is removed from a domain if an algorithm running in low-order polynomial time demonstrates that this assignment cannot be part of a solution. Other reduction operations include the elimination of values by interchangeability or substitutability [7,11], the merging of domain values [10], the elimination of variables [9,5,3] and the introduction of symmetry-breaking constraints [1,8].

This paper studies local (and hence polytime-detectable) properties of binary CSP instances which allow value elimination or variable elimination while preserving satisfiability. We show that allowing arbitrary quantification over variables and values as well as arbitrary conditions on the compatibilities of pairs of assignments provides a rich and largely unexplored source of reduction operations.

* Supported by ANR Project ANR-10-BLAN-0210 and EPSRC grant EP/L021226/1.

Definition 1. *A* binary CSP instance *I consists of*

- *a set X of n variables,*
- *a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$,*
- *a relation $R_{xy} \subseteq \mathcal{D}(x) \times \mathcal{D}(y)$, for each pair of distinct variables $x, y \in X$, which consists of the set of pairs of values (a, b) which can simultaneously be assigned to variables (x, y).*

A partial solution *to I on $Y \subseteq X$ is a mapping $s : Y \to D$ where, for all $x \neq y \in Y$ we have $(s(x), s(y)) \in R_{xy}$. A* solution *to I is a partial solution on X.*

For simplicity of presentation, Definition 1 assumes that there is exactly one constraint relation for each pair of variables $\{x, y\}$. If $R_{xy} \neq \mathcal{D}(x) \times \mathcal{D}(y)$, then we say that variable x *constrains* variable y. If $(a, b) \in R_{xy}$, then the assignments $\langle x, a\rangle$ and $\langle y, b\rangle$ are *compatible*, otherwise *incompatible*.

In previous work we showed that there are exactly four variable-elimination rules based on so-called irreducible existential patterns [3]. In the present paper we give strict generalisations of all these rules. We also give value-elimination rules which are strict generalisations of neighbourhood substitution [7]. The paper is organised as follows: Section 2 and Section 3 present rules for, respectively, value elimination and variable elimination, Section 4 gives a particular value-elimination rule which generalises neighbourhood substitution but can be applied in the same time complexity, Section 5 gives the complexity of recovering all solutions after applying our value or variable elimination rules, while Section 6 discusses the difficulty of characterising all value or variable elimination rules based on local properties.

2 Value Elimination

For each rule which tells us when a value can be eliminated from a domain, there is a corresponding property which holds if and only if no value eliminations can be performed by this rule. Following the tradition of consistency properties, we state our rules in the form of positive properties which are satisfied if and only if no eliminations are possible.

We begin by recalling the notions of arc consistency and neighbourhood substitution, illustrated in Figure 1. In figures, each bullet represents a variable-value assignment, assignments to the same variable are grouped together within the same oval and compatible (incompatible) pairs of assignments are linked by solid (broken) lines.

Definition 2. *A value $b \in \mathcal{D}(x)$ is* AC-supported *if $\forall y \in X \setminus \{x\}$, $\exists c \in \mathcal{D}(y)$ such that $(b, c) \in R_{xy}$. We say that c is an* AC support *for $\langle x, b\rangle$ at y.*

Any assignment value $b \in \mathcal{D}(x)$ which is not AC-supported can be eliminated from $\mathcal{D}(x)$ without losing any solutions since the assignment $\langle x, b\rangle$ cannot be part of any solution.

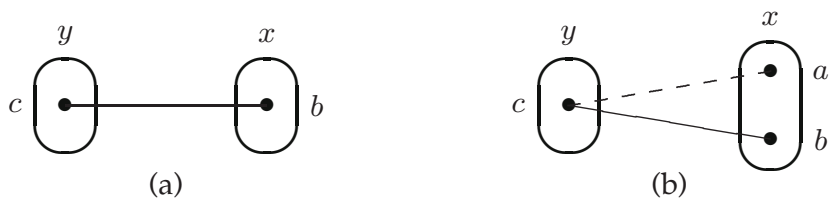

Fig. 1. Illustration of the notions of (a) arc consistency, (b) neighbourhood substitution

In a binary CSP instance we can eliminate a value $b \in \mathcal{D}(x)$ by neighbourhood substitution if $\exists a \in \mathcal{D}(x) \setminus \{b\}$ such that $\forall y \in X \setminus \{x\}$, $\nexists c \in \mathcal{D}(y)$ such that $(a, c) \notin R_{xy}$ and $(b, c) \in R_{xy}$ [7]. This is because in any solution the assignment $\langle x, b\rangle$ can be replaced by the assignment $\langle x, a\rangle$. The corresponding positive property can be defined as follows.

Definition 3. *A value $b \in \mathcal{D}(x)$ is* neighbour-supported *if $\forall a \in \mathcal{D}(x) \setminus \{b\}$, $\exists y \in X \setminus \{x\}$, $\exists c \in \mathcal{D}(y)$ such that $(a, c) \notin R_{xy}$ and $(b, c) \in R_{xy}$. We say that $\langle y, c\rangle$ is a* neighbour-support *of (x, b, a)*

Clearly, an elimination by arc consistency is possible if and only if some variable-value assignment has no AC-support and a neighbourhood substitution elimination is possible if and only if some variable-value assignment is not neighbour-supported. We require the following definition in order to give more general rules for value elimination than arc consistency and neighbourhood substitution.

Definition 4. *A* value-elimination condition *(or simply a* val-elim condition*) is a polytime-computable property $P(x, b)$ of an assignment $\langle x, b\rangle$ in a CSP instance I such that when $P(x, b)$ holds, the instance I' obtained from I by eliminating b from $\mathcal{D}(x)$ is satisfiable if and only if I is satisfiable.*

A val-elim condition allows us to eliminate values from domains while conserving at least one solution (if one exists). In binary CSP, two val-elim conditions on assignment $\langle x, b\rangle$ are: (1) $b \in \mathcal{D}(x)$ is not AC-supported, (2) $b \in \mathcal{D}(x)$ is not neighbour-supported. We now introduce two other notions of support which if not satisfied allow us to eliminate a value from a domain. The first of these is illustrated in Figure 2.

Given a binary CSP instance I, let $I[\langle y, c\rangle]$ denote the instance which results by assigning c to y and by eliminating all values e from other domains $\mathcal{D}(w)$ ($w \in X \setminus \{y\}$) such that $(c, e) \notin R_{yw}$. Suppose that for all possible assignments c to y in I, b is neighbourhood substitutable by some value (not necessarily the same for each value c) in $I[\langle y, c\rangle]$. Then b can be deleted from $\mathcal{D}(x)$ in I without changing the satisfiability of I. This idea is captured by the following positive property of conditional neighbour (CN) support.

Definition 5. *A value $b \in \mathcal{D}(x)$ is* CN-supported *if $\forall y \in X \setminus \{x\}$, $\exists c \in \mathcal{D}(y)$ such that: (1) $(b, c) \in R_{xy}$ and (2) $\forall a \in \mathcal{D}(x) \setminus \{b\}$ with $(a, c) \in R_{xy}$, $\exists z \in X \setminus \{x, y\}$, $\exists d \in \mathcal{D}(z)$ such that $(c, d) \in R_{yz}$, $(b, d) \in R_{xz}$ and $(a, d) \notin R_{xz}$.*

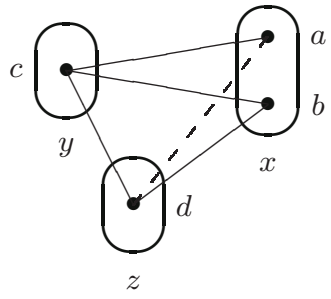

Fig. 2. Illustration of the definition that b is CN-supported

In other words, $b \in \mathcal{D}(x)$ is *CN-supported* if $\forall y \in X \setminus \{x\}$, $\langle x, b\rangle$ has an AC support c at y such that $\forall a \in \mathcal{D}(x) \setminus \{b\}$ with $(a, c) \in R_{xy}$, $\exists z \in X \setminus \{x, y\}$, $\exists d \in \mathcal{D}(z)$ such that $(c, d) \in R_{yz}$ and $\langle z, d\rangle$ is a neighbour-support of (x, b, a). It follows immediately from this definition that a CN-supported assignment is also AC-supported, and (almost immediately) that it is neighbour-supported.

When $\langle y, c\rangle$ is a neighbour-support of (x, b, a), as illustrated in Figure 1(b), it may still be possible to replace b by a in all solutions provided we also replace c by another value d. As we will see in the proof of Proposition 1, below, this is the motivation behind the following notion of extended-neighbour (EN) support, illustrated in Figure 3.

Definition 6. *A value $b \in \mathcal{D}(x)$ is* EN-supported *if $\forall a \in \mathcal{D}(x) \setminus \{b\}$, $\exists y \in X \setminus \{x\}$, $\exists c \in \mathcal{D}(y)$ such that: (1) $(a, c) \notin R_{xy}$, $(b, c) \in R_{xy}$ and (2) $\forall d \in \mathcal{D}(y)$ with $(a, d) \in R_{xy}$, $\exists z \in X \setminus \{x, y\}$, $\exists e, f \in \mathcal{D}(z)$ such that $(a, e) \in R_{xz}$, $(d, e) \notin R_{yz}$, $(c, f) \in R_{yz}$ and $(d, f) \notin R_{yz}$.*

In other words, $b \in \mathcal{D}(x)$ is EN-supported if $\forall a \in \mathcal{D}(x) \setminus \{b\}$, there is a neighbour-support $\langle y, c\rangle$ of (x, b, a) such that condition (2) of Definition 6 holds. It follows that a EN-supported assignment is also neighbour-supported.

Example 1. Suppose that in a binary CSP instance I, there is a subset S of the variables such that each domain $\mathcal{D}(x)$ ($x \in S$) contains a default value 0, where assigning 0 to a variable in S is only possible if we assign 0 to all variables in S, and this partial solution ($s(x) = 0$ for $x \in S$) is compatible with all possible

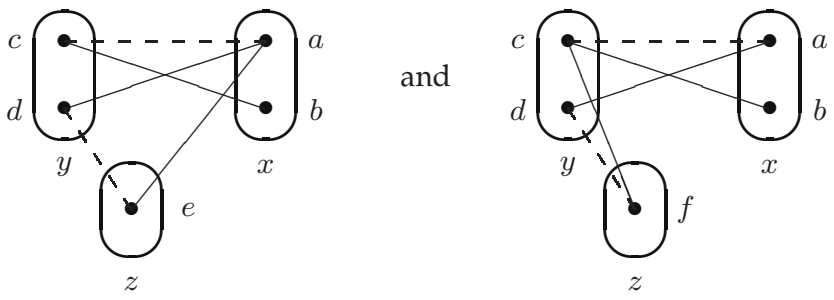

Fig. 3. Illustration of the definition that b is EN-supported

assignments to all variables in $X \setminus S$, i.e. $\forall x, y \in S$, $(0, c) \in R_{xy} \Leftrightarrow c = 0$, and $\forall x \in S, \forall y \notin S$, $(0, c) \in R_{xy}$ for all $c \in \mathcal{D}(y)$. Let b be any value in $\mathcal{D}(x) \setminus \{0\}$, for some $x \in S$. We will show that b is *not* EN-supported. Setting $a = 0$, if $y \in X \setminus \{x\}$ and $c \in \mathcal{D}(y)$ are such that $(a, c) \notin R_{xy}$ and $(b, c) \in R_{xy}$, then we necessarily have $y \in S$. So $\exists d = 0 \in \mathcal{D}(y)$ with $(a, d) = (0, 0) \in R_{xy}$, such that

- $\forall z \in S \setminus \{x, y\}$, $\forall e \in \mathcal{D}(z)$ with $(a, e) \in R_{xz}$, we have $e = 0$ and hence $(d, e) = (0, 0) \in R_{yz}$, and
- $\forall z \in X \setminus S$, $\forall f \in \mathcal{D}(z)$, we have $(d, f) = (0, f) \in R_{yz}$

It follows from Definition 6 that no $b \in \mathcal{D}(x) \setminus \{0\}$ is EN-supported, and hence, anticipating Proposition 1, we can reduce the domains of all variables $x \in S$ to a singleton $\{0\}$ by eliminating all such values b.

We now show that the notions of support given in Definitions 5 and 6 allow us to define val-elim conditions: Proposition 1, below, tells us that assignments with no CN-support or no EN-support can be eliminated while conserving satisfiability. If $b \in \mathcal{D}(x)$ is not neighbour-supported, then it is neither CN-supported nor EN-supported. Thus eliminating values that are not CN-supported or not EN-supported implies eliminating a superset of those values that can be eliminated by neighbourhood substitution.

Proposition 1. *Both of the following conditions on assignment $\langle x, b\rangle$ are val-elim conditions in binary CSP instances:*

1. *$b \in \mathcal{D}(x)$ is not CN-supported.*
2. *$b \in \mathcal{D}(x)$ is not EN-supported.*

Proof. Let I be a binary CSP instance and suppose that s is a solution to I such that $s(x) = b$. In both cases, we will show that I has a solution s' in which $s'(x) \neq b$.

1. Since $b \in \mathcal{D}(x)$ is not CN-supported, $\exists y \in X \setminus \{x\}$ such that $\forall c \in \mathcal{D}(y)$ with $(b, c) \in R_{xy}$, $\exists a(y, c) \in \mathcal{D}(x) \setminus \{b\}$ with $(a(y, c), c) \in R_{xy}$ such that $\forall z \in X \setminus \{x, y\}$, $\forall d \in \mathcal{D}(z)$ with $(c, d) \in R_{yz}$ and $(b, d) \in R_{xz}$, we have $(a(y, c), d) \in R_{xz}$. Define s' to be identical to s, except that $s'(x) = a(y, s(y))$. Now the assignment $\langle x, a(y, s(y))\rangle$ is compatible with $\langle y, s(y)\rangle$ (by definition of $a(y, s(y))$). Furthermore (again by definition of $a(y, s(y))$) the assignment $\langle x, a(y, s(y))\rangle$ is compatible with all assignments which are compatible with both of $\langle x, b\rangle$ and $\langle y, s(y)\rangle$ and hence with all assignments $\langle z, s(z)\rangle$ ($z \in X \setminus \{x, y\}$). It follows that s' is a solution.

   It is worth pointing out that this proof is valid even in the special case in which $X = \{x, y\}$.
2. Since $b \in \mathcal{D}(x)$ is not EN-supported, $\exists a \in \mathcal{D}(x) \setminus \{b\}$ such that $\forall y \in X \setminus \{x\}$, $\forall c \in \mathcal{D}(y)$ with $(a, c) \notin R_{xy}$ and $(b, c) \in R_{xy}$, $\exists d(y, c) \in \mathcal{D}(y)$ with $(a, d(y, c)) \in R_{xy}$, such that $\forall z \in X \setminus \{x, y\}$, either
   (a) $\forall e \in \mathcal{D}(z)$ with $(a, e) \in R_{xz}$, we have $(d(y, c), e) \in R_{yz}$, or
   (b) $\forall f \in \mathcal{D}(z)$ with $(c, f) \in R_{yz}$, we have $(d(y, c), f) \in R_{yz}$

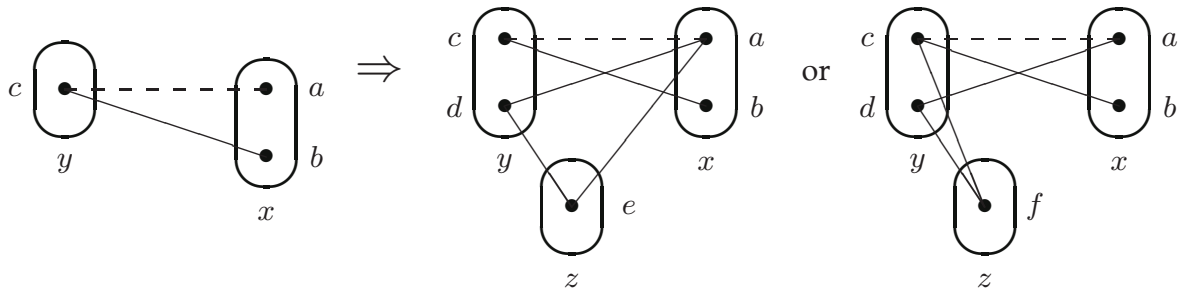

Fig. 4. $b \in \mathcal{D}(x)$ is *not* EN-supported

This is illustrated in Figure 4. Recall that s is a solution such that $s(x) = b$. Let $Y := \{y \in X \mid (a, s(y)) \in R_{xy}\}$ and $\overline{Y} := X \setminus (Y \cup \{x\})$. Define s' as follows

$$s'(v) = \begin{cases} a & \text{if } v = x, \\ s(v) & \text{if } v \in Y, \\ d(v, s(v)) & \text{otherwise.} \end{cases}$$

The assignments $\langle v, s(y)\rangle$ ($v \in Y$) are all compatible with $\langle x, a\rangle$ (by definition of Y) and with each other (since they are all part of the solution s). The assignments $\langle v, d(v, s(v))\rangle$ ($v \in \overline{Y}$) are all compatible with $\langle x, a\rangle$ (by definition of $d(v, s(v))$). Furthermore (again by the definition of $d(v, s(v))$, whether it is (a) or (b) that holds) the assignments $\langle v, d(v, s(v))\rangle$ ($v \in \overline{Y}$) are all compatible with all assignments which are compatible both with $\langle x, a\rangle$ and $\langle v, s(v)\rangle$ and hence with all assignments $\langle w, s(w)\rangle$ for all $w \in Y$ (which are compatible with $\langle x, a\rangle$ by definition of Y and with $\langle v, s(v)\rangle$ since s is a solution). To complete the proof that s' is a solution, it suffices to prove that $\forall v \neq w \in \overline{Y}$, $(d(v, s(v)), d(w, s(w))) \in R_{vw}$. Suppose, for a contradiction, that $(d(v, s(v)), d(w, s(w))) \notin R_{vw}$. Then, when $y = v$, $c = s(v)$ and $z = w$, we necessarily fall into case (b), since setting $e = d(w, s(w))$ contradicts case (a). But then setting $f = d(w, s(w))$ in case (b) implies that $(s(v), d(w, s(w))) \notin R_{vw}$. By a symmetrical argument, exchanging v and w, we immediately have that $(d(v, s(v)), s(w)) \notin R_{vw}$. But applying case (b) to $y = v$, $c = s(v)$, $z = w$ and $f = s(w)$ implies that $(d(v, s(v)), s(w)) \in R_{vw}$. From this contradiction we can deduce that s' is a solution.

It is worth pointing out that this proof is valid even in the special case in which I has only two variables x, y. In this case, either b can be eliminated by neighbourhood substitution, or there is some solution $(a, d(y, c))$ to I which does not require the assignment $\langle x, b\rangle$.

The following two examples show that the two rules given in Proposition 1 allows us to eliminate certain values which are neither arc-inconsistent nor neighbourhood substitutable.

Example 2. Consider the arc-consistent CSP instance I_4 with $X = \{w, x, y, z\}$, $\mathcal{D}(w) = \mathcal{D}(x) = \mathcal{D}(y) = \mathcal{D}(z) = \{1, 2, 3\}$ and five constraints $x \neq y$, $y = z$, $x \neq z$,

$w \neq y$, $w = z$. No eliminations are possible by neighbourhood substitution, but any $b \in \mathcal{D}(x)$ can be eliminated since it is not CN-supported: $\forall c \in \mathcal{D}(y)$, $b \in \mathcal{D}(x)$ is neighbourhood substitutable in $I_4[\langle y, c\rangle]$.

Example 3. Consider the arc-consistent CSP instance with $X = \{x, y, z\}$, $\mathcal{D}(x) = \mathcal{D}(y) = \mathcal{D}(z) = \{1, 2, 3\}$ and three constraints $x \neq y$, $x \neq z$, $(y, z) \notin \{(1, 3), (3, 1)\}$. No eliminations are possible by neighbourhood substitution, but the assignment $b = 2 \in \mathcal{D}(x)$ can be eliminated since it is not EN-supported: this follows from the symmetry between variables y,z and the fact that the value $a = 1 \in \mathcal{D}(x)$ is such that $\forall c \in \mathcal{D}(y)$ with $(a, c) \notin R_{xy}$ and $(b, c) \in R_{xy}$ (i.e. $c = 1$), $\exists d = 2 \in \mathcal{D}(y)$ with $(a, d) \in R_{xy}$, such that $\forall e \in \mathcal{D}(z)$, $(d, e) \in R_{yz}$.

Our two value-elimination rules are complementary since in Example 2, all variable-value assignments are EN-supported and in Example 3, all variable-value assignments are CN-supported.

3 Variable Elimination

In this section we present conditions under which a variable x can be eliminated from a binary CSP instance while preserving satisfiability. A simple example of such a condition is that $\exists a \in \mathcal{D}(x)$ which is compatible with all assignments to all other variables. Another simple example is that the variable x has a singleton domain $\{a\}$. This second example demonstrates that when eliminating the variable x we need to retain the projections onto $X \setminus x$ of all constraints whose scope includes x, since in this example we must first eliminate from all domains $\mathcal{D}(y)$ ($y \neq x$) those values that are not compatible with $\langle x, a\rangle$. Thus, the instance I' obtained by *eliminating a variable* x from a binary CSP instance I is identical to I except that (1) $\forall y \neq x$, we have deleted from $\mathcal{D}(y)$ all values b such that $\langle y, b\rangle$ has no AC-support at x in I, and (2) we have deleted the variable x and all constraints with x in their scope.

As another example, consider the case when an assignment $\langle x, a\rangle$ is such that all other values in the domain of x can be removed one by one, by elimination thanks to one of the val-elim conditions given in Section 2. The variable x can again be eliminated while preserving satisfiability. We can relate this to previous variable-elimination rules as follows. By the above discussion, it is possible to eliminate a variable x when all values $b \in \mathcal{D}(x)$, except for an assignment $\langle x, a\rangle$, are not EN-supported at $\langle x, a\rangle$ (in the sense that there is no neighbour-support $\langle y, c\rangle$ of (x, b, a) which satisfies Condition (2) of Definition 6). This rule strictly subsumes two previously published variable-elimination rules (corresponding to the absence of the existential patterns $\exists$snake or $\exists$invsubBTP in arc-consistent binary CSP instances) [3].

We require the following formal definition in order to give further variable-elimination rules.

Definition 7. *A* satisfiability-preserving variable-elimination condition *(or a* var-elim condition*) is a polytime-computable property $P(x)$ of a variable x in a binary CSP instance I such that when $P(x)$ holds the instance I' obtained from I by*

eliminating x from I is satisfiable if and only if I is satisfiable. Such a property $P(x)$ is a solution-preserving variable-elimination condition *(*sol-var-elim condition*) if it is possible to construct a solution to I from any solution s' to I' in polynomial time.*

A sol-var-elim condition not only allows us to eliminate variables while preserving satisfiability but also allows the polynomial-time recovery of at least one solution to the original instance I from a solution to the reduced instance I'. All the var-elim properties given in this paper are also sol-var-elim properties.

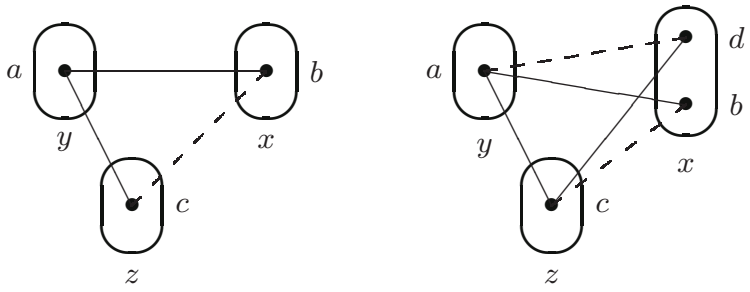

Fig. 5. Illustration of the definition that (a) a variable x is Triangle-supported, (b) a variable x is $\exists\forall$BTP-supported

The following notion of support is illustrated in Figure 5(a). It says that it is not the case that $\exists y \neq x$ such that for all $a \in \mathcal{D}(y)$ to y, in $I[\langle y, a\rangle]$ (the reduced instance consisting of the set of assignments compatible with $\langle y, a\rangle$) there is an assignment $\langle x, b\rangle$ compatible with all assignments to all variables $z \in X \setminus \{x, y\}$.

Definition 8. *A variable x is* Triangle-supported *if $\forall y \in X \setminus \{x\}$, $\exists a \in \mathcal{D}(y)$ such that $\forall b \in \mathcal{D}(x)$ with $(b, a) \in R_{xy}$, $\exists z \in X \setminus \{x, y\}$, $\exists c \in \mathcal{D}(z)$ such that $(a, c) \in R_{yz}$ and $(b, c) \notin R_{xz}$.*

It is known that if for a given variable x in an arc-consistent binary CSP instance I, the set of (in)compatibilities (known as a broken triangle) shown in Figure 5(b) occurs for no two values $b, d \in \mathcal{D}(x)$ and no two assignments a, c to two other variables y, z, then the variable x can be eliminated from I without changing the satisfiability of I [5,3]. The following notion of support, based on the same broken triangle shown in Figure 5(b), leads to a strict generalisation of the broken-triangle property (BTP) variable-elimination rule [5]. We can observe that, unlike BTP, this new rule does not require arc consistency. The corresponding positive property is given by the following definition.

Definition 9. *A variable x is* $\exists\forall$BTP-supported *if $\exists y \in X \setminus \{x\}$, $\exists a \in \mathcal{D}(y)$ such that $\forall b \in \mathcal{D}(x)$ with $(b, a) \in R_{xy}$, $\exists z \in X \setminus \{x, y\}$, $\exists c \in \mathcal{D}(z)$ with $(a, c) \in R_{yz}$ and $(b, c) \notin R_{xz}$, such that $\exists d \in \mathcal{D}(x)$ with $(d, c) \in R_{xz}$ and $(d, a) \notin R_{xy}$.*

The following notion of support is illustrated in Figure 6.

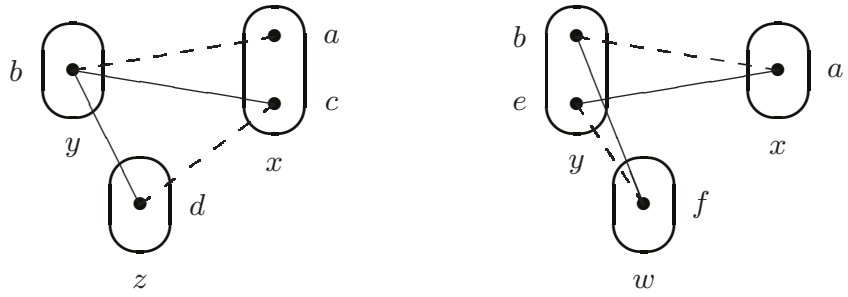

Fig. 6. Illustration of cases (a) and (b) of Definition 10 that variable x is crab-supported

Definition 10. *A variable x is* crab-supported *if $\forall a \in \mathcal{D}(x)$, $\exists y \in X \setminus \{x\}$, $\exists b \in \mathcal{D}(y)$ with $(a,b) \notin R_{xy}$ such that (1) $\forall c \in \mathcal{D}(x)$ with $(c,b) \in R_{xy}$, $\exists z \in X \setminus \{x,y\}$, $\exists d \in \mathcal{D}(z)$ with $(b,d) \in R_{yz}$ and $(c,d) \notin R_{xz}$, and (2) $\forall e \in \mathcal{D}(y)$ with $(a,e) \in R_{xy}$, $\exists w \in X \setminus \{x,y\}$, $\exists f \in \mathcal{D}(w)$ with $(b,f) \in R_{yw}$ and $(e,f) \notin R_{yw}$.*

Proposition 2. *Each of the following properties of variable x are sol-var-elim conditions in binary CSP instances:*

1. *x is not Triangle-supported.*
2. *x is not $\exists\forall$BTP-supported and $\mathcal{D}(x) \neq \emptyset$.*
3. *x is not crab-supported.*

Proof. Let I be a binary CSP instance and suppose that s' is a solution to I', the instance obtained by eliminating variable x from I. In each of the three cases, we will show that I has a solution s. In each case our proof is constructive and there is an obvious polynomial-time algorithm to produce s from s'.

1. Since x is not Triangle-supported, $\exists y \in X \setminus \{x\}$ such that $\forall a \in \mathcal{D}(y)$, $\exists b(a) \in \mathcal{D}(x)$ with $(b(a),a) \in R_{xy}$ such that $\forall z \in X \setminus \{x,y\}$, $\forall c \in \mathcal{D}(z)$ with $(a,c) \in R_{yz}$, we have $(b(a),c) \in R_{xz}$. Define s as follows: $s(v) = s'(v)$ ($v \in X \setminus \{x\}$) and $s(x) = b(s'(y))$. The assignment $\langle x, b(s'(y))\rangle$ is compatible with $\langle y, s'(y)\rangle$ (by definition of $b(s'(y))$) and is compatible with all of the assignments $\langle v, s'(v)\rangle$ ($v \in X \setminus \{x\}$) again by definition of $b(s'(y))$ since $(s'(y), s'(v)) \in R_{yv}$. Hence s is a solution to I.
   It is easily verified that this proof is valid even in the special case $X = \{x,y\}$.
2. If $X = \{x\}$, then since $\mathcal{D}(x) \neq \emptyset$, I has a solution. So from now on we assume that $|X| \geq 2$. Since x is not $\exists\forall$BTP-supported, $\forall y \in X \setminus \{x\}$, $\forall a \in \mathcal{D}(y)$, $\exists b(y,a) \in \mathcal{D}(x)$ with $(b(y,a),a) \in R_{xy}$ such that $\forall z \in X \setminus \{x,y\}$, $\forall c \in \mathcal{D}(z)$ with $(a,c) \in R_{yz}$ and $(b(y,a),c) \notin R_{xz}$, $\forall d \in \mathcal{D}(x)$ with $(d,c) \in R_{xz}$, we have $(d,a) \in R_{xy}$.

   For $v \in X \setminus \{x\}$, let $\mathrm{Im}(v) := \{d \in \mathcal{D}(x) \mid (d, s'(v)) \in R_{xv}\}$. If $y, z \in X \setminus \{x\}$ are such that $(b(y, s'(y)), s'(z)) \notin R_{xz}$, then setting $a = s'(y)$, $c = s'(z)$, we can deduce that $(d, s'(z)) \in R_{xz} \Rightarrow (d, s'(y)) \in R_{xy}$ and hence that $\mathrm{Im}(z) \subseteq \mathrm{Im}(y)$. Indeed, we have $\mathrm{Im}(z) \subset \mathrm{Im}(y)$ since $b(y, s'(y)) \in \mathrm{Im}(y) \setminus \mathrm{Im}(z)$. Now choose some $y \in X \setminus \{x\}$ such that $\mathrm{Im}(y)$ is minimal for inclusion among the

sets $\text{Im}(v)$ ($v \in X \setminus \{x\}$). Then the assignment $\langle x, b(y, s'(y))\rangle$ is compatible with all the assignments $s'(z)$ ($z \in X \setminus \{x, y\}$) (otherwise we would have $\text{Im}(z) \subset \text{Im}(y)$ which would contradict the minimality of $\text{Im}(y)$). Therefore, s is a solution to I, where $s(v) = s'(v)$ ($v \in X \setminus \{x\}$) and $s(x) = b(y, s'(y))$.

3. Since x is not crab-supported, $\exists a \in \mathcal{D}(x)$ such that $\forall y \in X \setminus \{x\}$, $\forall b \in \mathcal{D}(y)$ with $(a, b) \notin R_{xy}$, at least one of the following two conditions holds:
   (a) $\exists c(y, b) \in \mathcal{D}(x)$ with $(c(y, b), b) \in R_{xy}$ such that $\forall z \in X \setminus \{x, y\}$, $\forall d \in \mathcal{D}(z)$ with $(b, d) \in R_{yz}$, we have $(c(y, b), d) \in R_{xz}$.
   (b) $\exists e(y, b) \in \mathcal{D}(y)$ with $(a, e(y, b)) \in R_{xy}$ such that $\forall w \in X \setminus \{x, y\}$, $\forall f \in \mathcal{D}(w)$ with $(b, f) \in R_{yw}$, we have $(e(y, b), f) \in R_{yw}$.

   If $X = \{x\}$, then since $a \in \mathcal{D}(x)$, I has a solution. If $X = \{x, y\}$ (with $y \neq x$), then I has a solution, either of the form $(c(y, b), b)$ or of the form $(a, e(y, b))$. So, from now on we assume that $|X| \geq 3$.

   Let $b \in \mathcal{D}(y)$ be such that $(a, b) \notin R_{xy}$. Let I_1 be identical to I except that we have made $\langle x, a\rangle$ compatible with the assignment $\langle y, b\rangle$. We will show that I_1 is satisfiable iff I is satisfiable. Furthermore, it follows directly from Definition 10 that I_1 is also not crab-supported. It will then follow, by a simple inductive argument, that we can make $\langle x, a\rangle$ compatible with all assignments to all other variables in I without changing its satisfiability. But then we can eliminate x from I since there is an assignment to x which is compatible with all assignments to all other variables.

   Suppose first that $\langle y, b\rangle$ satisfies condition (a), i.e. $\exists c(y, b) \in \mathcal{D}(x)$ with $(c(y, b), b) \in R_{xy}$ such that $\forall z \in X \setminus \{x, y\}$, $\forall d \in \mathcal{D}(z)$ with $(b, d) \in R_{yz}$, we have $(c(y, b), d) \in R_{xz}$. Let I_1 be identical to I except that $(a, b) \in R_{xy}$ in I_1. Suppose that I_1 has a solution s_1 such that $s_1(x) = a$ and $s_1(y) = b$. To show that I and I_1 have the same satisfiability, it suffices to show that I also has a solution. Consider any $z \in X \setminus \{x, y\}$ and let $d = s_1(z)$. Since s_1 is a solution to I_1, $(b, d) \in R_{yz}$. Thus, by condition (a), $(c(y, b), d) \in R_{xz}$. Furthermore, $(c(y, b), b) \in R_{xy}$. Define s by $s(v) = s_1(v)$ ($v \in X \setminus \{x\}$) and $s(x) = c(y, b)$. Then s is a solution to I, since we have just shown that $\langle x, c(y, b)\rangle$ is compatible with $\langle v, s_1(v)\rangle$ for all $v \in X \setminus \{x\}$.

   Suppose now that $\langle y, b\rangle$ satisfies condition (b), i.e. $\exists e(y, b) \in \mathcal{D}(y)$ with $(a, e(y, b)) \in R_{xy}$ such that $\forall w \in X \setminus \{x, y\}$, $\forall f \in \mathcal{D}(w)$ with $(b, f) \in R_{yw}$, we have $(e(y, b), f) \in R_{yw}$. Again, let I_1 be identical to I except that $(a, b) \in R_{xy}$ in I_1. Suppose that I_1 has a solution s_1 such that $s_1(x) = a$ and $s_1(y) = b$. To show that I and I_1 have the same satisfiability, it suffices to show that I also has a solution. Consider any $w \in X \setminus \{x, y\}$ and let $f = s_1(w)$. Since s_1 is a solution to I_1, $(b, f) \in R_{yw}$. Thus by condition (b), $(e(y, b), f) \in R_{yw}$. Furthermore, $(a, e(y, b)) \in R_{xy}$. Define s by $s(v) = s_1(v)$ ($v \in X \setminus \{y\}$) and $s(y) = e(y, b)$. Then s is a solution to I, since we have just shown that $\langle y, e(y, b)\rangle$ is compatible with $\langle v, s_1(v)\rangle$ for all $v \in X \setminus \{y\}$.

The var-elim rule given by Proposition 2(2) subsumes the BTP var-elim rule [5]. Examples of the BTP var-elim rule include a variable x which is only constrained by one other variable in an arc-consistent instance or a Boolean variable x in a path-consistent instance. However, eliminating a variable with no

$\exists\forall$BTP support is strictly stronger then the BTP var-elim rule. This is demonstrated by the fact that it also subsumes the rule that allows us to eliminate a variable x when an assignment to x is compatible with all assignments to all other variables. Another generic example is when all occurrences of the BTP pattern shown in Figure 5(b) on variable x occur on pairs of values $b, d \in S \subset \mathcal{D}(x)$ and each assignment a to each other variable $y \neq x$ has an AC-support at x in $\mathcal{D}(x) \setminus S$.

The var-elim rule given by Proposition 2(3) is a strict generalisation of two previously published var-elim rules (corresponding to the absence of the existential patterns $\exists$subBTP or $\exists$snake in arc-consistent binary CSP instances) [3].

4 Practical Considerations

In binary CSP instances with a large number of variables and/or with large domains, applying the value and variable elimination rules given in this paper may not be practical. Thus, to demonstrate the practical utility of our approach, we now give a weaker version of the notion of EN-support which is nevertheless strictly stronger than the notion of neighbour-support. It leads to a val-elim rule that is strictly stronger than neighbourhood substitution but that can be applied in the same worst-case time complexity [6].

Definition 11. *A value $b \in \mathcal{D}(x)$ is* snake-supported *if $\forall a \in \mathcal{D}(x) \setminus \{b\}$, $\exists y \in X \setminus \{x\}$, $\exists c \in \mathcal{D}(y)$ such that: (1) $(a, c) \notin R_{xy}$, $(b, c) \in R_{xy}$ and (2) $\forall d \in \mathcal{D}(y)$ with $(a, d) \in R_{xy}$, $\exists z \in X \setminus \{x, y\}$, $\exists f \in \mathcal{D}(z)$ such that $(c, f) \in R_{yz}$ and $(d, f) \notin R_{yz}$.*

In other words, $b \in \mathcal{D}(x)$ is snake-supported if $\forall a \in \mathcal{D}(x) \setminus \{b\}$, there is a neighbour-support $\langle y, c\rangle$ of (x, b, a) such that $\forall d \in \mathcal{D}(y)$ with $(a, d) \in R_{xy}$, (y, c, d) has a neighbour-support $\langle z, f\rangle$ for some $z \in X \setminus \{x, y\}$. This is illustrated by the right-hand side of Figure 3. An assignment which is not snake-supported is not EN-supported and hence, by Proposition 1, can be eliminated.

In order to establish and maintain the property that all assignments are snake-supported, we use the following data structures: AC-supps(x,s,y) (for all $x, y \in X$ such that y constrains x and for all $s \in \mathcal{D}(x)$), neighbour-supps(y,p,q), neighbour-supp-vars(y,p,q), diamond-supps(y,p,q), snake-supps(y,p,q) (for all $y \in X$ and for all $p, q \in \mathcal{D}(y)$), neighbour-supps-at(y,p,q,z) (for all $y, z \in X$ such that y constrains z and for all $p, q \in \mathcal{D}(y)$), and hinge-supps(y,p,x,s) (for all $x, y \in X$ such that y constrains x and for all $p \in \mathcal{D}(y)$, $s \in \mathcal{D}(x)$), where

- AC-supps(x,s,y) $= \{q \in \mathcal{D}(y) \mid (s, q) \in R_{xy}\}$
- neighbour-supps(y,p,q) $= \{\langle z, r\rangle \mid r \in \mathcal{D}(z) \wedge (p, r) \in R_{yz} \wedge (q, r) \notin R_{yz}\}$
- neighbour-supps-at(y,p,q,z) $= \{r \in \mathcal{D}(z) \mid \langle z, r\rangle \in$ neighbour-supps(y,p,q)$\}$
- neighbour-supp-vars(y,p,q) $= \{z \in X \mid$ neighbour-supps-at(y,p,q,z) $\neq \emptyset\}$
- diamond-supps(y,p,q) $= \{\langle x, s\rangle \in$ neighbour-supps(y,q,p) $\mid$ $\exists\langle z, r\rangle \in$ neighbour-supps(y,p,q) with $z \neq x$ $\}$
- hinge-supps(y,p,x,s) $= \{q \in \mathcal{D}(y) \setminus \{p\} \mid \langle x, s\rangle \in$ diamond-supps(y,p,q)$\}$
- snake-supps(x,t,s) $= \{\langle y, p\rangle \in$ neighbour-supps(x,t,s) $\mid$ $|$hinge-supps(y,p,x,s)$| = |$AC-supps(x,s,y)$|$ $\}$.

These different notions of support are illustrated in Figure 7: in Figure 7(a), $\langle z, r\rangle \in$ neighbour-supps(y,p,q); in Figure 7(b), $\langle x, s\rangle \in$ diamond-supps(y,p,q) and $q \in$ hinge-supps(y,p,x,s); in Figure 7(c), $\langle y, p\rangle \in$ snake-supps(x,t,s) if $\forall q \in \mathcal{D}(y)$, $q \in$ AC-supps(x,s,y) $\Rightarrow q \in$ hinge-supps(y,p,x,s).

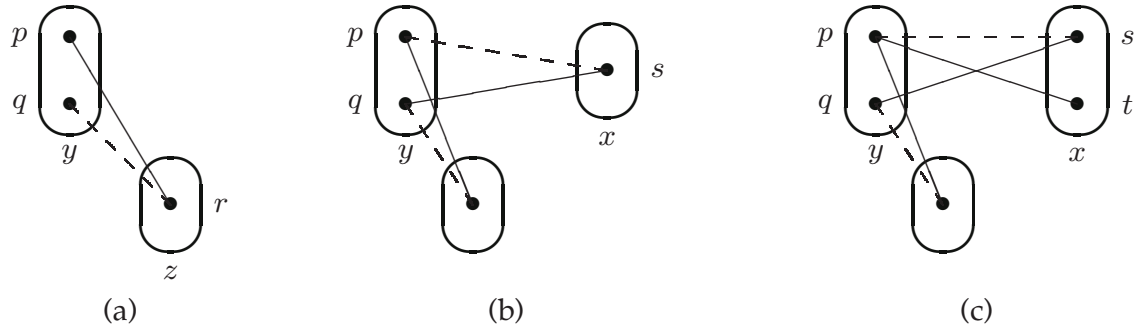

Fig. 7. Illustration of (a) neighbour, (b) diamond and hinge, and (c) snake supports

We can see from Figure 7 and Definition 11, that $t \in \mathcal{D}(x)$ is snake-supported if and only if $\forall s \in \mathcal{D}(x) \setminus \{t\}$, snake-supps($x$,$t$,$s$) $\neq \emptyset$. A value t is therefore deleted from $\mathcal{D}(x)$ when snake-supps(x,t,s) $= \emptyset$ for some $s \in \mathcal{D}(x) \setminus \{t\}$.

Let e be the number of pairs of variables which constrain each other, and let d be the maximum domain size. We can store subsets of a finite set S (such as the set of all variable-value assignments) in the form of a doubly linked list (whose length is also stored) and an array indexed by elements of S and containing pointers to this list. This allows the basic operations of addition, deletion and test of membership and of size to be performed in $O(1)$ time. The six data structures, given above, require $O(ed^3)$ space when stored in this way. We calculate and maintain diamond-supps(y,p,q) using the fact that

$$\text{diamond-supps}(y,p,q) = \text{neighbour-supps}(y,q,p) \quad \text{if } |\text{neighbour-supp-vars}(y,p,q)| > 1,$$

$$\text{diamond-supps}(y,p,q) = \{\langle z, r\rangle \in \text{neighbour-supps}(y,q,p) \mid z \neq x\} \quad \text{if neighbour-supp-vars}(y,p,q) = \{x\}.$$

The above six data structures can be calculated in $O(ed^3)$ from their definitions. Then values $t \in \mathcal{D}(x)$ which are not snake-supported can be eliminated, which may provoke new eliminations. Maintaining the above data structures until convergence (i.e. to the point at which all assignments are snake-supported) can be achieved in $O(ed^3)$ time since assignments can only be deleted and never added to the data structures.

5 Recovering All Solutions

In some applications, it is important to return all solutions to a CSP instance. We therefore study in this section whether it is possible to efficiently recover all

solutions to a binary CSP instance after elimination of variables and/or values by our rules.

Proposition 3. *Let I be a binary CSP instance and let S be the set of all solutions to the instance I' obtained after applying a sequence σ of operations given by the elimination of values that are not CN-supported or the elimination of variables that are not Triangle-supported, or not $\exists\forall$BTP-supported, or not crab-supported. Then the set of all solutions to I can be found from (S, σ) in $O(|S_I|ed + 1)$ time, where S_I is the set of solutions to I.*

Proof. In the trivial case in which $|S_I| = 0$, we necessarily have as input $S = \emptyset$ which can clearly be tested for in $O(1)$ time.

First consider the elimination of a single variable x from an instance I by one of the three variable-elimination rules. As observed in the proof of Proposition 2, each solution of the reduced instance can be extended to a solution of I. This implies that the number of solutions cannot decrease when we reinstate the variable x. Clearly each solution of I is an extension of a solution of the reduced instance. So testing all possible extensions of each solution of the reduced instance will produce all solutions of I in time $O(|S_I|e_x d)$, where e_x is the number of binary constraints with x in their scope.

Now consider the elimination of a value b from the domain of a variable x due to the fact that b is not CN-supported. As observed in the proof of Proposition 1, s is a solution to I with $s(x) = b$ implies that there is a solution s' to the reduced instance I' such that $s'(x) \neq b$ and $s'(v) = s(v)$ for $v \neq x$. To determine all solutions of I including the assignment $\langle x, b\rangle$ from the set of all solutions of the reduced instance thus requires only $O(|S_I|e_x)$ time.

Summing over all variables x and, in the case of value-eliminations, over all assignments to x, we obtain a total time complexity of $O(|S_I|ed+1)$, as claimed.

On the other hand, the following proposition indicates that eliminating values with no snake-support or no EN-support is not useful if we require all solutions. Since a value which is not snake-supported is not EN-supported, we only need to consider the former.

Proposition 4. *Let I be a binary CSP instance and let I' be the instance obtained from I after eliminating all values that are not snake-supported or not arc consistent. Even if we are given the set of all solutions to I', determining whether I has more than one solution is NP-complete.*

Proof. This problem is clearly in NP. It therefore suffices to give a polynomial reduction from the known NP-complete problem binary CSP. Let J be an arbitrary instance of binary CSP on variables X where, without loss of generality, we assume $\forall x \in X, 0 \notin \mathcal{D}(x)$ in J. We build an instance I on variables $X \cup \{x_0\}$ where $x_0 \notin X$ and the domain of variable x_0 in I is $\{0, 1\}$. We add an extra value 0 to each domain $\mathcal{D}(x)$ ($x \in X$). In I, for all variables $y \in X$, the assignment $\langle x_0, 0\rangle$ is compatible only with the assignment $\langle y, 0\rangle$, whereas the assignment $\langle x_0, 1\rangle$ is compatible with all the assignments $\langle y, a\rangle$ for $a \neq 0$; furthermore for each $y, z \in X$, the assignment $\langle y, 0\rangle$ is compatible with all assignments to z.

In I, the value $1 \in \mathcal{D}(x_0)$ is not snake-supported, and hence can be eliminated from the domain of x_0. After establishing arc consistency, all domains are reduced to the singleton $\{0\}$. Hence the reduced instance has exactly one solution. In the instance I, the assignment $\langle x_0, 0\rangle$ only belongs to the solution assigning 0 to each variable, whereas the assignment $\langle x_0, 1\rangle$ is compatible with exactly the set of solutions to the instance J. Thus, determining the existence of a second solution to I is equivalent to determining the satisfiability of J.

6 Theoretical Discussion

We now look into the question of whether there are other rules for the elimination of values or variables (which are not subsumed by known rules or the rules we have given in this paper). To avoid confusion, we use the specific terms CSP-value and CSP-variable to refer to names of values and variables to be quantified. We consider very general rules of the form $Q(A_{var} \cup A_{val})f(E(A))[v]$, where A is a set of variable-value assignments $\langle x, a\rangle$ in which each CSP-value a occurs exactly once, A_{var} (A_{val}) is the set of CSP-variables (CSP-values) occurring in A, $Q(A_{var} \cup A_{val})$ is a sequence of quantifications on $A_{var} \cup A_{val}$, $E(A)$ is the list of the compatibilities of all pairs of assignments from A to two distinct CSP-variables (i.e. the list of truth values of $(a, b) \in R_{xy}$ for each $(\langle x, a\rangle, \langle y, b\rangle) \in A^2$ with $x \neq y$), $f : \{0, 1\}^m \to \{0, 1\}$ is any Boolean function (where $m = |E(A)|$), and v is the CSP-variable or CSP-value which can be eliminated whenever $Q(A_{var} \cup A_{val})f(E(A))$ holds.

For $Q(A_{var} \cup A_{val})f(E(A))[v]$ to be *well-formed* we require that

1. Each CSP-value in A_{val} and each CSP-variable in A_{var} occurs exactly once in $Q(A_{var} \cup A_{val})$,
2. In $Q(A_{var} \cup A_{val})$ each CSP-variable $x \in A_{var}$ is quantified $\exists x \in X \setminus Y$ or $\forall x \in X \setminus Y$ where Y is the set of CSP-variables which has already been quantified (i.e. those CSP-variables appearing to the left of x in $Q(A_{var} \cup A_{val})$),
3. In $Q(A_{var} \cup A_{val})$ each CSP-value a is quantified $\forall a \in \mathcal{D}(x)$ or $\exists a \in \mathcal{D}(x) \setminus H_x$ where x is a CSP-variable which has already been quantified, and H_x is the set of CSP-values which have already been quantified over $\mathcal{D}(x)$,
4. v is a CSP-variable in A_{var} or a CSP-value in A_{val},
5. f is not identically equal to FALSE.

We have chosen to impose that universal quantification of CSP-values be over all values in a domain whereas existential quantification of CSP-values be over all unused values, since all the rules given in this paper can be expressed using this convention. Note that since the various kinds of support (such as neighbour-support, CN-support, etc.) are the negation of the corresponding elimination rule, in the definition of each kind of *support*, existential quantification of CSP-values is over all values in a domain and universal quantification of CSP-values is over all unused values.

Unfortunately, exhaustive search even concerning rules on a small number of CSP-variables and CSP-values rapidly becomes impossible since the number of

Boolean functions on m arguments is 2^{2^m}. Previously, we have studied different forms of forbidden patterns [2,3,4]. Forbidding a *flat pattern* on assignments A corresponds to a rule $Q(A_{var} \cup A_{val})f(E(A))$ where all quantifiers in Q are $\forall$ and the function f is a clause. *Quantified* (respectively, *existential*) *patterns* are of the same form except that the sequence of quantifications Q begins $\exists x \in X$ (respectively, $\exists x \in X$, $\exists a_1 \in \mathcal{D}(x), \ldots, \exists a_r \in \mathcal{D}(x)$) [3]. The rules we consider in this paper are thus much more general in that we allow any (well-formed) sequence of quantifications but also because we allow any Boolean function of the compatibilities.

A valid rule is interesting if it is not too expensive to apply and there is no other rule which both strictly subsumes it and is no more expensive to apply. Not only is the number of cases to consider very large, but the number of interesting var-elim or val-elim rules $Q(A_{var} \cup A_{val})f(E(A))[v]$ could possibly turn out to be very large. It should also be pointed out that certain reduction operations, such as singleton arc consistency, cannot be expressed as local properties.

7 Conclusion

This paper describes several novel reduction operations for binary CSP which are neither based on consistency nor on substitutability. They reduce search space size either by elimination of variables or by the elimination of values. We showed that one of these operations can be applied in the same time complexity as neighbourhood substitution but is strictly stronger. From a practical point of view, further research is required to determine the utility of the rules given in this paper, for example, as preprocessing operations on large-scale real-world instances, or to identify tractable problem domains in which all variables can be eliminated by our variable-elimination rules. From a theoretical point of view, the most interesting challenge is the characterisation of all such rules.

Acknowledgements. I would like to thank David Cohen, Guillaume Escamocher, Peter Jeavons and Stanislav Živný for their insightful comments during discussions concerning this work.

References

1. Cohen, D.A., Jeavons, P., Jefferson, C., Petrie, K.E., Smith, B.M.: Symmetry definitions for constraint satisfaction problems. Constraints 11(2-3), 115–137 (2006)
2. Cohen, D.A., Cooper, M.C., Creed, P., Marx, D., Salamon, A.Z.: The tractability of CSP classes defined by forbidden patterns. J. Artif. Intell. Res. (JAIR) 45, 47–78 (2012)
3. Cohen, D.A., Cooper, M.C., Escamocher, G., Zivny, S.: Variable elimination in binary CSP via forbidden patterns. In: Rossi, F. (ed.) IJCAI, pp. 517–523. AAAI Press, Menlo Park (2013)
4. Cooper, M.C., Escamocher, G.: A dichotomy for 2-constraint forbidden CSP patterns. In: Hoffmann, J., Selman, B. (eds.) AAAI, pp. 464–470. AAAI Press (2012)

5. Cooper, M.C., Jeavons, P.G., Salamon, A.Z.: Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. Artif. Intell. 174(9-10), 570–584 (2010)
6. Cooper, M.C.: Fundamental properties of neighbourhood substitution in constraint satisfaction problems. Artif. Intell. 90(1-2), 1–24 (1997)
7. Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Proceedings of AAAI-91, pp. 227–233 (1991)
8. Gent, I.P., Petrie, K.E., Puget, J.-F.: Symmetry in constraint programming. In: van Beek, P., Walsh, T., Rossi, F. (eds.) Handbook of Constraint Programming, pp. 327–374. Elsevier (2006)
9. Larrosa, J., Dechter, R.: Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. Constraints 8(3), 303–326 (2003)
10. Likitvivatanavong, C., Yap, R.H.C.: Eliminating redundancy in csps through merging and subsumption of domain values. ACM SIGAPP Applied Computing Review 13(2) (2013)
11. Prestwich, S.D.: Full Dynamic Substitutability by SAT Encoding. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 512–526. Springer, Heidelberg (2004)