Theoretical Computer Science 410 (2009) 4769–4783

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Local consistency for extended CSPs

Michael J. Maher*

NICTA, Australia
University of NSW, Australia

ARTICLE INFO

Keywords:
Constraint satisfaction problem
Local consistency
Arc consistency
Pairwise consistency
Constraint programming

ABSTRACT

We extend the framework of Constraint Satisfaction Problems to make it more suitable for/applicable to modern constraint programming languages where both constraint satisfaction and constraint solving have a role. Some rough principles for local consistency conditions in the extended framework are developed, appropriate notions of local consistency are formulated, and relationships between the various consistency conditions are established.

1. Introduction

The study of constraint satisfaction problems (CSPs) has been remarkably successful. The CSP framework provides a flexible basis for formulating NP-complete problems, which can then be addressed by a wide range of methods. The study of systematic search in this framework has yielded techniques for performing search more efficiently and characterized classes of problems for which polynomial time is sufficient to find solutions (see, for example, [16]).

However, the CSP framework also has some weaknesses. In particular, the framework assumes that data values come from a domain of constants where the only pre-defined relations are equality and its negation. This is in contrast to the use of many of the ideas from the study of CSPs in practice. Constraint programming systems routinely use interpreted values – such as integers – use functions – such as addition – and use pre-defined relations – such as ordering or arithmetic relations – to formulate and solve problems.

The complex constraints used in these systems – the global constraints [3] of CHIP [19], the hard constraints [33] of CLP($\Re$) [32], non-linear equations in Real-Paver [25] and Numerica [46], constraints in ILOG Solver [29], the demons written in CHIP or Solver, and predicates defined in Constraint Handling Rules [23], Claire [9], or Oz [47] – all exploit and infer pre-defined relations when solving problems. In some ways these complex constraints are the counterpart of constraints in the CSP framework, and searching for solutions of a query in these systems corresponds to search in CSPs. But there are several aspects of the CSP framework, as generally used, that are not appropriate for these applications.

Traditional treatments of CSPs are not directly applicable to complex constraints because of assumptions and emphases that are not appropriate for complex constraints. In particular, most treatments: consider only constants as values; assume relations are finite and defined extensionally; assume that there is only one constraint on a given set of variables; almost exclusively study binary relations; formulate notions of local consistency that focus on variables and the values that they may take; and do not address the possibility of other relations between variables. As a consequence of the first two points, these treatments cannot include any non-trivial constraint solving. There are several works that recognize and/or avoid some of these assumptions (for example, [41,18,17,48]) but not to the extent that they are applicable to complex constraints.

The aim of this paper is to develop a suitable framework for addressing constraint programming search problems. In the next section, the CSP framework is extended to include a richer class of relations that are solved to global consistency.

* Corresponding address: NICTA, Locked Bag 6016, University of NSW, Sydney, NSW 1466, Australia.
E-mail address: michael.maher@nicta.com.au.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.07.042

(Equivalently, the collection of these relations in a problem is tested for satisfiability.) It represents a synthesis of constraint satisfaction and constraint solving.

The remainder of the paper investigates local consistency in this extended framework. After outlining weaknesses of existing local consistency conditions for constraint programming search problems in Section 3, several are reformulated and generalized for the extended framework. In Section 4 the formulation of arc consistency in [38] is derived from the standard description, following the approach recommended in [37]. This sets the pattern for discussion and reformulation of node consistency (Section 5), pairwise and k-wise consistency (Section 6), and restricted consistencies (Section 8). A k-fold consistency is introduced in Section 7 as an extension of the reformulated arc consistency, and its relationship to other consistency conditions is shown. Singleton consistencies are extended in Section 9. In Section 10 we discuss an orthogonal generalization of local consistencies that involves relaxing the meaning of constraints. A preliminary version of this work was presented in [39].

Before going forward, a choice of terminology is needed, because the areas of constraint satisfaction and constraint solving use the word "constraint" differently. In constraint satisfaction, constraints are the uninterpreted relations that are the basis of the combinatorial problem to be solved. In constraint solving, constraints are the pre-defined relations whose conjunctions can be solved to global consistency through algorithmic techniques. The distinction is not always straightforward in existing work, but in the framework proposed in this paper we will distinguish the two roles played by relations in these two areas. To avoid confusion, we will use "constraint" only to refer to a class of pre-defined relations that are solved to global consistency in a constraint solver. Constraints in the sense of constraint satisfaction will be referred to as "properties". (Thus, in this terminology, CSPs do not involve constraints! Similarly, "global constraints" are not constraints – they are implementations of properties – and, furthermore, are generally not solved to global consistency.)

2. Extending CSPs

This section proposes an extension of the CSP framework that addresses some of the weaknesses identified in the introduction. We begin by recalling the definition of a CSP in our modified terminology.

A *property* over a set of variables V is a pair $(\tilde{x}, \mathcal{P})$ where $\tilde{x}$ is a sequence of n variables in V and $\mathcal{P}$ is a relation of arity n. There is an implicit identification between the variables in the list and the columns/attributes of $\mathcal{P}$. Often this will be written as $\mathcal{P}(\tilde{x})$.

A *Constraint Satisfaction Problem (CSP)* is a 3-tuple $\langle Vars, Prop, D\rangle$ where $Vars$ is a set of variables, $Prop$ is a set of properties over $Vars$, and D maps each $x \in Vars$ to a finite set of constants (the domain of the variable). We can alternatively view D as a conjunction of unary relations, one for each $x \in Vars$. We will use both notations in what follows. A CSP $\mathcal{C} = \langle Vars, Prop, D\rangle$ is a *sub-CSP* of another CSP $\mathcal{C}' = \langle Vars', Prop', D'\rangle$ iff $Vars = Vars'$, $Prop = Prop'$ and $\forall x \in Vars\ D(x) \subseteq D'(x)$. A *binary CSP* is a CSP where all properties are formed from binary relations. Usually in a CSP the relation of each property is defined extensionally, as a set of tuples. A solution to a CSP is a function s mapping each variable to a constant such that for every $x \in Vars$, $s(x) \in D(x)$, and for every $(\tilde{x}, \mathcal{P}) \in Prop$, $s(\tilde{x}) \in \mathcal{P}$.

To extend this framework we need to specify the pre-defined relations (constraints) that are admitted. Following the formulation in constraint logic programming [31], we specify these with a constraint domain.

A *signature* Σ is a set of function and relation symbols, each with an associated arity. We assume that the binary relation symbol $=$ and the relational constants *true* and *false* in Σ.

A *constraint domain* over a signature Σ is a pair $(\mathcal{D}, \mathcal{L})$ where $\mathcal{D}$ is a Σ-structure and $\mathcal{L}$ is a set of logical formulas over Σ and a set of variables. A *constraint* is a formula $c \in \mathcal{L}$. A *primitive constraint* c is a constraint of the form $r(t_1, \ldots, t_n)$ defined by $\mathcal{D}$ where r is a relation symbol and the t_i are terms. If $\tilde{x}$ is the free variables in c, we sometimes write $c(\tilde{x})$. Thus $\mathcal{L}$ specifies the syntax of constraints and $\mathcal{D}$ specifies their semantics. When we need to specify the subclass of constraints with free variables from a set V, we write $\mathcal{L}(V)$. We assume $=$ is interpreted as identity in $\mathcal{D}$, that *true* (*false*) is always (respectively, never) satisfied, and that $\mathcal{L}$ is closed under variable renaming, conjunction, and existential quantification. The *conjunctive language generated by* a set S of constraints is the smallest set that contains all constraints in S, and is closed under conjunction and variable renaming. For notational simplicity, in most of this paper the constraint domain will be left implicit and we will write ψ in place of $\mathcal{D} \models \psi$.

Example 2.1. In finite domain constraint programming, as employed in practice, the constraint domain $\mathcal{FD}$ involves integers and finite domain constraints. The signature Σ contains no function symbols. It contains $=$, *true*, *false* and infinitely many unary relation symbols, one for every finite set of integers.[1] The primitive constraints have the form $x :: S$ where x is a variable and S is a finite set of integers. The language of constraints $\mathcal{L}$ is the conjunctive language generated by the primitive constraints. That is, a constraint is a finite conjunction of primitive constraints. The constraints are interpreted in a structure $\mathbf{Z}_{FD}$ where the values are integers and the primitive constraints $x :: S$ hold only for values v for x such that $v \in S$. Thus $\mathcal{FD}$ is $(\mathbf{Z}_{FD}, \mathcal{L})$. □

[1] In practice, of course, there are only symbols for sets contained in some fixed finite set.

Given a set of variables *Vars* and a constraint domain $(\mathcal{D}, \mathcal{L})$, a *valuation* is a function $v : Vars \to \mathcal{D}$. A valuation on a subset of variables $\tilde{x}$ is the restriction of a valuation to $\tilde{x}$, denoted $v|_{\tilde{x}}$. The *complement* in a constraint c of a valuation v on $\tilde{x}$ is a formula equivalent to $c \wedge \neg \bigwedge_{x \in \tilde{x}} x = v(x)$. We say a constraint domain *admits complementation* of v in c if there is a constraint equivalent to $c \wedge \neg \bigwedge_{x \in \tilde{x}} x = v(x)$. A valuation v satisfies a constraint $c(\tilde{x})$ if $c(v(\tilde{x}))$ evaluates to True in the structure $\mathcal{D}$. v satisfies a property $(\tilde{x}, \mathcal{P})$ if $v(\tilde{x}) \in \mathcal{P}$; if $v(\tilde{x}) = \tilde{a}$ we sometimes write this as $\mathcal{P}(\tilde{a})$.

We now extend the CSP framework, following the philosophy of the CLP scheme [30]. A problem is now parameterized by a constraint domain, and the variable domain D is replaced by an environment of constraints.

Definition 2.1. An *Extended Constraint Satisfaction Problem (ECSP)* with signature Σ is a 4-tuple $\langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ where $(\mathcal{D}, \mathcal{L})$ is a Σ-constraint domain, *Vars* is a set of variables, *Prop* is a set of properties over *Vars*, and C is a constraint from $\mathcal{L}(Vars)$, called the *constraint environment*.[2]

A *solution* of the ECSP is a valuation v that satisfies the constraints C and the properties in *Prop*.

A CSP $\mathcal{C} = \langle Vars, Prop, D\rangle$ can be considered to be an ECSP $\mathcal{E}_{\mathcal{C}} = \langle(\mathcal{D}_{\mathcal{C}}, \mathcal{L}_{\mathcal{C}}), Vars, Prop, C\rangle$ where Σ contains all constants, and unary predicate symbols p_S for each subset S of constants; $\mathcal{L}_{\mathcal{C}}$ is the collection of conjunctions of unary predicates; $\mathcal{D}_{\mathcal{C}}$ is the structure with a domain consisting of the constants in Σ that interprets each $p_S(x)$ as the relation $x \in S$; and C is the conjunction $\bigwedge_{x \in Vars} p_{D(x)}(x)$. We refer to $(\mathcal{D}_{\mathcal{C}}, \mathcal{L}_{\mathcal{C}})$ as a *CSP constraint domain*.

ECSPs are able to represent a larger array of problems than CSPs, and represent them more naturally. We now outline several classes of constraint problems that can be formulated as ECSPs.

In most uses of finite domain constraint programming, the primitive constraints restrict a variable to a finite interval of integers, variables range over integers and lists of integers, and properties are complex constraints such as *cumulative*, *alldifferent*, *element*, as well as arithmetic relations such as $x + y \leq z$.

The problem of finding solutions to non-linear equations and inequalities over the reals [4] can be formulated as an ECSP, where the primitive constraints are floating point bounds on variables (for example, $x \leq f$, where f is a floating point number) over the real numbers, and the properties are the non-linear equations and inequalities.

Similarly, the solving of finite set constraints [24] can be formulated as an ECSP, where additional constraints are containment relations (and, perhaps, other relations [45]) between a set variable and a set (for example, $S \subseteq \{a, b, c, d\}$), the set of values includes the set of finite sets of constants (determined by Σ), and the properties represent relations between set variables, such as $S_1 \subseteq S_2$ or $S_1 \cap S_2 = S_3$.

The above examples involve unary primitive constraints, where solving these constraints to global consistency is straightforward. However, constraints of greater arity, requiring more sophisticated solving techniques, are needed in applications like temporal reasoning (where there may be precedence constraints $x < y$), and to reflect languages like CLP($\Re$), where constraints may contain arbitrarily many variables. CLP($\Re$) queries without user-defined predicates are ECSPs where the constraints are linear arithmetic equations and inequalities over the real numbers, and the properties are non-linear equations and hard constraints like *pow(x, y, z)*.

Finally, the problem of finding solutions in Mixed Integer Linear Programming can be viewed as an ECSP where the constraints are linear inequalities over the real numbers and the properties all have the form *int(x)*, where this asserts that the value of x is an integer.

In all these examples, the underlying values are infinite in number or have internal structure, the properties are not represented extensionally and generally are not binary, and there are pre-defined relations (constraints) that are central to the expression and solution of the problems. Although the CSP framework might theoretically be capable of representing these problems by treating constraints as properties, using possibly infinite relations to represent them, and admitting infinite domains for variables, such a representation would be far removed from the practice of solving these problems.

An ECSP, like a CSP, comes without any specific operational interpretation. However, an abstract execution model is a search tree where each node (except, possibly, the root) is an ECSP satisfying the invariant:

> the constraint environment is satisfiable and the ECSP satisfies a local consistency condition, or the constraint environment is unsatisfiable and the node has no children

Such an execution model requires a method for obtaining local consistency and a constraint solver to test satisfiability. A local consistency condition is a requirement on *Prop* and C that can be decomposed as conjunction of conditions on parts of *Prop* and/or C. We say that a local consistency condition A is *weaker* than another condition B (or B is *stronger* than A) if every ECSP that satisfies B also satisfies A. Two consistency conditions are *incomparable* if neither is weaker than the other.

Many constraint programs are constructed to generate constraints and properties in a first phase, and then search for solutions. The ECSP execution model reflects the second, search, phase. Constraint propagation achieved by implemented properties in that phase corresponds to achieving a form of local consistency in an ECSP. (For example, see [38].) Thus specific local consistency conditions, such as arc consistency, represent specific levels of constraint propagation.

The following technical lemma relates implication within a logical language of properties and constraints (2) with a notion of closure (1) that is a meta-logical statement. It will be used to relate formulations of consistency conditions in CSPs and ECSPs. In the lemma, $\exists_{-\tilde{x}}$ denotes the existential quantification of all variables except $\tilde{x}$, and $\to$ denotes implication.

[2] Note that C may be a conjunction of primitive constraints.

Lemma 2.1. *Let ψ and ϕ be formulas involving both properties and constraints in a constraint domain $\mathcal{D}$, c and c' be constraints, and $\tilde{x}$ be a set of variables. Consider the following statements:*
(1) $(\psi \wedge \phi \wedge c) \rightarrow c'$ *implies* $(\phi \wedge c) \rightarrow c'$
(2) $(\phi \wedge c) \rightarrow \exists_{-\tilde{x}}\ (\psi \wedge \phi \wedge c)$.

1. *If $vars(c') \subseteq \tilde{x}$ and* (2) *holds, then* (1) *holds.*
2. *If $\mathcal{D}$ admits complementation of valuations on $\tilde{x}$ in c, and* (1) *holds for all c' where $vars(c') \subseteq \tilde{x}$, then* (2) *holds.*

Proof. 1. Suppose $(\psi \wedge \phi \wedge c) \rightarrow c'$. Since $vars(c') \subseteq \tilde{x}$, $(\exists_{-\tilde{x}}\ (\psi \wedge \phi \wedge c)) \rightarrow c'$. As a consequence of (2), $\phi \wedge c \rightarrow c'$. That is, (1) holds.

2. We prove the contrapositive. Suppose (2) does not hold. Then there is a valuation for $\tilde{x}$ that can be extended to a valuation v that satisfies $\phi \wedge c$ but not to a valuation that satisfies $\psi \wedge \phi \wedge c$. Let c' be $\exists_{-\tilde{x}}\ c \wedge c_v$, where c_v is the complement of v in c. Then a valuation v' for all variables satisfies $c \rightarrow c'$ iff v' and v are not identical on $\tilde{x}$. Consider any valuation v'. If v' and v are not identical on $\tilde{x}$ then v' satisfies $\psi \wedge \phi \wedge c \rightarrow c'$. If v' and v are identical on $\tilde{x}$ then v' does not satisfy $\psi \wedge \phi \wedge c$ (from the choice of v) and hence v' satisfies $\psi \wedge \phi \wedge c \rightarrow c'$.

Thus $\psi \wedge \phi \wedge c \rightarrow c'$ holds for every valuation v' but $\phi \wedge c \rightarrow c'$ does not hold for v. That is, (1) does not hold for all c'. □

The two statements in the lemma are equivalent for CSPs if we restrict $\tilde{x}$ to a single variable.

Corollary 2.1. *Let $(\mathcal{D}_{\mathcal{C}}, \mathcal{L}_{\mathcal{C}})$ be a CSP constraint domain.*
Let $\tilde{x}$ be a singleton set of variables $\{x\}$, and consider the statements of Lemma 2.1.
Then (2) *holds iff* (1) *holds for all constraints c' with $vars(c') = \{x\}$.*

Proof. It is easy to see that the complement of any valuation on x in a constraint of $\mathcal{L}_{\mathcal{C}}$ is equivalent to a constraint of $\mathcal{L}_{\mathcal{C}}$. Then the corollary follows from Lemma 2.1. □

3. Local consistency

There have been many different local consistency properties proposed over many years. Almost all are formulated – whether explicitly or implicitly – in terms of instantiations of variables and extensions of consistent instantiations.[3]

For example, a large class of local consistency properties is as follows. A CSP is (i, j)-*consistent* [21] if any consistent instantiation of i variables can be extended to a further j variables. k-consistency [20] is $(k-1, 1)$-consistency. In particular, for binary CSPs,[4] arc consistency is $(1, 1)$-consistency and path consistency is equivalent to $(2, 1)$-consistency.

The idea of local consistency has been fruitful in improving systematic search for solving CSPs. However, existing treatments are unsuitable in a number of respects as the basis for systematic search in practical constraint programming languages, particularly those with non-extensional definitions of properties or non-trivial constraints.

The lack of any constraints beyond domain constraints algorithmically solved to global consistency is inherent in the CSP framework. An emphasis on binary CSPs can also be tied to the origins of constraint satisfaction, but it is clear that current practice involves non-binary properties, and encoding these as binary properties seems impractical in general. However several works have addressed the issue of generalizing local consistency conditions to properties of arbitrary arity. Generalized arc consistency [41] applies to properties of arbitrary arity, as do relational consistencies [17]. In addition, pairwise consistency [2] and its extension to k-wise consistency [26], because they were defined originally in a database context, from the beginning were independent of arity, as are later extensions hyper-k-consistency [34] and ω-consistencies [43].

Some consistency conditions, such as path consistency, are enforced by modifying properties. Such an approach is very difficult if properties are not represented extensionally, and expensive in terms of space if they are represented extensionally. The latter point has led to further investigation of domain filtering consistencies [15] such as inverse consistencies [22], restricted path consistencies [7,14] and singleton consistencies [13,44], that only modify variable domains. Since "global constraints" and related properties are often implemented as reactive threads of computation – and not extensionally – it is only such consistency conditions that show promise of widespread applicability to constraint programming search problems.

However, even for these consistency conditions, enforcement algorithms assume that values can be deleted from variable domains. In general, in constraint programming, this assumption is invalid since many constraint programming systems provide constraints that cannot express arbitrary variable domains. This has led to several approximations of arc consistency where domains are replaced by intervals of integers [40], real numbers [6], or finite sets [24], and, more generally, approximation spaces [12]. [49] proposed a variety of interval consistencies, but that proposal does not consider the possibility of constraints other than intervals, and cannot apply to languages like CLP($\Re$).

Finally, although it is clear that both variables and properties are essential to the CSP framework, most local consistency conditions define locality only in terms of variables. For example, (i, j)-consistency addresses various sub-problems (of the main problem) containing $i + j$ variables. As a consequence, many of these consistency conditions are defined in terms of

[3] A consistent instantiation of a set S of variables is an instantiation that satisfies all properties whose variables come only from S.

[4] Under the assumption that there is at most one property on a given pair of variables.

extending a consistent instantiation for a set V of variables. Considering consistent instantiations introduces the effect of an unknown number of unknown properties with variables from V when determining consistency. The (i, j)-consistencies and the relational consistencies of [17] are among those formulated this way. Such consistency conditions are practically impossible to achieve when implementing properties as reactive threads, as is illustrated in [39].

The remainder of this paper formulates local consistency conditions that address the above points. These conditions are: independent of arity; formulated in a manner independent of the particular class of constraints, as closure requirements; and the locality of the conditions is based purely on properties, not on variables. In many cases they are generalizations of consistency conditions for CSPs. These are consistency conditions that have *prima facie* the potential to reflect the behavior of implemented properties.

4. Arc consistency

(Generalized) arc consistency of a property is often formulated as requiring that the instantiation of any variable by a value in the domain of the variable can be extended to an instantiation of all variables that satisfies the property. Originally formulated for binary properties, it was generalized to properties of arbitrary arity [41].

Using logical notation, we can write this as

$$x \in D(x) \rightarrow \exists_{-x} (\mathcal{P} \wedge D)$$

where $\exists_{-x}$ denotes the existential quantification of all variables except x. If we adapt this definition to constraints, instead of variable domains, we obtain:

For every variable x,

$$\exists_{-x} c \rightarrow \exists_{-x} (P \wedge c)$$

This says that every value for x that is consistent with the constraint environment c can be extended to a solution of P and c.

Such a formulation is very close in spirit to the original definition, while generalizing from domains to general constraints, but it considers only one variable at a time. Consequently, the effect of c in this formulation is only to express unary (domain-like) constraints. We can interpret this formulation as saying that if any unary information is available in $P \wedge c$, it is already available in c.

This interpretation provides the basis for the further generalization of arc consistency to a form where variables are less important. We focus on the notion that certain information in $P \wedge c$ is contained wholly within c.

If we apply this idea too readily we reach a definition

$$c \rightarrow (P \wedge c)$$

which simply implies that P is irrelevant. We must restrict the information required to be embedded in c to certain types. We cannot do that directly with the above formulation, so we replace it with a weaker statement.

Definition 4.1. A property $\mathcal{P}$ is *arc consistent with a constraint* c if, for every constraint c',

$$(\mathcal{P} \wedge c) \rightarrow c' \quad \text{implies } c \rightarrow c'$$

An ECSP $\langle (\mathcal{D}, \mathcal{L}), \mathit{Vars}, \mathit{Prop}, C \rangle$ is arc consistent if every $\mathcal{P} \in \mathit{Prop}$ is arc consistent with C.

This relaxes the previous statement by restricting it to information expressible with constraints c'. It formulates the condition as a closure requirement on the constraint environment c. In terms of maintaining arc consistency when $\mathcal{P}$ cannot be modified, this definition implies that all information in $P \wedge c$ that is expressible by constraints must be added to c. Such a formulation emphasizes that enforcing consistency involves expressing information implicit in the problem as explicit constraints.

This definition unifies several existing forms of local consistency, including generalized arc consistency, interval consistency and rule consistency for finite domain languages, and some forms of consistency used for floating point intervals over continuous domains. See [38] for more details. Sound and propagation-complete implementations of properties achieve precisely this level of consistency [38].

In [36], Le Provost and Wallace define the constraint extracted by a propagation agent in a way that is essentially equivalent to Definition 4.1, although formulated differently. That work also employed approximation constraints, which anticipated the use of constraint classes in the more general Definition 4.2.

It might not be clear that the definition of arc consistency for ECSPs is equivalent to arc consistency for CSPs, but this follows from Corollary 2.1.

Proposition 4.1. *Let $\mathcal{C}$ be a CSP and let $\mathcal{E}_\mathcal{C}$ be the equivalent ECSP. Then $\mathcal{C}$ is generalized arc consistent iff $\mathcal{E}_\mathcal{C}$ is arc consistent in the extended sense of Definition 4.1.*

Proof. Suppose the ECSP $\mathcal{E}_{\mathcal{C}} = \langle(\mathcal{D}_{\mathcal{C}}, \mathcal{L}_{\mathcal{C}}), Vars, Prop, C\rangle$ is arc consistent in the extended sense. If $\tilde{x}$ is a single variable x then the complement in C of a valuation v on x is the constraint obtained by replacing $p_S(x)$ by $p_{S-\{v(x)\}}(x)$ in C. (Recall that $p_S(x)$ is the constraint that restricts the possible values of x to S.) Thus part 2 of Lemma 2.1 can be applied for any single variable x. Hence, for every property $\mathcal{P}$ and variable x, $\exists_{-x}\, c \to \exists_{-x}\,(P \wedge c)$. That is, $\mathcal{C}$ is generalized arc consistent.

Suppose the CSP $\mathcal{C} = \langle Vars, Prop, D\rangle$ is generalized arc consistent. For any constraint c', since it is a conjunction of unary primitive constraints, we can write c' as $\wedge_{x\in Vars} c'_x(x)$. Suppose $\mathcal{P} \wedge C \to c'$. Then $\mathcal{P} \wedge C \to c'_x(x)$, for each x. By part 1 of Lemma 2.1 if $\mathcal{P} \wedge C \to c'_x(x)$ then $C \to c'_x(x)$. Since this holds for every x, $C \to c'$. That is, $\mathcal{E}_{\mathcal{C}}$ is arc consistent in the extended sense. □

The study of the *minimum* property in [38] indicates that, in some cases, the definition of arc consistency might be too strong, because c' might involve variables not present in the property. Such a situation can make it difficult to implement the properties to achieve arc consistency. In such cases we can consider a slightly looser form of arc consistency where c' is required to only contain variables appearing in $\mathcal{P}$. It is this looser form that is used in [36]. The above proposition also holds for loose arc consistency.

Similarly, for arc consistency and the other consistencies we will discuss, it may be too computationally expensive to infer *all* constraints c'. However, we can introduce a range of weaker consistencies by limiting the class of constraints that must be represented explicitly in the environment.

Definition 4.2. A property $\mathcal{P}$ is *arc consistent with a constraint c wrt a class of constraints $\mathcal{L}' \subseteq \mathcal{L}$*, if, for every constraint $c' \in \mathcal{L}'$,

$$(\mathcal{P} \wedge c) \to c' \quad \text{implies } c \to c'$$

An ECSP $\langle(\mathcal{D}_{\mathcal{C}}, \mathcal{L}_{\mathcal{C}}), Vars, Prop, C\rangle$ is arc consistent for a class of constraints $\mathcal{L}' \subseteq \mathcal{L}$ if every $\mathcal{P} \in Prop$ is arc consistent with C wrt $\mathcal{L}'$.

An ECSP is arc consistent if it is arc consistent for $\mathcal{L}$.

Although arc consistency for ECSPs is a generalization of generalized arc consistency for CSPs, and some forms of arc consistency (for example, interval consistency) are weaker than generalized arc consistency, it is not true in general that generalized arc consistency is stronger than the extended form of arc consistency.

Example 4.1. Consider the property $\mathcal{P}(x, y, z)$ defined by

$\mathcal{P}$		
1	1	1
1	2	1
2	3	2
3	1	3

and the constraint environment C that states that x, y and z are in the interval $1 \ldots 3$. Then $\mathcal{P}$ is generalized arc consistent wrt the domains of the variables, but it is not arc consistent in the extended sense if the constraint language $\mathcal{L}$ contains equations. This because $\mathcal{P} \wedge C \to x = z$ but $C \not\to x = z$. □

This situation arises because of the existence of a constraint language $\mathcal{L}$ that can express relations that cannot be expressed with domain constraints.

5. Node consistency

Node consistency is the special case of generalized arc consistency where the property involved is unary. Thus it is already covered by the discussion in the previous section. The initial formulation is that

$$C \to \mathcal{P}$$

It is noteworthy that, in an ECSP, a unary property is not necessarily eliminable in the manner that it is in a CSP. The problem is that the property might not be representable by constraints. Consequently, under the above formulation, node consistency often cannot be achieved. For example, let $odd(x)$ refer to the property that holds for odd numbers between 0 and 100, and suppose the constraint language admits only interval constraints. Then the constraints are unable to represent the property.

On the other hand, the formulation as a closure requirement

$$(\mathcal{P} \wedge C) \to c' \quad \text{implies } C \to c'$$

concerns only information representable by constraints, and consequently this relaxed form of node consistency is achievable. Essentially C must contain the tightest outside approximation of $\mathcal{P} \wedge C$ by constraints. In the *odd* example, consistency is achieved when c is $1 \le x \le 99$.

Even so, there are situations where extended node consistency (and, more generally, arc consistency) is not attainable. For example, consider a constraint language over the real numbers permitting only rational bounds (for example, $x \le \frac{1}{2}$), and the property $x \le \sqrt{2}$. Since there is no least rational upper bound for $\sqrt{2}$, node consistency cannot be achieved for this property and constraint domain. On the other hand, this problem does not arise for floating point bounds since there is a least floating point upper bound of $\sqrt{2}$.

6. *k*-wise consistency

A collection of properties is *pairwise consistent* [2] if, for every pair of properties $\mathcal{P}_1$ and $\mathcal{P}_2$ in the collection

$$(\exists_{-vars(\mathcal{P}_i)}\ \mathcal{P}_1 \wedge \mathcal{P}_2) \leftrightarrow \mathcal{P}_i \quad \text{for } i = 1, 2$$

This condition means that $\mathcal{P}_1$ and $\mathcal{P}_2$ are essentially independent — that the presence of $\mathcal{P}_2$ does not eliminate any solutions of $\mathcal{P}_1$, and *vice versa*. It was proposed in the context of relational databases with no domains or constraints on variables, and it can be maintained through the use of semi-joins. Adapting it for ECSPs (and CSPs), where we look for consistency with respect to the constraint environment C (respectively, the variable domains), we have, for $i = 1, 2$:

$$(\exists_{-vars(\mathcal{P}_i)}\ \mathcal{P}_1 \wedge \mathcal{P}_2 \wedge C) \leftrightarrow (\exists_{-vars(\mathcal{P}_i)}\ \mathcal{P}_i \wedge C)$$

which only requires that $\mathcal{P}_1$ and $\mathcal{P}_2$ are independent on solutions of C.

As with arc consistency, we give a relaxed formulation that makes the closure requirement explicit:

For $i = 1, 2$ and constraints c' with $vars(c') \subseteq vars(\mathcal{P}_i)$

$$(\mathcal{P}_1 \wedge \mathcal{P}_2 \wedge C) \rightarrow c' \quad \text{implies } (\mathcal{P}_i \wedge C) \rightarrow c'$$

Pairwise consistency was generalized to k-wise consistency [26] which, after adapting it to CSPs and ECSPs, requires that for every subset $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ of properties in the collection and for $i = 1, \ldots, k$

$$\left(\exists_{-vars(\mathcal{P}_i)} \bigwedge_{j=1}^{k} \mathcal{P}_j \wedge C\right) \leftrightarrow \exists_{-vars(\mathcal{P}_i)}\ (\mathcal{P}_i \wedge C)$$

After relaxation this definition becomes:

Definition 6.1. An ECSP $\langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ is *k-wise consistent* if, for every $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\} \subseteq Prop$ of size k, for $i = 1, \ldots, k$ and all constraints c' with $vars(c') \subseteq vars(\mathcal{P}_i)$

$$\left(\bigwedge_{j=1}^{k} \mathcal{P}_j \wedge C\right) \rightarrow c' \quad \text{implies } (\mathcal{P}_i \wedge C) \rightarrow c'$$

Obviously 1-wise consistency is trivial. Using Lemma 2.1 we find that, for $k \geq 2$, this formulation of k-wise consistency is strictly weaker than the adapted formulation for CSPs.

Proposition 6.1. *Let $\mathcal{C}$ be a CSP and let $\mathcal{E}_\mathcal{C}$ be the equivalent ECSP. If $\mathcal{C}$ is k-wise consistent then $\mathcal{E}_\mathcal{C}$ is k-wise consistent in the extended sense above. But $\mathcal{E}_\mathcal{C}$ might be k-wise consistent in the extended sense when $\mathcal{C}$ is not k-wise consistent, for $k \geq 2$.*

Proof. Suppose the CSP $\mathcal{C} = \langle Vars, Prop, D\rangle$ is k-wise consistent. Let C be $\bigwedge_{x \in Vars} p_{D(x)}(x)$. For any constraint c', since it is a conjunction of unary primitive constraints, we can write c' as $\wedge_{x \in Vars} c'_x(x)$. Suppose $\bigwedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow c'$. Then $\bigwedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow c'_x(x)$, for every x. By part 1 of Lemma 2.1 if $\bigwedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow c'_x(x)$ then $\mathcal{P}_i \wedge C \rightarrow c'_x(x)$. Since this holds for every x, $\mathcal{P}_i \wedge C \rightarrow c'$. This argument holds for each $i = 1, \ldots, k$ and for any k properties $\mathcal{P}_1, \ldots, \mathcal{P}_k$ in *Prop*, so $\mathcal{E}_\mathcal{C}$ is k-wise consistent in the extended sense.

For the second half, consider the CSP $\mathcal{C}$ with properties $\mathcal{P}_1(x, y)$ and $\mathcal{P}_2(x, y)$ defined by

$\mathcal{P}_1$	
a	b
b	a
b	b

$\mathcal{P}_2$	
a	b
b	a

and a domain of $\{a, b\}$ for the variables x and y. Then $\mathcal{E}_\mathcal{C}$ is pairwise consistent in the extended sense but $(\mathcal{P}_1 \wedge \mathcal{P}_2) \leftarrow \mathcal{P}_1$ does not hold. Hence $\mathcal{C}$ is not pairwise consistent.

This argument readily extends to any $k > 2$, using $k - 2$ additional properties not sharing any variables with any other property. □

k-wise consistency is incomparable with arc consistency, in general. For example, a CSP with properties that have disjoint variables is 2-wise consistent even when it is not arc consistent. Conversely, the following example shows a CSP that is arc consistent but not 2-wise consistent.

Example 6.1. Consider the CSP $\mathcal{C}$ with properties $\mathcal{P}_1(w, x, y)$ and $\mathcal{P}_2(x, y, z)$ defined by

$\mathcal{P}_1$		
a	b	c
b	c	a
c	a	b

$\mathcal{P}_2$		
c	b	a
a	c	b
b	a	c

and a domain of $\{a, b, c\}$ for the variables w, x, y and z. Then $\mathcal{E}_\mathcal{C}$ is clearly arc consistent but not 2-wise consistent because $(\mathcal{P}_1 \wedge \mathcal{P}_2) \rightarrow$ *false* while neither property on its own causes the inconsistency. □

7. *k*-fold consistency

The natural generalization of extended arc consistency is to consider more than one property at a time.

Definition 7.1. An ECSP $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ is *k-fold consistent* if, for every $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\} \subseteq Prop$ of size k, and every constraint c'

$$\bigwedge_{i=1}^{k} \mathcal{P}_i \wedge C \to c' \quad \text{implies } C \to c'$$

Thus 1-fold consistency is extended arc consistency. As with arc consistency, we can also consider the looser form, where each variable in c' also occurs in a property. k-fold consistency, like the previous local consistency conditions that we have discussed, defines locality purely in terms of properties, and not in terms of variables.

Relational m-consistency [17] has some similarity to k-wise and k-fold consistency (when $m = k$), but also involves variable-based locality. Relational m-consistency requires that for every set of m properties and every subset $\tilde{y}$ of variables used in those properties every solution to the sub-CSP defined by $\tilde{y}$ can be extended to solve all m properties.

Extending the definition to ECSPs, an ECSP $\langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ is *relationally m-consistent* if, for every $\{\mathcal{P}_1(\tilde{x}_1), \mathcal{P}_2(\tilde{x}_2), \ldots, \mathcal{P}_m(\tilde{x}_m)\} \subseteq Prop$ of size m, for every set of variables $\tilde{y}$ such that $\tilde{y} \subseteq \cup_{i=1}^{m} \tilde{x}_i$

$$\left(\bigwedge_{vars(\mathcal{P})\subseteq\tilde{y}} \mathcal{P} \wedge C\right) \to \exists_{-\tilde{y}} \bigwedge_{i=1}^{m} \mathcal{P}_i \wedge C$$

Weakening with Lemma 2.1, this condition becomes: for all constraints c' with $vars(c') \subseteq vars(\mathcal{P}_i)$

$$\bigwedge_{i=1}^{m} (\mathcal{P}_i \wedge C) \to c' \quad \text{implies} \quad \bigwedge_{vars(\mathcal{P})\subseteq\tilde{y}} \mathcal{P} \wedge C \to c'$$

When $\tilde{y}$ is the empty set this reduces to m-fold consistency. Thus relational m-consistency is stronger than m-fold consistency. In particular, when $m = 1$ we have that relational 1-consistency (also called relational arc consistency in [17]) is stronger than extended arc consistency. Similarly, taking $\tilde{y}$ to be $\tilde{x}_i$ for each i, we find that relational m-consistency is stronger than m-wise consistency whenever we have, for all i and j, $\tilde{x}_i \not\subseteq \tilde{x}_j$.

k-fold consistency is also closely related to conjunctive consistency [8], once we extend that notion to ECSPs. Let a covering of an ECSP $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ be a set of subsets of *Prop* such that every property of *Prop* occurs in at least one subset. Conjunctive consistency wrt a covering requires that, for each subset, the conjunction of properties in the subset, when considered as a single property, is arc consistent. k-fold consistency is conjunctive consistency with respect to a covering that contains exactly subsets of *Prop* of cardinality k. The proof follows directly from the definitions.

Proposition 7.1. *Let $\mathcal{C} = \langle Vars, Prop, D\rangle$ be a CSP containing at least k properties, and let $\mathcal{E}$ be the equivalent ECSP. Let S_k be the set of all subsets of Prop of cardinality k. Then*

$\mathcal{E}$ is k-fold consistent iff $\mathcal{C}$ is conjunctive consistent wrt S_k.

It is not surprising that k-fold consistency, for the various values of k, forms a hierarchy of local consistency conditions. This also holds for the looser form where the variables of c' must occur in a property.

Proposition 7.2. *Let $\mathcal{E}$ be an ECSP containing at least k properties.*

If $\mathcal{E}$ is k-fold consistent, then $\mathcal{E}$ is $(k-1)$-fold consistent. However, the converse is not true. That is, $\mathcal{E}$ might be $(k-1)$-fold consistent, but not k-fold consistent.

Proof. Since any set X of size $k-1$ is a subset of some set Y of size k, if $\wedge_{i\in X} \mathcal{P}_i \wedge c \to c'$ then $\wedge_{i\in Y} \mathcal{P}_i \wedge c \to c'$. If $\mathcal{E}$ is k-fold consistent, then $c \to c'$. Thus $\mathcal{E}$ is $(k-1)$-fold consistent.

To show the converse is not true, consider the cyclic graph with k vertices and k edges (that is, the k-ring). Let the vertices be $a_0, a_1, \ldots, a_{k-1}$, and let e be the relation with tuples $\langle a_i, a_{(i+1) \bmod k}\rangle$ for each $i \in \{0, 1, \ldots, k-1\}$. We take this as a constraint domain, where $\Sigma = \{=, e, a_0, a_1, \ldots, a_{k-1}\}$, $\mathcal{D}_k$ is the Σ-structure determined by the cyclic graph, and $\mathcal{L}$ is the conjunctive language generated by Σ.

Let $\mathcal{E} = \langle(\mathcal{D}_k, \mathcal{L}), Vars, Prop, C\rangle$ be an ECSP where $Vars = \{x_1, \ldots, x_{k+1}\}$, *Prop* is the set of properties $\{e(x_1, x_2), e(x_2, x_3), \ldots, e(x_{k-1}, x_k), e(x_k, x_{k+1})\}$, and C is *true*.

Then $\mathcal{E}$ is not k-fold consistent, since the conjunction of properties implies $x_1 = x_{k+1}$, but this is not reflected in C. However, for any subset of $k-1$ properties no further inference can be drawn, so $\mathcal{E}$ is $(k-1)$-fold consistent. □

We now turn our attention to the relationship between k-fold consistency and k-wise consistency.

The *relational structure* of an ECSP $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ is a pair $\langle Vars, \{\tilde{x} \mid \exists \mathcal{P}\ \mathcal{P}(\tilde{x}) \in Prop\}\rangle$. Let $\mathcal{E}$ be an ECSP and let $\langle Vars, \{\tilde{x}_1, \ldots, \tilde{x}_m\}\rangle$ be the relational structure of $\mathcal{E}$. Let $\tilde{x}_0 = Vars - \cup_{i=1}^{m} \tilde{x}_i$. We say that $\mathcal{E}$ has *lossless decomposition of constraints* if, for every constraint $c \in \mathcal{L}$,

$$c \leftrightarrow \bigwedge_{i=0}^{m} \exists_{-\tilde{x}_i} c$$

Lossless decomposition of constraints is similar to lossless joins and join dependencies in relational database theory [35]. As there, the relational structure is determined by properties (relations), but here the lossless decomposition applies to the constraint environment, rather than a universal relation.

It is straightforward to show that any conjunction of unary constraints has a lossless decomposition, independent of the relational structure. We use this fact and the next lemma to prove the following proposition.

Lemma 7.1. *Suppose an ECSP $\mathcal{E}$ has lossless decomposition of constraints and is arc consistent and k-wise consistent. Then $\mathcal{E}$ is k-fold consistent.*

Proof. Let $\mathcal{P}_1, \ldots, \mathcal{P}_m$ be the properties of $\mathcal{E}$ and let $\tilde{x}_i$ be the variables of $\mathcal{P}_i$. For any constraint c', let c'_i denote $\exists_{-\tilde{x}_i} c'$. For any k properties $\mathcal{P}_1, \ldots, \mathcal{P}_k$ we can reason as follows. If $\wedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow c'$ then $\wedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow c'_i$, for $i = 0, \ldots, m$. By k-wise consistency, $\mathcal{P}_i \wedge C \rightarrow c'_i$ and hence, by arc consistency, $C \rightarrow c'_i$, for each $i > 0$. Since $\wedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow c'_0$ we have $(\exists_{-\tilde{x}_0}\ \wedge_{j=1}^{k} \mathcal{P}_j \wedge C) \wedge \exists_{-\tilde{x}_0} C \rightarrow c'_0$. If $\wedge_{j=1}^{k} \mathcal{P}_j \wedge C$ is satisfiable then $C \rightarrow c'_0$. By lossless decomposition of constraints, applied to c', $C \rightarrow c'$. If $\wedge_{j=1}^{k} \mathcal{P}_j \wedge C$ is not satisfiable then $\wedge_{j=1}^{k} \mathcal{P}_j \wedge C \rightarrow$ *false*. By the same argument as above, we must have $C \rightarrow$ *false* and hence $C \rightarrow c'$, for any c'. Thus $\mathcal{E}$ is k-fold consistent. □

We now show that k-fold consistency is a stronger condition than k-wise consistency, but in a practical sense it is not much stronger.

Proposition 7.3. *Let $\mathcal{E}$ be an ECSP.*

1. *If $\mathcal{E}$ is k-fold consistent then $\mathcal{E}$ is k-wise consistent.*
2. *Suppose $\mathcal{L}$ is generated conjunctively from unary constraints. If $\mathcal{E}$ is k-wise consistent and arc consistent then $\mathcal{E}$ is k-fold consistent.*

Proof. Consider $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$.

1. Suppose $(\bigwedge_{j=1}^{k} \mathcal{P}_j \wedge C) \rightarrow c'$. By k-fold consistency, $C \rightarrow c'$. Hence, $(\mathcal{P}_i \wedge C) \rightarrow c'$.

2. Every conjunction c of unary constraints can be written in the form $\bigwedge_{x \in Vars} c_x(x)$ where $c_x(x)$ is the conjunction of all unary constraints in c involving x. For any set of variables $\tilde{x}$, $\exists_{-\tilde{x}} c \leftrightarrow \bigwedge_{x \in \tilde{x}} c_x(x)$. If $Vars = \cup_{i=0}^{m} \tilde{x}_i$ then

$$\begin{aligned} C &\leftrightarrow \bigwedge_{x \in Vars} C_x(x) \\ &\leftrightarrow \bigwedge_{i=0}^{m} \bigwedge_{x \in \tilde{x}_i} C_x(x) \\ &\leftrightarrow \bigwedge_{i=0}^{m} \exists_{-\tilde{x}_i} C \end{aligned}$$

Thus $\mathcal{E}$ has lossless decomposition of constraints.

Hence, by Lemma 7.1, $\mathcal{E}$ is k-fold consistent. □

In many settings of interest – CSPs, finite integer domains, finite sets, and interval approaches to continuous domains – the primitive constraints are unary and arc consistency is maintained. Thus, in these settings there is no practical difference between k-fold consistency and k-wise consistency. On the other hand, when addressing constraint solvers for non-unary constraints, as in CLP($\Re$) or in temporal reasoning, the combination of k-wise consistency and arc consistency is not as strong as k-fold consistency.

Example 7.1. Consider a linearly ordered set, where the only constraint relations are $<$ and $\leq$. For concreteness we choose the integers.

Consider the two properties $\mathcal{P}_1(x, y)$ and $\mathcal{P}_2(y, z)$ with relations defined as follows:

$\mathcal{P}_1$	
0	4
1	5
2	9

$\mathcal{P}_2$	
4	1
5	3
9	6

Consider an ECSP $\mathcal{E}$ where the properties are $\mathcal{P}_1$ and $\mathcal{P}_2$ and the constraint environment c is $0 \leq x \leq 2 \wedge 4 \leq y \leq 9 \wedge 1 \leq z \leq 6 \wedge x < y \wedge z < y$.

Then $\mathcal{E}$ is arc consistent, since neither property implies any stronger constraint. $\mathcal{E}$ is 2-wise consistent because $(\exists z\ \mathcal{P}_1 \wedge \mathcal{P}_2 \wedge c) \leftrightarrow (\exists z\ \mathcal{P}_1 \wedge c)$ and $(\exists x\ \mathcal{P}_1 \wedge \mathcal{P}_2 \wedge c) \leftrightarrow (\exists x\ \mathcal{P}_2 \wedge c)$. But $\mathcal{E}$ is not 2-fold consistent because $\mathcal{P}_1(x, y) \wedge \mathcal{P}_2(y, z) \rightarrow x < z$. □

A variation of this example applies to partially ordered sets with 3 separate linearly ordered subsets of length 2 and 3 other elements. Another variation applies to constraint domains with at least 3 elements and only $=$ and $\neq$ as primitive constraints. A somewhat different example makes the same point when the constraint language permits linear inequalities, as in CLP($\Re$).

Example 7.2. Consider the two properties $\mathcal{P}_1(x, y)$ and $\mathcal{P}_2(y, z)$ with relations defined as follows:

$\mathcal{P}_1$	
0	99
10	10
99	0

$\mathcal{P}_2$	
99	99
10	20
0	99

Suppose these are the properties in an ECSP $\mathcal{E}$ over the real numbers with linear equations and inequalities as constraints. Let the constraint environment C be $C_1 \wedge C_2$, where C_1 is

$$x + y \leq 99 \wedge 89x + 10y \geq 990 \wedge 10x + 89y \geq 990$$

and C_2 is

$$z \leq 99 \wedge 79y + 10z \geq 990 \wedge 79y - 89z \leq -990$$

Then $\mathcal{E}$ is arc consistent, since C_1 (C_2) expresses the convex closure of $\mathcal{P}_1(x, y)$ (respectively $\mathcal{P}_2(y, z)$). Furthermore, $\mathcal{E}$ is 2-wise consistent. Informally, this is because the conjunction of $\mathcal{P}_1(x, y)$ and $\mathcal{P}_2(y, z)$ does not eliminate any tuples from $\mathcal{P}_1$ or $\mathcal{P}_2$. A detailed argument is left to the reader.

But $\mathcal{E}$ is not 2-fold consistent: $x+y = z$ is clearly a consequence of $\mathcal{P}_1(x, y) \wedge \mathcal{P}_2(y, z)$, but C does not imply $x+y = z$. □

Currently it is unclear how to systematically separate k-fold consistency from k-wise and arc consistency. Constraint domains that are sufficiently expressive can represent finite relations as constraints, so that all these consistencies reduce to constraint satisfiability in such constraint domains. But for less expressive constraints and/or infinite properties separation remains unclear.

2-fold consistency is a promising alternative to path consistency when a stronger consistency than solely arc consistency is desired. It has been little investigated,[5] perhaps because it reduces to arc consistency on binary CSPs, under the assumption that there is at most one property between each pair of variables. However, in other constraint domains it is a local consistency stronger than arc consistency and incomparable to path consistency [39].

Proposition 7.4. *Let $\mathcal{C}$ be a binary CSP with at most one property between each pair of variables, and let $\mathcal{E}$ be the corresponding ECSP. Then $\mathcal{E}$ is 2-fold consistent iff $\mathcal{C}$ is arc consistent.*

Proof. By Proposition 4.1, $\mathcal{C}$ is arc consistent iff $\mathcal{E}$ is arc consistent. By Proposition 7.2, and the observation that arc consistency is 1-fold consistency, if $\mathcal{E}$ is 2-fold consistent then $\mathcal{E}$ is arc consistent. To address the reverse direction, suppose $\mathcal{E}$ is arc consistent but not 2-fold consistent. Then there are properties $\mathcal{P}_1(x, y)$, $\mathcal{P}_2(y, z) \in \mathit{Prop}$ and a constraint c such that $\mathcal{P}_1(x, y) \wedge \mathcal{P}_2(y, z) \wedge C \rightarrow c$ but $C \not\rightarrow c$. Since $\mathcal{E}$ is arc consistent we must have $\mathcal{P}_1(x, y) \wedge C \not\rightarrow c$ and $\mathcal{P}_2(y, z) \wedge C \not\rightarrow c$. Without loss of generality we can assume that c is a constraint on a single variable.

If c is a constraint on y then there must be a valuation $x = a, y = b$ satisfying $\mathcal{P}_1(x, y) \wedge C \wedge \neg c$ and a valuation $y = b, z = d$ satisfying $\mathcal{P}_2(y, z) \wedge C \wedge \neg c$. But then $x = a, y = b, z = d$ is a valuation satisfying $\mathcal{P}_1(x, y) \wedge \mathcal{P}_2(y, z) \wedge C \wedge \neg c$, contradicting the claim that c is a witness to lack of 2-fold consistency.

If c is a constraint on x then there must be a valuation $x = a, y = b$ satisfying $\mathcal{P}_1(x, y) \wedge C \wedge \neg c$. Now, if there is a valuation $y = b, z = d$ satisfying $\mathcal{P}_2(y, z) \wedge C \wedge \neg c$ then, as above, we contradict the claim that c is a witness to lack of 2-fold consistency. However, if $y = b$ cannot be extended to a solution of $\mathcal{P}_2(y, z) \wedge C$ then $\mathcal{E}$ is not arc consistent, contradicting our initial supposition.

If c is a constraint on z the argument is symmetric with the one where c is a constraint on x. Hence, if $\mathcal{E}$ is arc consistent it is also 2-fold consistent. □

When there may be more than one property on two variables (as can occur in non-binary CSPs or non-normalized binary CSPs) there is a distinction between arc consistency and 2-fold consistency, as Example 6.1 demonstrates.

[5] Although the finite domain constraint solvers discussed in [27] can be viewed as partial implementations of 2-fold consistency, and we have already noted that k-fold consistency is closely related to conjunctive consistency.

8. Restricted consistencies

Restricted path consistency (RPC) [7] was designed to strengthen arc consistency towards path consistency on binary CSPs without requiring the deletion of tuples from properties. It does this by performing a path consistency check only when a discovery of inconsistency allows a value to be deleted from a variable domain.

Consider a binary CSP $\mathcal{C} = \langle Vars, Prop, D\rangle$. A tuple $x = a, y = b$ of $\mathcal{P}_0(x, y)$ is path consistent [42] if for every variable z, $x = a, y = b$ can be extended to satisfy all properties on x, y, z. In this case we say that $y = b$ is a *path consistent support* for $x = a$ in $\mathcal{P}_0(x, y)$ and, vice versa, $x = a$ is a path consistent support for $y = b$. $\mathcal{C}$ is *path consistent* if every tuple of every property is path consistent. $\mathcal{C}$ is *restricted path consistent* if it is arc consistent and, for every variable x and property in *Prop* including x (say $\mathcal{P}_0(x, y)$) and for every value $a \in D(x)$ such that there is a unique $b \in D(y)$ with $\mathcal{P}_0(a, b)$, the tuple $x = a, y = b$ is path consistent. (Instead of requiring that $x = a, y = b$ is path consistent we could, equivalently, require that $x = a$ has a path consistent support.) These definitions vary somewhat from the original by quantifying over properties, but reduce to the original definitions when we assume a unique property between any two variables.

We also can formulate the definition of path consistency in logic as follows: for every tuple in every property $\mathcal{P}_0(x, y)$, and every variable z and corresponding properties $\mathcal{P}_1(x, z)$ and $\mathcal{P}_2(y, z)$

$$(\exists_{-x,y}\ \mathcal{P}_0(x, y) \wedge D) \rightarrow \exists_{-x,y}\ (P_0(x, y) \wedge P_1(x, z) \wedge P_2(y, z) \wedge D)$$

There are many ways in which the idea of path consistency might be extended to properties of greater arity and to ECSPs, for example, relational (2, 3)-consistency and relational path consistency [17]. Here we will consider only one:

For all properties $\mathcal{P}_0(\tilde{x})$, $\mathcal{P}_1(\tilde{y})$, $\mathcal{P}_2(\tilde{z})$ such that $\tilde{x} \cap \tilde{y} \neq \emptyset$, $\tilde{x} \cap \tilde{z} \neq \emptyset$, and $(\tilde{y} \cap \tilde{z}) - \tilde{x} \neq \emptyset$:

$$\left(\exists_{-\tilde{x}}\ \mathcal{P}_0(\tilde{x}) \wedge C\right) \rightarrow \exists_{-\tilde{x}}\ (P_0(\tilde{x}) \wedge P_1(\tilde{y}) \wedge P_2(\tilde{z}) \wedge C)$$

Weakening this as suggested by Lemma 2.1, we have:

Definition 8.1. An ECSP $\langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ is *path consistent* if, for all properties $\mathcal{P}_0(\tilde{x})$, $\mathcal{P}_1(\tilde{y})$, $\mathcal{P}_2(\tilde{z})$ such that $\tilde{x} \cap \tilde{y} \neq \emptyset$, $\tilde{x} \cap \tilde{z} \neq \emptyset$, and $(\tilde{y} \cap \tilde{z}) - \tilde{x} \neq \emptyset$, and for constraints c' with $vars(c') \subseteq \tilde{x}$,

$$P_0(\tilde{x}) \wedge P_1(\tilde{y}) \wedge P_2(\tilde{z}) \wedge C \rightarrow c' \quad \text{implies } P_0(\tilde{x}) \wedge C \rightarrow c'$$

We can see immediately that this formulation of path consistency is weaker than 3-wise consistency. It is strictly weaker, as demonstrated in [39]. It is also shown in [39] that path consistency is not comparable in strength to pairwise consistency and 2-fold consistency, even in binary CSPs.

Similarly, the existence of a path consistent support for $x = a$ can be expressed as

$$\tilde{\exists}\ (P_0(\tilde{x}) \wedge P_1(\tilde{y}) \wedge P_2(\tilde{z}) \wedge C \wedge x = a)$$

The idea of a restricted consistency is that a consistency check be performed only when an inconsistency would lead to a revision of a variable domain. Obviously it can be applied to any local consistency condition in place of path consistency. For the remainder of the paper we use *X-consistency* to refer to an unspecified local consistency condition. We now further generalize the formulation so that it can apply to ECSPs, and base it on an almost arbitrary consistency. We refer to this as *restricted X-consistency* or RXC.

We introduce a parameter $\mathcal{G}$ that describes the criteria for performing a X-consistency check on a property and the information derived should the check fail. If $\langle\phi, c\rangle \in \mathcal{G}$ then X-consistency should only be applied on tuples satisfying ϕ; if the X-consistency check fails then c should be added to the constraint environment.

We need to slightly restrict the class of X-consistency conditions for which we define restricted X-consistency. We say a consistency condition is *uniform* if it has the form $\wedge_{\mathcal{P} \in Prop} \psi(\mathcal{P}, C)$, for some formula ψ. If X-consistency is uniform then, for any property $\mathcal{P}'$ (not necessarily in *Prop*), we say that $\mathcal{P}'$ is *X-consistent in* $\mathcal{E}$ if $\psi(\mathcal{P}', C)$ holds. Uniformity ensures that different properties in *Prop* are not treated differently. It allows us to consider X-consistency one property at a time. The local consistency conditions discussed in this paper are uniform, though a little reformulation might be necessary to make this apparent.

If $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ we write $\mathcal{E} \wedge C'$ for the ECSP $\mathcal{E}' = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C \wedge C'\rangle$ obtained by augmenting the constraint environment of $\mathcal{E}$ by C'.

Definition 8.2. Let $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ be an ECSP, let X-consistency be a uniform consistency condition. For any property $\mathcal{P}(\tilde{x})$ let $\mathcal{G}(\mathcal{P}, C)$ be a set of pairs $\langle\phi, c\rangle$ where ϕ is a formula and c is a constraint, both with free variables from $\tilde{x}$.

A formula ϕ has *X-consistent support* in a property $\mathcal{P}(\tilde{x}) \in Prop$ if there is a constraint $c' \in \mathcal{L}(\tilde{x})$ such that $c' \rightarrow \phi$ and the subrelation of $\mathcal{P}$ satisfying c' is X-consistent in $\mathcal{E} \wedge c'$.

A property $\mathcal{P} \in Prop$ is *restricted X-consistent wrt* $\mathcal{G}$ *and* C if, for every $\langle\phi, c\rangle \in \mathcal{G}(\mathcal{P}, C)$ such that

$$\mathcal{P} \wedge \neg\phi \wedge C \rightarrow c \quad \text{and} \quad C \not\rightarrow c$$

we have that ϕ has X-consistent support in $\mathcal{P}$.

The ECSP $\mathcal{E}$ is *restricted X-consistent wrt* $\mathcal{G}$ (or *RXC wrt X-consistency and* $\mathcal{G}$) if $\mathcal{E}$ is arc consistent and every property $\mathcal{P} \in Prop$ is restricted X-consistent wrt $\mathcal{G}$ and C.

The definition can be interpreted as: if knowing $\neg\phi$ would enable us, using $\mathcal{P}$, to infer something new (c) then ϕ must have X-consistent support in $\mathcal{P}$, where the X-consistent support ensures that $\neg\phi$ is not inferred by X-consistency applied to $\mathcal{P}$. This definition improves on the definition in [39]. If X-consistency is formulated independent of arity, and is local with regard to properties, then restricted X-consistency also has these characteristics.

To retrieve the original definition of RPC – for the ECSP $\mathcal{E}_\mathcal{C}$ corresponding to a CSP $\mathcal{C}$ – we can take X-consistency to be path consistency and define

$$\mathcal{G}_1(\mathcal{P}, C) = \{\langle x = a, p_{D(x)-\{a\}}(x)\rangle \mid \exists!\tilde{x}\, \mathcal{P}(\tilde{x}) \wedge \exists_{-\tilde{x}}\, C \wedge x = a\}$$

where $\exists!\tilde{z}\, Q(\tilde{z})$ denotes that there exists a unique value for $\tilde{z}$ such that $Q(\tilde{z})$ holds. In this case, $x = a$ is the formula ϕ with a unique tuple of $\mathcal{P}$ that satisfies C where x has the value a and $p_{D(x)-\{a\}}(x)$ (which is the way of expressing $x \neq a$ within $\mathcal{L}_\mathcal{C}$) is the constraint c corresponding to the updated domain of x. We verify that, for these parameters, RXC is restricted path consistency.

Proposition 8.1. *Let $\mathcal{C}$ be a binary CSP and let $\mathcal{E}_\mathcal{C}$ be the equivalent ECSP. Let $\mathcal{G}_1(\mathcal{P}, C) = \{\langle x = a, p_{D(x)-\{a\}}(x)\rangle \mid \exists!\tilde{x}\, \mathcal{P}(\tilde{x}) \wedge \exists_{-\tilde{x}}\, C \wedge x = a\}$.*

$\mathcal{C}$ is restricted-path consistent iff $\mathcal{E}_\mathcal{C}$ is RXC wrt path consistency and $\mathcal{G}_1$.

Proof. Let $\mathcal{C} = \langle Vars, Prop, D\rangle$ and let the corresponding ECSP be $\mathcal{E}_\mathcal{C} = \langle(\mathcal{D}_\mathcal{C}, \mathcal{L}_\mathcal{C}), Vars, Prop, C_D\rangle$. By Proposition 4.1, $\mathcal{C}$ is arc consistent iff $\mathcal{E}_\mathcal{C}$ is arc consistent in the extended sense. Thus we need only consider the path consistency aspect of RPC. In the proof, we refer to path consistency in $\mathcal{E}_\mathcal{C}$ as X-consistency, to distinguish it from path consistency in $\mathcal{C}$.

Suppose $\mathcal{E}_\mathcal{C}$ is RXC wrt $\mathcal{G}_1$ and X-consistency, and consider the CSP $\mathcal{C}$. Let $x \in Vars$ and value $a \in D(x)$, and for each property in $Prop$ involving x, say $\mathcal{P}_0(x, y)$, suppose there is a unique value $b \in D(y)$ such that $\mathcal{P}_0(a, b)$. Then $\mathcal{G}_1(\mathcal{P}_0, C_D)$ contains $\langle x = a, p_{D(x)-\{a\}}(x)\rangle$, which we will refer to as $\langle\phi, c\rangle$. Clearly $\mathcal{P}_0 \wedge \neg\phi \wedge C_D \rightarrow c$ and $C_D \not\rightarrow c$.

Let z be another variable and suppose there are properties $\mathcal{P}_1(x, z)$ and $\mathcal{P}_2(y, z)$ in $Prop$. Since $\mathcal{E}_\mathcal{C}$ is RXC, $x = a$ has X-consistent support in $\mathcal{P}_0$. That is, there is c' such that $c' \rightarrow x = a$ and the subrelation of $\mathcal{P}_0$ satisfying c' is X-consistent in $\mathcal{E} \wedge c'$. Since $y = b$ is the unique support for $x = a$ in $\mathcal{P}_0$, and by X-consistency in $\mathcal{E} \wedge c'$, there is a value $d \in D(z)$ such that $\mathcal{P}_1(a, d)$ and $\mathcal{P}_2(b, d)$, that is, $x = a \wedge y = b$ is path consistent. It follows that $\mathcal{C}$ is RPC.

For the other direction, suppose $\mathcal{C}$ is RPC and consider $\mathcal{E}_\mathcal{C}$. Let $\mathcal{P}_0(x, y)$ be a property and suppose there is $\langle\phi, c\rangle \in \mathcal{G}_1(\mathcal{P}_0, C_D)$ such that $\mathcal{P}_0 \wedge \neg\phi \wedge C_D \rightarrow c$ and $C_D \not\rightarrow c$. Then ϕ has the form $x = a$ and c is $p_{D(x)-\{a\}}(x)$ where $\langle a, b\rangle$ is the only tuple satisfying $\mathcal{P}_0(x, y) \wedge C_D \wedge x = a$, and hence $y = b$ is the only support for $x = a$ in $\mathcal{C}$. By the definition of RPC for $\mathcal{C}$, $x = a$ has a path consistent support and hence if there is a variable z and properties $\mathcal{P}_1(x, z)$ and $\mathcal{P}_2(y, z)$ in $Prop$ then there is a value $d \in D(z)$ such that $\mathcal{P}_1(a, d)$ and $\mathcal{P}_2(b, d)$. Taking $x = a \wedge y = b$ as c' in the definition of X-consistent support we see that ϕ has X-consistent support in $\mathcal{P}_0$. It follows that $\mathcal{E}_\mathcal{C}$ is restricted path consistent. □

The above definition also captures k-RPC [14] when $\mathcal{G}$ restricts to bindings $x = a$ with k or fewer tuples in $\mathcal{P}$. To retrieve the original definition of k-RPC – for the ECSP $\mathcal{E}_\mathcal{C}$ corresponding to a CSP $\mathcal{C}$ – we can define

$$\mathcal{G}_k(\mathcal{P}, C) = \{\langle x = a, p_{D(x)-\{a\}}(x)\rangle \mid \exists^{\leq k}\tilde{x}\, \mathcal{P}(\tilde{x}) \wedge \exists_{-\tilde{x}}\, C \wedge x = a\}$$

where $\exists^{\leq k}\tilde{z}\, Q(\tilde{z})$ denotes that there exist fewer than (or exactly) k valuations for $\tilde{z}$ such that $Q(\tilde{z})$ holds.

To represent Max-RPC we define

$$\mathcal{G}_{max}(\mathcal{P}, C) = \{\langle x = a, p_{D(x)-\{a\}}(x)\rangle \mid \exists\tilde{x}\, \mathcal{P}(\tilde{x}) \wedge C \wedge x = a\}$$

Restricted X-consistency satisfies some monotonicity properties. The proofs are straightforward.

Proposition 8.2. *Let $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ be an extended CSP.*

- *Let $\mathcal{G}(\mathcal{P}, C)$ and $\mathcal{G}'(\mathcal{P}, C)$ be sets of formula-constraint pairs and suppose $\forall\mathcal{P} \in Prop\ \mathcal{G}(\mathcal{P}, C) \subseteq \mathcal{G}'(\mathcal{P}, C)$.*
  If $\mathcal{E}$ is restricted X-consistent wrt $\mathcal{G}'$ then $\mathcal{E}$ is restricted X-consistent wrt $\mathcal{G}$.
- *Suppose X-consistency is stronger than Y-consistency then, for any $\mathcal{G}$, restricted X-consistency is stronger than restricted Y-consistency.*

Notice that $\mathcal{G}_1(\mathcal{P}, C) \subseteq \mathcal{G}_k(\mathcal{P}, C) \subseteq \mathcal{G}_{k+1}(\mathcal{P}, C) \subseteq \mathcal{G}_{max}(\mathcal{P}, C)$. Using monotonicity, we have the results of [15] relating these restricted consistencies, in the context of ECSPs. That is, RPC is weaker than k-RPC, which is weaker than $(k + 1)$-RPC, which is weaker than Max-RPC.

9. Extending singleton consistencies

A CSP $\mathcal{C}$ is singleton arc consistent (SAC) if all variable domains are non-empty and the CSP resulting from the restriction of any one variable domain to a singleton can be made arc consistent [13,44]. That is, for each possible singleton domain there is an arc consistent sub-CSP of $\mathcal{C}$ that has that singleton domain. Clearly, the idea of singleton consistency can be applied to any consistency condition [13], and not only arc consistency.

We can view SAC and other singleton consistencies as applying a consistency condition one level deeper in the search tree, assuming branching in the search tree is done by choosing values for a variable. That suggests we consider other constraints

that might be the basis for branching during a search. In particular, branching by domain splitting is widely used in interval reasoning on non-linear constraints over the reals, and also is useful on some integer finite domain problems.

Consequently, we define a notion of singleton consistencies that is parameterized by a set of branching constraints and a consistency condition. Notice that the use of "singleton" in reference to these extended consistency conditions is now a misnomer, since domains do not occur explicitly in ECSPs and, in general, the effect of branching will not produce a singleton domain. Perhaps *branching consistencies* is a more accurate name, but we will refer to these as singleton consistencies, for consistency with their CSP equivalents.

Definition 9.1. Let $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ be an ECSP. Let X-consistency be a consistency condition and let S be a set of constraints.

$\mathcal{E}$ is *SXC wrt S and X-consistency* if for every $s \in S$ such that $C \wedge s$ is satisfiable, there is a satisfiable constraint environment C' such that $C' \rightarrow (C \wedge s)$ and $\mathcal{E} \wedge C'$ is X-consistent.

Notice that C' is not necessarily the constraint environment computed by an X-consistency method from $C \wedge c$; the existence of C' implies that the computed X-consistent environment also exists.

Let $Bind = \{x = a \mid x \in Vars, a \text{ is a constant in } \Sigma\}$ be the set of all bindings of variables to values in a CSP constraint domain, and $Mid = \{x \leq a, x \geq a \mid x \in Vars, D(x) = [l, h], a = \frac{l+h}{2}\}$ be the set of all domain splittings at interval midpoints. SXC consistency wrt *Mid* and arc consistency is an interesting alternative to SAC since it involves exploring fewer branches (albeit each with a weaker strengthening of the constraint environment). We can recover the original definition of SAC for CSPs by taking S to be *Bind*, and taking arc consistency in place of X-consistency.

Proposition 9.1. *Let $\mathcal{C}$ be a CSP and let $\mathcal{E}_\mathcal{C}$ be the equivalent ECSP.*
$\mathcal{C}$ is SAC iff $\mathcal{E}_\mathcal{C}$ is SXC consistent wrt Bind and arc consistency.

Proof. Suppose $\mathcal{E}_\mathcal{C}$ is SXC consistent wrt S and X-consistency. For every $s \in S$, let $\mathcal{E}_s = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C'\rangle$ where $C'$is the constraint environment referred to in Definition 9.1. Then $\mathcal{E}_s$ is arc consistent, since X-consistency is arc consistency. Since $C' \rightarrow (C \wedge s)$, C' incorporates a stronger domain for each variable than in C_D and restricts the domain of the variable in s to a singleton. Thus the CSP where $\mathcal{C}$ is modified so that D is restricted to a singleton by s is arc consistent, by Proposition 4.1. Thus $\mathcal{C}$ is singleton arc consistent.

Suppose $\mathcal{C}$ is singleton arc consistent. For each binding s used to make a singleton domain, let D_s be the domain of the corresponding arc consistent sub-CSP and C_s be the constraint environment corresponding to D_s. Then C_s is satisfiable and $C_s \rightarrow (C_D \wedge s)$. By Proposition 4.1, $\langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C_s\rangle$ is arc consistent. Since this holds for any binding s that is consistent with D, $\mathcal{E}_\mathcal{C}$ is SXC consistent wrt S and X-consistency. □

We now establish a relationship between restricted and singleton consistencies. Because RXC and SXC are parameterized in different ways, we restrict attention to parameterizations that are compatible.

We say RXC wrt $\mathcal{G}$ *is in tune with* SXC wrt S (or, simply, $\mathcal{G}$ is in tune with S) if, for all properties $\mathcal{P}(\tilde{x}) \in Prop$, all constraint environments C, and all $\langle\phi, c\rangle \in \mathcal{G}(\mathcal{P}, C)$ such that $\mathcal{P} \wedge \neg\phi \wedge C \rightarrow c$ and $C \not\rightarrow c$, there is a constraint $c_S(\tilde{x}) \in S$ such that $C \wedge c_S$ is satisfiable and $C \wedge c_S \rightarrow \phi$. In this case, whenever an X-consistency check must be made in enforcing RXC there is a branching constraint that focuses SXC on this case.

The following result is the counterpart of Theorem 6 of [15], which showed that Max-RPC is weaker than SAC on binary CSPs.

Proposition 9.2. *Let $\mathcal{E} = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C\rangle$ be an ECSP. Let $\mathcal{G}(\mathcal{P}, C)$ be a set of pairs $\langle\phi, c\rangle$ and S be a set of constraints such that $\mathcal{G}$ is in tune with S. Let X-consistency be a uniform local consistency.*
If $\mathcal{E}$ is arc consistent and SXC-consistent wrt S and X-consistency then $\mathcal{E}$ is restricted X-consistent wrt $\mathcal{G}$.

Proof. Suppose $\mathcal{E}$ is SXC-consistent and arc consistent, but is not restricted X-consistent wrt $\mathcal{G}$. By the latter, there is a property $\mathcal{P}$ and $\langle\phi, c\rangle \in \mathcal{G}(\mathcal{P}, C)$ such that $\mathcal{P} \wedge \neg\phi \wedge C \rightarrow c$ and $C \not\rightarrow c$, but ϕ does not have X-consistent support in $\mathcal{P}$, that is, for every $c' \in \mathcal{L}(\tilde{x})$ such that $c' \rightarrow \phi$, the subrelation of $\mathcal{P}$ satisfying c' is not X-consistent in $\mathcal{E}$. Since $\mathcal{G}$ is in tune with S, there is a constraint $c_S \in S$ such that $C \wedge c_S \rightarrow \phi$ and $C \wedge c_S$ is satisfiable. Since $\mathcal{E}$ is SXC-consistent wrt S and X-consistency, there is a satisfiable constraint C' such that $C' \rightarrow (C \wedge c_S)$ and $\mathcal{E}' = \langle(\mathcal{D}, \mathcal{L}), Vars, Prop, C'\rangle$ is X-consistent. Clearly $C' \rightarrow \phi$. But, because ϕ does not have X-consistent support in $\mathcal{P}$, the subrelation of $\mathcal{P}$ satisfying C' is not X-consistent in $\mathcal{E} \wedge C'$. This contradiction shows that $\mathcal{E}$ must be restricted X-consistent wrt $\mathcal{G}$. □

10. Consistencies using a relaxed domain

In addition to modulating consistencies with variations in the language of constraints used within the consistency definition, as in Section 4, we can also vary the constraint domain that defines the constraints. In particular, we can relax a consistency by interpreting the constraints and/or properties within a relaxed domain. First we must define this concept.

Definition 10.1. Let $\mathcal{R}$ and $\mathcal{D}$ be structures over a signature that define both constraints and properties. We say $\mathcal{R}$ is a *relaxation* of $\mathcal{D}$ if $\mathcal{D}$ is a substructure of $\mathcal{R}$ and, for every conjunction ψ of properties and constraints, every solution of ψ in $\mathcal{D}$ is also a solution of ψ in $\mathcal{R}$.

Relaxation has been used in related contexts. In integer programming, a standard approach to solving a problem involves first relaxing the problem and solving it over the real numbers [28]. It is straightforward to see that integer constraints interpreted over the real numbers form a relaxation of those constraints in the sense of the previous definition. Similarly, CAL [1] uses non-linear constraints over the reals, but solves them over the complex numbers. The use of box consistency [4] for interval reasoning on non-linear functions and relations is an example where the underlying domain is not relaxed; the relaxation comes from an interval extension of the original properties [4]. Other work [11,5] has used a relaxation of a constraint domain to characterize the behavior of an incomplete constraint solver.

Using a relaxation we can define a weaker form of any consistency by requiring only that the information that can be inferred in the relaxed domain must be made explicit in the constraint environment C.

Thus, for example, extended arc consistency wrt a relaxation $\mathcal{R}$ of $\mathcal{D}$ requires that

$$\mathcal{R} \models (\mathcal{P} \wedge C) \rightarrow c' \quad \text{implies } \mathcal{D} \models C \rightarrow c'$$

This is, in general, a weaker consistency than extended arc consistency because $\mathcal{R}$ will entail a subset of the formulas entailed by $\mathcal{D}$ (of course, if $\mathcal{R}$ is chosen to be $\mathcal{D}$ this reduces to Definition 4.1). The use of a relaxed consistency condition occurs in the constraint domain of integer finite domain constraints. [10] notes the difference between bounds consistency based on the integers and the reals (called, respectively, *bounds*($\mathcal{Z}$) and *bounds*($\mathcal{R}$) consistency in [10]) in this constraint domain, and cites uses in the literature of each. For more information on the relationship between these consistencies, the reader is referred to [10].

Although the variety of ways that consistencies can be defined makes a general result difficult to formulate, we can at least express the effect of relaxation on consistencies presented as an implication.

Proposition 10.1. *Let $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{R}_1$, $\mathcal{R}_2$ be structures where $\mathcal{R}_i$ is a relaxation of $\mathcal{D}_i$, for $i = 1, 2$.*
A consistency of the form

$$\mathcal{D}_1 \models \psi \quad \textit{implies } \mathcal{D}_2 \models \phi$$

is stronger than

$$\mathcal{R}_1 \models \psi \quad \textit{implies } \mathcal{D}_2 \models \phi$$

and weaker than

$$\mathcal{D}_1 \models \psi \quad \textit{implies } \mathcal{R}_2 \models \phi$$

The result follows immediately from the structure of the consistency and the definition of relaxation. It applies to arc, path, k-wise and k-fold consistencies according to the formulations in previous sections.

11. Conclusion

This paper has developed the framework of extended constraint satisfaction problems. This represents a synthesis of constraint satisfaction and constraint solving that is suitable for studying search problems that arise in constraint programming. It provides a uniform framework in which we can study finite domain constraint programming, interval methods on non-linear real arithmetic constraints, and search in languages such as CLP($\Re$). In addition, several generalized local consistency conditions have been developed, with a focus on locality of properties, rather than variables, which is more appropriate in handling relations that are implemented as reactive threads of computation. Of these, we have identified some, such as 2-fold consistency, restricted 2-fold consistency, and SXC consistency wrt *Mid* and arc consistency, that appear worthy of further investigation.

Acknowledgements

Thanks to Rina Dechter, Jeremy Frank, Mark Wallace, Toby Walsh and anonymous reviewers for discussions and suggestions that were helpful for this paper.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] A. Aiba, K. Sakai, Y. Sato, D.J. Hawley, R. Hasegawa, Constraint logic programming language CAL, in: Proceedings of the 2nd. International Conference on Fifth Generation Computer Systems, vol. 1, 1988, pp. 263–276.
[2] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, Journal of the ACM 30 (3) (1983) 479–513.
[3] N. Beldiceanu, E. Contejean, Introducing global constraints in CHIP, Mathematical Computer Modelling 20 (12) (1994) 97–123.
[4] F. Benhamou, D.A. McAllester, P. Van Hentenryck, CLP (Intervals) revisited, in: International Symposium on Logic Programming, 1994, pp. 124–138.
[5] F. Benhamou, J.-L. Massat, Boolean pseudo-equations in constraint logic programming, in: Proc. 10th International Conference on Logic Programming, 1993, pp. 517–531.
[6] F. Benhamou, W.J. Older, Applying interval arithmetic to real, integer, and Boolean constraints, Journal of Logic Programming 32 (1) (1997) 1–24.

[7] P. Berlandier, Improving domain filtering using restricted path consistency, in: Proc. IEEE International Conference on Artificial Intelligence Applications, CAIA, 1995.
[8] C. Bessière, J.C. Régin, Local consistency on conjunctions of constraints, in: J.C. Régin, W. Nuijtens (Eds.) Proceedings ECAI'98 Workshop on Non-binary constraints, Brighton, UK, pp. 53–59.
[9] Y. Caseau, F. Josset, F. Laburthe, CLAIRE: Combining sets, search and rules to better express algorithms, Theory and Practice of Logic Programming 2 (6) (2002) 769–805.
[10] C.W. Choi, W. Harvey, J.H.M. Lee, P.J. Stuckey, Finite domain bounds consistency revisited, in: Proc. Australian Conference on Artificial Intelligence, in: LNCS, vol. 4304, Springer, 2006, pp. 49–58.
[11] A. Colmerauer, Naive solving of non-linear constraints, in: F. Benhamou, A. Colmerauer (Eds.), Constraint Logic Programming: Selected Research, MIT Press, 1993, pp. 89–112.
[12] A. Colmerauer, Solving the multiplication constraint in several approximation spaces, in: Proc. ICLP, 1, 2001.
[13] R. Debruyne, C. Bessière, Some practicable filtering techniques for the constraint satisfaction problem, IJCAI 1 (1997) 412–417.
[14] R. Debruyne, C. Bessière, From restricted path consistency to max-restricted path consistency, in: Proc. Principles and Practice of Constraint Programming, 1997, pp. 312–326.
[15] R. Debruyne, C. Bessière, Domain filtering consistencies, Journal of Artificial Intelligence Research 14 (2001) 205–230.
[16] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[17] R. Dechter, P. van Beek, Local and global relational consistency, Theoretical Computer Science 173 (1) (1997) 283–308.
[18] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95.
[19] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, The constraint logic programming language CHIP, in: Proceedings of the 2nd. International Conference on Fifth Generation Computer Systems, 1988, pp. 249–264.
[20] E.C. Freuder, Synthesizing constraint expressions, Communications of the ACM 21 (11) (1978) 958–966.
[21] E.C. Freuder, A sufficient condition for backtrack-bounded search, Journal of the ACM 32 (4) (1985) 755–761.
[22] E.C. Freuder, C.D. Elfe, Neighborhood inverse consistency preprocessing, in: Proc. AAAI/IAAI, vol. 1, 1996, pp. 202–208.
[23] T.W. Frühwirth, Theory and practice of constraint handling rules, Journal of Logic Programming 37 (1–3) (1998) 95–138.
[24] C. Gervet, Interval propagation to reason about sets: Definition and implementation of a practical language, Constraints 1 (3) (1997) 191–244.
[25] L. Granvilliers, F. Benhamou, Algorithm 852: RealPaver: An interval solver using constraint satisfaction techniques, ACM Transactions on Mathematical Software 32 (1) (2006) 138–156.
[26] M. Gyssens, On the complexity of join dependencies, ACM TODS 11 (1) (1986) 81–108.
[27] W. Harvey, P.J. Stuckey, Constraint representation for propagation, in: Proc. Conf. on Principles and Practice of Constraint Programming, 1998, pp. 235–249.
[28] F.S. Hillier, G.J. Lieberman, Introduction to Operations Research, McGraw-Hill, 2001.
[29] ILOG Inc., ILOG Solver 4.2 User's Manual, 1998.
[30] J. Jaffar, J-L. Lassez, Constraint logic programming, in: Proc. 14th ACM Symposium on Principles of Programming Languages, 1987, pp. 111–119.
[31] J. Jaffar, M.J. Maher, Constraint logic programming: A survey, Journal of Logic Programming 19 & 20 (1994) 503–581.
[32] J. Jaffar, S. Michaylov, P. Stuckey, R.H.C. Yap, The CLP($\mathfrak{R}$)language and system, ACM Transactions on Programming Languages 14 (3) (1992) 339–395.
[33] J. Jaffar, S. Michaylov, R.H.C. Yap, A methodology for managing hard constraints in CLP systems, in: Proc. ACM-SIGPLAN Conference on Programming Language Design and Implementation, 1991, pp. 306–316.
[34] P. Jégou, On the consistency of general constraint-satisfaction problems, in: AAAI, 1993, pp. 114–119.
[35] P.C. Kanellakis, Elements of relational database theory, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, Elsevier, 1990, pp. 1073–1156.
[36] T. Le Provost, M. Wallace, Generalised constraint propagation over the CLP scheme, Journal of Logic Programming 16 (3) (1993) 319–360.
[37] M.J. Maher, Adding constraints to logic-based formalisms, in: K.R. Apt, V. Marek, M. Truszczynski, D.S. Warren (Eds.), The Logic Programming Paradigm: A 25 Years Perspective, in: Artificial Intelligence Series, Springer-Verlag, 1999, pp. 313–331.
[38] M.J. Maher, Propagation completeness of reactive constraints, in: Proc. International Conference on Logic Programming, 2002, pp. 148–162.
[39] M.J. Maher, A synthesis of constraint satisfaction and constraint solving, in: Proc. Conference on Principles and Practice of Constraint Programming, in: LNCS, vol. 2833, Springer, 2003, pp. 525–538.
[40] K. Marriott, P.J. Stuckey, Programming with Constraints : An Introduction, MIT Press, 1998.
[41] R. Mohr, G. Masini, Good Old Discrete Relaxation, in: Proc. ECAI, 1988, pp. 651–656.
[42] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Information Sciences 7 (1974) 95–132.
[43] W. Pang, S.D. Goodwin, Consistency in general CSPs, in: PRICAI 2000, 2000, pp. 469–479.
[44] P. Prosser, K. Stergiou, T. Walsh, Singleton consistencies, in: Proc. Int. Conf. on Principles and Practice of Constraint Programming, in: LNCS, vol. 1894, 2000, pp. 353–368.
[45] A. Sadler, C. Gervet, Hybrid set domains to strengthen constraint propagation and reduce symmetries, in: Proc. CP, in: LNCS, vol. 3258, 2004, pp. 604–618.
[46] P. Van Hentenryck, A gentle introduction to NUMERICA, Artificial Intelligence 103 (1–2) (1998) 209–235.
[47] P. Van Roy, P. Brand, D. Duchier, S. Haridi, M. Henz, C. Schulte, Logic programming in the context of multiparadigm programming: The Oz experience, Theory and Practice of Logic Programming 3 (6) (2003) 715–763.
[48] M. Wallace, Search in AI: Escaping from the CSP straightjacket, in: Proc. of European Conf. on Artificial Intelligence, IOS Press, 2000, pp. 770–776.
[49] T. Walsh, Relational Consistencies, APES Technical Report, APES-28-2001, 2001.