Language, Definition and Optimal Computation of CSP Approximations

Arnaud Lallouet, Thi Bich Hanh Dao and AbdelAli Ed-Dbali
Université d'Orléans — LIFO
BP 6759 — F-45067 Orléans cedex 2

Abstract

In this paper, we introduce a formal framework to describe CSP approximations (usually called consistencies), showing the importance of the language (or data-structure) used to perform this consistency. We introduce the notion of R-consistency which takes into account the representation of the data and which generalizes many known consistencies. We then automatically derive from a CSP and its approximation scheme a rule system describing the constraint propagation and we propose an optimal algorithm in the spirit of AC4 to achieve it.

Introduction

This paper introduces the following notions:

- *The formalism of CSP approximation:* the main novelty of the CSP approximation framework is to show how the language used for the computation, or equivalently the data-structure used to represent the problem, is a central issue in determining the level of local consistency which can be performed. We propose to make explicit the language used for the computation by approximating a CSP C to be solved by another CSP K used as a data-structure.
- *R-consistency:* this form of consistency defines the pruning of tuples relatively to the approximating CSP K. It does not make other assumptions about represented data. For example, variables and their domains may not be accessible.
- *A rule system:* from a CSP C, an approximation framework K and a notion of consistency, we automatically derive a rule system to perform the consistency.
- *The RC4 algorithm:* this algorithm optimally enforces the desired level of consistency. Its first part consists in the rule generation in such a way that its second part can optimally perform the propagation. Optimality is understood in the same meaning as in the AC4 algorithm (Mohr & Henderson 1986). In particular, it does not mean that this algorithm could be faster than optimized ones (see (Bessière 1994) for one of the first discussion on this topic).

CSP approximations and R-consistency

In this section, we present a high-level model for the approximation computed by some consistencies. This set-theoretic formulation allows to describe very precisely the approximation process without introducing a particular language to express constraints or reductions done in the solving process. The basic idea is to clearly separate the CSP to be solved from its approximation (and more generally from the sequence of approximating CSPs). In solvers, the approximating CSP represents the evolving data-structure used for the computation.

Let V be a set of variables and $D = (D_X)_{X \in V}$ their (finite) domains. For $W \subseteq V$, we denote by D^W the set of tuples on W, namely $\Pi_{X \in W} D_X$. Therefore, we have $D^V = \Pi D$. Projection of a tuple or a set of tuples on a set of variables is denoted by $|$, natural join of two sets of tuples is denoted by $\bowtie$. If A is a set, then $\mathcal{P}(A)$ denotes its powerset and for $a \in A$, A_{-a} denotes $A - \{a\}$.

Definition 1 (Constraint) *A* constraint c *is a pair* (W, T) *where*

- $W \subseteq V$ *is the* arity *of the constraint* c *and is denoted by* $var(c)$.
- $T \subseteq D^W$ *is the* solution *of* c *and is denoted by* $sol(c)$.

A *CSP* is a set of constraints. The join of two constraints is defined as a natural extension of the join of tuples: the *join* of c and c' is the constraint $c \bowtie c' = (var(c) \cup var(c'), sol(c) \bowtie sol(c'))$. Similarly, for $c = (W, T)$ and $U \subseteq V$, we define:

- the projection $c|_U = (W \cap U, T|_U)$.
- the cardinal $|c| = |sol(c)|$.

Join is naturally extended to CSPs and the *solutions* of a CSP C are $sol(\bowtie C)$. A direct computation of this join is too expensive to be tractable. We define the following notion of approximation between constraints and also between CSPs:

Definition 2 (Approximation ordering) *A constraint* $c' = (W', T')$ approximates $c = (W, T)$*, denoted by* $c \subseteq c'$*, if:*

$$var(c) = var(c') \text{ and } sol(c) \subseteq sol(c')$$

A CSP K approximates C*, denoted by* $C \subseteq K$*, if:*

$$\bowtie C \subseteq \bowtie K$$

In the following, we consider that the resolution of the CSP C will be done within the approximation defined by K. Intuitively, the CSP K is intended to be physically represented, for example (but not exclusively) by sets of tuples. This is why C and K may be built on completely different constraints. When all constraints in K are unary, and thus represent the domain of variables, it yields to the well-known "domain reduction scheme" used in most solvers and represented in figure 1. More generally, K defines the language we are allowed to manipulate: an access to the constraints of C can be made only via this representation. The good choice of K is a very important step and is related to the choice of an ontology in the field of Knowledge Representation. We

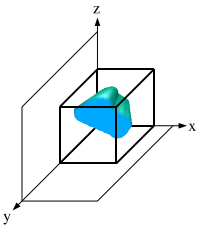

Figure 1: Domain approximation of a CSP.

call K the *approximating* CSP. In the following, C represents the CSP to be solved and K the approximating CSP. Here is an example of two different approximating CSPs:

Example 3 *Let C be the CSP composed of one constraint: $c = (\{X, Y, Z\}, \{(0,0,0), (0,0,1), (0,1,0), (1,0,1)\})$. Here are two approximating CSPs $K_1 = \{x, y, z\}$ and $K_2 = \{x, yz\}$ which are represented in figure 2. Here $x = (\{X\}, \{(0), (1)\})$ is an unary constraint representing the domain of X and $yz = (\{Y, Z\}, \{(0,0), (0,1), (1,0)\})$ is an arbitrary binary constraint. The constraints y and z are defined in the same straightforward way.*

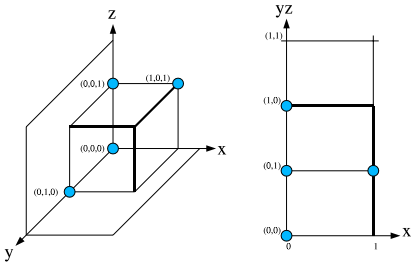

Figure 2: Two different approximations for a CSP.

Most of the time, switching from C to K provides a gain in terms of memory consumption. For example, in the domain reduction approximation for n variables and a domain of size m, it boils down from m^n to $m * n$. But the trade-off is that representable approximations are less precise since they are limited to (the union of) cartesian products. Most (all?) solvers only represent variable domains and hence the best approximation, which is computed by arc-consistency when only one constraint is processed at a time, is limited by this representation of data. The use of non-unary constraints in K opens the way to a very fine tuning between the precision of the approximation and the memory consumption. Of course, the use of arc-consistency on a non-unary approximating CSP allows prunings which are impossible to get in the domain approximation framework and which are different from those obtained by path-consistency or k-consistency, where two or more constraints are processed at a time. By using locally a precise representation, it is possible to strenghten the efficiency of filtering on some important variables.

For a CSP C and a constraint k, we call *C-scope* of k the set $C(k) = \{c \in C \mid var(c) \cap var(k) \neq \emptyset\}$. An approximating CSP K is *precise* w.r.t. C if $\forall k \in K, \forall c \in C$, $var(k) \cap var(c) \neq \emptyset \Rightarrow var(k) \subseteq var(c)$. The domain reduction scheme is a precise approximation. Precise approximations provide a simplification in the rule generation algorithm.

Since our goal is to find the best approximation (in some sense), we call an approximating CSP K a *search state* and the set of such states is the *search space* $\mathcal{S} = \Pi_{k \in K} \mathcal{P}(D^{var(k)})$. We call *singletonic* a CSP K such that $|\bowtie K| = 1$ (such a CSP represents a single tuple). A consistency is a subset $\mathcal{C} \subseteq \mathcal{S}$ such that:

$$\forall K \in \mathcal{S}, \ |\bowtie K| = 1 \Longrightarrow (K \subseteq C \Longleftrightarrow K \in \mathcal{C})$$

In other words, a consistency contains (a representation of) every solution of C and rejects every non-solution. All known local consistencies differ by the way they treat non-singletonic CSPs. Usually, $\mathcal{C}$ is closed by union. Global consistency is exactly the set of representations in K of solutions of C and null-consistency is $\mathcal{S}$ itself: it does nothing or, which is the same, all search states are consistent. Note also that in this framework, consistencies can be compared easily by set-inclusion. Moreover, it gives new lights on the basic nature of consistencies, for example, it is obvious that no local consistency can enforce global consistency if $\nexists k \in K, var(k) = var(C)$. This explains for example why a n-consistent (Mohr & Masini 1988) CSP can still have no solution.

We present here a notion of relational consistency called *R-consistency* which takes into account the separation between C and K.

Definition 4 (c-R-consistency) *A constraint $k \in K$ is c-R-consistent if:*

$$k \in K(c) \Longrightarrow k \subseteq (c \bowtie K(c))\,|_{var(k)}$$

Intuitively, it means that if a tuple belongs to k, then it must extend a solution of c in the current approximation $K(c)$, $K(c)$ being the K-scope of c. The approximating CSP K is c-R-consistent if $\forall k \in K$, k is c-R-consistent.

Definition 5 (1R-consistency) *An approximating CSP K of a CSP C is 1R-consistent if $\forall c \in C$, K is c-R-consistent.*

Definition 6 (nR-consistency) *An approximating CSP K of a CSP C is nR-consistent if $\forall \{c_1, \ldots, c_n\} \subseteq C$, K is $(c_1 \bowtie \ldots \bowtie c_n)$-R-consistent.*

We also call 1R-consistency *ArcR-consistency* and 2R-consistency *PathR-consistency*.

Proposition 7 *In the particular case of the domain reduction scheme, 1R-consistency coincides with node-, arc- and hyperarc-consistencies and mR-consistency amounts to relational $(1, m)$-consistency (Dechter & van Beek 1997). In particular, 2R-consistency is the relational path-consistency.*

The goal of a consistency-enforcing algorithm is to build a sequence of approximations $(K_i)_{i=1..n}$ from an arbitrary search state $K_0 = K$ to the greatest consistent state included in K.

Propagation rules

Enforcing R-consistency can be made explicit by a rule system we call *propagation rules*. In this paper, we adopt a logic programming presentation: for a constraint $c = (W, T)$ and a tuple $a \in D^W$, we associate an *atom* $c(a)$. We call *Herbrand base* $HB(K)$ the set of atoms obtained by instanciation of constraints of K. Following the same idea, a constraint c is identified with the set of its atoms $\{c(a) | a \in sol(c)\}$.

The intuitive idea is that, since K is intended to be the evolving data-structure, the propagation occurs between atoms of $HB(K)$. An atom is intended to be true if it is eliminated by the propagation process, which means that it does not participate to any solution of the CSP C.

Definition 8 (Propagation rule) *A propagation rule is a definite clause $h \leftarrow B$ where $h \in HB(K)$ and $B \subseteq HB(K)$.*

A propagation rule states that the head atom h can be deleted whenever all atoms in its body are. In the domain reduction case, an atom represents a value in a variable's domain. Rule bodies must be carefully chosen to cover all possible reason for the head to be removed. Our rule generation algorithm provides a system of such rules for all possible atoms in $HB(K)$.

We first present rule generation on three examples before stating the general case:

Example 9 (Arc-consistency) *Let $V = \{X, Y\}$, let C be composed of only one constraint $c = (\{X, Y\}, \{(1, 2), (1, 3)\})$ and K be composed of two constraints $x = (\{X\}, \{(1), (2)\})$ and $y = (\{Y\}, \{(2), (3)\})$ representing the domain of the two variables. The Herbrand base is $\{x(1), x(2), y(2), y(3)\}$ and every atom will be the head of one (or more) rule. Let us build first the rule for $x(1)$. The supports (in the sense of AC4 (Mohr & Henderson 1986)) of $x(1)$ are $y(2)$ and $y(3)$. Hence it yields the rule:*

$$x(1) \leftarrow y(2), y(3)$$

and this is the only rule for $x(1)$. For $x(2)$, and since this atom has no support, we generate the fact:

$$x(2) \leftarrow$$

For y, both $y(2)$ and $y(3)$ have $x(1)$ as only support, yielding the rules:

$$y(2) \leftarrow x(1)$$
$$y(3) \leftarrow x(1)$$

In the second example, we deal with hyperarc-consistency (arc-consistency for non-binary constraints) but still in the domain reduction approximation framework. In this example, we have to select the interesting part of the constraint to find the atoms of K involved to maintain the head of a rule:

Example 10 (Hyperarc-consistency) *Let $V = \{X, Y, Z\}$, let C be a CSP with one constraint $c = (\{X, Y, Z\}, \{(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 3)\}$ described in figure 3 and K be composed of three constraints $x = (\{X\}, \{(1), (2)\})$, $y = (\{Y\}, \{(2), (3)\})$ and $z = (\{Z\}, \{(3), (4)\})$ representing the domain of the variables. Let us find the rules for $x(1)$. The useful part of*

X	Y	Z
1	2	3
1	2	4
1	3	4
2	3	3

Figure 3: Selection of c where $X = 1$.

the constraint c is the set of tuples where $X = 1$ (the grey part in figure 3). In other words, $x(1)$ can be removed if all these tuples of c are discarded because of the removal of values in the domain of Y and Z. The first tuple of c, $(1, 2, 3)$ is discarded either if $y(2)$ or $z(3)$ is removed, the second if $y(2)$ or $z(4)$ is removed and the last one if $y(3)$ or $z(4)$ is removed. Then, in order to discard all these tuples of c, we just have to consider the removal of any one atom in each line. For example, the removal of $y(2)$ and $y(3)$ is enough to forbid all three atoms, and so are the removal of $y(2)$ and $z(4)$. By taking all possibilities, we get the following set of rules (rules with $*$ are redundant):

$$\begin{aligned}
x(1) &\leftarrow y(2), y(3) \\
x(1) &\leftarrow y(2), z(4) \\
x(1) &\leftarrow y(2), z(4), y(3) \quad * \\
x(1) &\leftarrow z(3), y(2), y(3) \quad * \\
x(1) &\leftarrow z(3), y(2), z(4) \quad * \\
x(1) &\leftarrow z(3), z(4), y(3) \quad * \\
x(1) &\leftarrow z(3), z(4)
\end{aligned}$$

Other rules include:

$$\begin{aligned}
x(2) &\leftarrow y(3) \\
x(2) &\leftarrow z(3) \\
y(2) &\leftarrow x(1) \\
y(2) &\leftarrow x(1), z(3) \quad * \\
y(2) &\leftarrow x(1), z(4) \quad *
\end{aligned}$$

The last example presents an approximating CSP different from the domain approximation one:

Example 11 (1R-consistency) *Let $V = \{X, Y, Z\}$, let C be a CSP with three constraints "$X \neq Y$", "$X + Y \leq Z$" and "$X + Y + Z = 8$" and K be composed of two constraints $xy = (\{X, Y\}, \{1, 2, 3\} \times \{1, 2, 3\})$ and $z = (\{Z\}, \{(2), (3), (4)\})$. Because "$X \neq Y$" has the same variables as the represented constraint xy, 1R-consistency is here equivalent to a generalization of node-consistency. Hence the generated rules are just facts:*

$$\begin{aligned}
xy(1, 1) &\leftarrow \\
xy(2, 2) &\leftarrow \\
xy(3, 3) &\leftarrow
\end{aligned}$$

Let us find the rules whose head is $xy(1,2)$. Since the selection of the tuples where $X = 1$ and $Y = 2$ in "$X+Y+Z = 8$" is empty, we just have the fact:

$$xy(1,2) \quad \leftarrow$$

Because of the two tuples $(1,2,3)$ and $(1,2,4)$ in $sol(c)$, the constraint "$X + Y \leq Z$" yields the following rule:

$$xy(1,2) \quad \leftarrow \quad z(3), z(4)$$

However, this rule is made redundant by the preceding fact.

General case To every atom of $HB(K)$ is associated zero, one or more rules, each one being a possible reason yielding to the atom's removal. Let $h(a) \in HB(K)$ be an atom intended to be the head of a rule. The body of one of these rules is constituted of atoms of $HB(K)$ which support $h(a)$ via some constraint $c \in C$. Let us be more precise for 1R-consistency, in which every constraint $c \in C(h)$ can be a reason for the removal of $h(a)$. For nR-consistency, we should pick up all subsets of C of size n. So, since we only focus on 1R-consistency, let $c \in C(h)$. Since a constraint is identified with a set of atoms, $\{h(a)\}$ is the constraint containing the only atom $h(a)$. We denote by $c[h(a)]$ the selection $c \bowtie \{h(a)\}|_{var(c)}$ of c which matches with $h(a)$ on the corresponding variables. This corresponds to the interesting part of c for the possible removal of $h(a)$. In example 10, $c[x(1)]$ corresponds to the grey part in figure 3.

The following notion of support extends the classical notion introduced for the presentation of AC4 (Mohr & Henderson 1986) to cope with approximating CSPs and n-ary constraints. Let $h(a)$ be an atom, $c \in C(h)$ and $c(b)$ be an atom of $c[h(a)]$. For $k \in K(c)_{-h}$, the *k-support* of $h(a)$ relatively to $c(b)$ is the set of atoms $supp_k(h(a), c(b)) = k \bowtie \{c(b)\}|_{var(k)}$.

Definition 12 (Support) *The* support *of $h(a)$ relatively to $c(b)$ is the family:*

$$supp(h(a), c(b)) = \{supp_k(h(a), c(b)) \mid k \in K(c)_{-h}\}$$

In example 10, $supp_y(x(1), c(1,2,4)) = \{y(2)\}$ and $supp(x(1), c(1,2,4)) = \{\{y(2)\}, \{z(4)\}\}$. A k-support is a set and not only an atom because arities of constraints in C and K may overlap without being included (for some $k \in K$ and $c \in C$, $var(k) \backslash var(c) \neq \emptyset$). A rule corresponds to a reason for all supports to be invalidated. For this, and since a support of an atom is a set, it is enough to have one of these sets suppressed. Hence each rule models a possible way for every support to be removed. Let $Bod(h(a)) = \Pi_{c(b) \in c[h(a)]} supp(h(a), c(b))$. One element of $Bod(h(a))$ is a family giving for every atom $c(b)$ in $c[h(a)]$ an element of $supp(h(a), c(b))$, or in other terms, a $supp_k(h(a), c(b))$ for some $k \in K(c)_{-h}$. For every $B = (B_i)_{i \in c[h(a)]} \in Bod(h(a))$, we associate a rule:

$$h(a) \leftarrow \bigcup_{i \in c[h(a)]} B_i$$

In example 10, $Bod(x(1)) = \{\{y(2)\}, \{z(3)\}\} \times \{\{y(2)\}, \{z(4)\}\} \times \{\{y(3)\}, \{z(4)\}\}$ and the rules are those given above. The set of these rules is a (ground) logic program and its minimal model is the set of atoms removed by the consistency. By construction, to each atom in this model corresponds a proof-tree explaining its removal.

Procedure 1 RC4 Generation phase

Require: CSP C, approximating CSP K
Ensure: Set of propagation rules S

```
S = ∅
for all h(a) ∈ HB(K) do
  for all c ∈ C(h) do
    c[h(a)] := c ⋈ {h(a)}|_var(c)
    if c[h(a)] = ∅ then
      S := S ∪ {h(a) ←}
    else
      for all c(b) ∈ c[h(a)] do
        for all k ∈ K(c)_-h do
          supp_k(h(a), c(b)) := k ⋈ {c(b)}|_var(k)
        end for
      end for
      for all (B_1, ..., B_|c[h(a)]|) ∈
            Π_{c(b)∈c[h(a)]} supp(h(a), c(b)) do
        S := S ∪ {h(a) ← ∪_{i=1}^{|c[h(a)]|} B_i}
        Eliminate redundant rules
      end for
    end if
  end for
end for
```

The RC-4 algorithm

The basic idea of the algorithm is a generalization of AC4 (Mohr & Henderson 1986). The first part of AC4 is devoted to set up a complex structure recording and linking all supports for a variable's value. The role of this structure is played here by the set of rules. Hence the first part of the RC4 algorithm consists in the rule generation (see Procedure 1) according to the theory described above.

In the second part of AC4, the structure is used to perform propagation and this propagation is optimal in the sense that every variable's value is considered only once. In RC4, this second part is a standard forward-chaining algorithm. The version we give here uses the same trick as AC4 to be optimal: the use of counters to record the propagations already done. The idea of this part of the algorithm is to associate to each rule one counter initialized to the lenght of the rule's body and decremented whenever an atom of the body gets deleted. At the beginning, a propagation queue is initialized with the set of facts (atoms without support) and each time a counter reaches 0, the atom of the head is deleted from the constraint's domain and placed in the propagation queue. Remaining rules with the same head are also deleted. The process stoppes when the queue is empty. This part of the algorithm is presented in Procedure 2. The optimality of this part comes from the fact that each atom is touched only once. This can also be viewed as an instance of the linear algorithm to compute a transitive closure due to (Dowling & Gallier 1984). However, the general optimality comes from the non-redundancy of the rules obtained after elimination (see (Brand 2002) for instance).

Related Work

The idea of CSP approximation has been sketched in (Dao *et al.* 2002) to set a framework for solver learning. We present in this paper a more in-depth study of its properties. In

Procedure 2 RC4 Propagation phase

for all rule $l \leftarrow r \in S$ **do**
  $c_{l \leftarrow r} = |r|$ */* counter is initialized to the number of atoms in r */*
end for
$Q := \{h(a) \mid$ the rule $h(a) \leftarrow$ is in the set of rules$\}$
while $Q \neq \emptyset$ **do**
  take an atom $h(a)$ in Q
  for all rule $l \leftarrow r \in S$ where $h(a) \in r$ **do**
    $c_{l \leftarrow r} := c_{l \leftarrow r} - 1$
    if $c_{l \leftarrow r} = 0$ **then**
      with l being $k(b)$, remove the tuple b from $sol(k)$
      $Q := Q \cup \{k(b)\}$
      remove from S all rules having $k(b)$ as head
    end if
  end for
end while

this framework, the new generalization of mR-consistency arises naturally. For $m > 2$, its space complexity is probably too high to be practical but so is k-consistency. Most usual notions of local consistency are variable-based (Mohr & Henderson 1986; Han & Lee 1988; Mohr & Masini 1988; Cooper 1989) and suppose that variable domains are accessible. Dechter and Van Beek (Dechter & van Beek 1997) have proposed a first notion of consistency based on relations, but which is not fully relational in our sense because it still needs to consider arbitrary sets of variables. In contrast, in our framework of CSP approximations, relations are really first class citizen. For example, access to a variable is possible only if K owns a unary relation to represent this variable's domain. Finally, all consistencies can be formalized in this framework , though they may not have a natural expression in a given language, like, for example some hypothetico-deductive consistencies such as singleton-consistency (Debruyne & Bessière 2001).

Rule-based formalisms have been introduced in constraint programming since a long time, for example with indexicals (van Hentenryck, Saraswat, & Deville 1991; Codognet & Diaz 1996) and Constraint Handling Rules (Frühwirth 1998). In contrast, our rule system is very low-level, showing immediate connection between variable values or constraint tuples. This does not mean that a more abstract rule system cannot be derived, for example for consistencies which exhibit some regularities like arc-consistency (Ferrand & Lallouet 2002). But one interest of such an instantiated rule system is that any consistency can be represented at this level (since all consistencies exploit some connexion between variables). However, in our opinion, propagation rules have more to offer: they are a high-level description of low-level algorithms and this point deserves more attention for further works, notably in applications which involve a deeper understanding of the propagation such as explanations (Jussien 2001), constraint retraction or the automatic transformation of the rule system.

Recently, Apt and Monfroy have proposed a rule generation algorithm intended to enforce a consistency named rule-consistency (Apt & Monfroy 2001). They distinguish two types of rules: equality rules and membership rules. Equality rules are of the form $X_1 = a_1, \ldots, X_n = a_n \rightarrow X_0 \neq a_0$ and enforce a weak consistency since they only fire when domains of $(X_i)_{i \in [1..n]}$ are singletons. Membership rules are of the form $X_1 \in S_1, \ldots, X_n \in S_n \rightarrow X_0 \neq a_0$ and the set of such rules they give enforces arc-consistency. But since the head and the body of a rule are not in the same set (the head corresponds to a value removal and the body to values which are still in the domains), these rules are not easily chainable without a specialized algorithm.

Acknowledgements. The authors would like to thank Gérard Ferrand for valuable discussions. This research is supported by the french CNRS grant ATIP/2JE095.

References

Apt, K. R., and Monfroy, E. 2001. Constraint programming viewed as rule-based programming. *Theory and Practice of Logic Programming* 1(6):713 – 750.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65:179–190.

Brand, S. 2002. A note on redundant rules in rule-based constraint programming. In S. Abdennadher, T. Früwirth, A. W., ed., *Workshop on Rule-Based Constraint Reasoning and Programming*, 77–90.

Codognet, P., and Diaz, D. 1996. Compiling constraints in clp(fd). *Journal of Logic Programming* 27(3):185–226.

Cooper, M. C. 1989. An optimal k-consistency algorithm. *Artificial Intelligence* 41(1):89–95.

Dao, T. B. H.; Lallouet, A.; Legtchenko, A.; and Martin, L. 2002. Indexical-based solver learning. In van Hentenryck, P., ed., *International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *LNCS*, 541–555. Ithaca, NY, USA: Springer.

Debruyne, R., and Bessière, C. 2001. Domain filtering consistencies. *Journal of Artificial Intelligence Research* 14:205–230.

Dechter, R., and van Beek, P. 1997. Local and global relational consistency. *Theoretical Computer Science* 173(1):283–308.

Dowling, W. F., and Gallier, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming* 1(3):267–284.

Ferrand, G., and Lallouet, A. 2002. A logic program characterization of domain reduction approximations in finite domain csps. In Stuckey, P., ed., *International Conference on Logic Programming*, volume 2401 of *LNCS*, 478–479. Springer-Verlag. Poster.

Frühwirth, T. 1998. Theory and practice of constraint handling rules. *Journal of Logic Programming* 37(1-3):95–138.

Han, C.-C., and Lee, C.-H. 1988. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence* 36(1):125–130.

Jussien, N. 2001. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*.

Mohr, R., and Henderson, T. C. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28(2):225–233.

Mohr, R., and Masini, G. 1988. Good old discrete relaxation. In Kodratoff, Y., ed., *European Conference on Artificial Intelligence*, 651–656. Munich, Germany: Pitmann Publishing.

van Hentenryck, P.; Saraswat, V.; and Deville, Y. 1991. Constraint processing in cc(fd). draft.