UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

A Model-theoretic Semantics for CSP Consistencies

Gérard Ferrand and Arnaud Lallouet
Université d'Orléans, LIFO

Rapport N^o **2002-05**

A Model-theoretic Semantics for CSP Consistencies

Gérard Ferrand and Arnaud Lallouet

Université d'Orléans — LIFO
BP 6759 — F-45067 Orléans cedex 2
{Gerard.Ferrand | Arnaud.Lallouet}@lifo.univ-orleans.fr

Abstract. We provide here a characterization of the approximations computed during the resolution of finite domain CSPs by a declarative semantics of a CLP program. We consider that the CSP is solved by the classical interleaving of search and consistency computed by domain reduction. At first, we propose a definite CLP program which allows to characterize the computed approximation by a greatest fixed-point. This framework applies nicely to (hyper)arc-consistency and can be further extended to other consistencies including non-unary constraint reduction. Moreover, motivated by other approaches of CSP representations by logic programs with stable model semantics, the computed approximation is also represented in a natural way by the well-founded semantics of another (normal) CLP program.

Keywords : CSP; Approximation; Domain reduction; Constraint Programming; Constraint Logic Programming; Well-founded model.

1 Introduction

Constraint Satisfaction Problems [23, 16], or in short CSPs, are a powerful knowledge representation technique which combine the declarativity of a high level specification with the existence of powerful resolution techniques. Emerging in the mid-70's, AI techniques for solving CSPs, namely *consistency* maintenance, have since been successful on classical OR problems, especially on finite domains. These techniques essentially consist in the computation of an over-appproximation of the CSP's solutions by so-called *constraint propagation*, a process by which forbidden values communicate via the variables shared by constraints to remove new forbidden values across the CSP.

But as the problem size and the complexity of the solving algorithms increase, there is a crucial need for an in-depth understanding of this process. For example, chaotic iterations [2] is a high-level formalization of the operational semantics of the solvers. In this approach, a solver is modeled by a set of constraint reduction operators and their associated consistency by their greatest common fixed-point.

But for debugging [6], validation or optimization purposes, new tools are needed, tools which can help to isolate a buggy part or to focus on some specific aspect of the solving process. These tools include visualization which allows to present to the user a graphical version of the computation trace (by which one can easily be convinced that a local consistency maintenance is truly not a local process). Moreover, advanced solvers such as the Choco system [12] make heavy use of propagation events and dynamic operator scheduling, making therefore the following of a trace even more unfriendly. At last, classical search techniques, mostly *variable labeling* and *domain splitting* are interleaved with consistency maintenance in order to obtain completeness.

This paper's contribution takes place in this stream of work by proposing to represent by a declarative semantics of a CLP program the approximation computed by a consistency. But since the consistency is an over-approximation, it cannot be the standard least fixed-point semantics, nor its computation can be made by a top-down goal-directed procedure such as CSLD-resolution.

We first propose a definite CLP program P and show that the greatest fixed-point of its associated operator T_P coincides with the consistency's computed approximation. This formulation enjoys also a nice logical reading since it is also the greatest model of the completed program P^*. But since the solving process does not end with the first consistency enforcement, we also show that consistent states obtained after arbitrary labeling steps are also modeled by downward closures of T_P starting from suitably restricted interpretations.

A second CLP program P_{neg} is obtained by a transformation of P by classical De Morgan laws and allows to model the individual contribution of each propagation rule. But since the program has negations, the semantics which turned out to be useful is the well-founded semantics [24]. Its negative part expresses the deleted values in the variable domains, the ones which do not participate to any solution. The computed approximation is thus completely defined by the complement of the negative part of P_{neg}'s well-founded semantics. This allows to establish a deep link between traditional CSP solving methods based on consistency and the knowledge representation methods based on stable model semantics of logic programs since the well-founded semantics is the least 3-valued stable model [22]. A logic program representation for CSPs is proposed in [18] and also in [15] and we contribute to this line of work by representing in this paradigm, extended to 3-valued logic, not only the solutions, but also the whole computational process.

In contrast with the usual ways of describing solvers, namely with CHR [9] and indexicals [25], we provide a logical model-theoretic framework instead of an operational one.

In summary, these programs both propose a system of rules with a logical semantics to model a CSP's computed approximation. The paper is organized as follows: in section 2, preliminaries about CSP solving and the CLP scheme are given. Section 3 introduces a common framework for both of them and is devoted to the study of the first (definite) program and its semantics. Section 4 gives the normal program P_{neg} and studies its well-founded semantics.

2 Preliminaries

2.1 CSPs

For a family $a = (a_i)_{i \in I}$ and $J \subseteq I$, we denote by $a|_J = (a_i)_{i \in J}$ the restriction of a to the set of indices J. We denote by A^I the cartesian power $I \longrightarrow A$ and by $\mathcal{P}(A)$ the powerset of A.

Let V be a denumerable set of variables and D a domain. Note that the use of a many sorted domain is possible in this context but we limit ourselves to a single domain in this paper for clarity. A *constraint* is a pair $c = (W, T)$ where $W = var(c) \subseteq V$ is the *arity* of c and $T = sol(c) \subseteq D^{var(c)}$ is a table containing the tuples solution of c. A *constraint satisfaction problem*, or in short *CSP*, is a set C of constraints. Finite domain constraint solving classically involves a finite domain D, as well as a finite number of constraints (and so of variables). A *solution* is a tuple of D^V satisfying all constraints, i.e. $s \in sol(C) \iff \forall c \in C, s|_{var(c)} \in sol(c)$. Solving a CSP means finding one (or all, but this is tackled by the same methods) of its solutions. It could be done only by enumeration but the combination of search and approximation methods called *consistencies* yields far better performance in practice.

One of the most common approximation is the domain reduction scheme [26] in which variable domains are represented, totally or at least partially by intervals. The approximation consists in pruning *inconsistent* values from variable domains, i.e. those which are not supported by any constraint, according to the values still present in the domains of the constraint's other variables. The best known consistency is the so-called *arc-consistency* [14] which holds for a binary constraint, say $c(X, Y)$, if, for each value d in the domain of X, there is a corresponding couplevalue e in the domain of Y such that $c(d, e)$ holds (and conversely by exchanging the role of X and Y). This notion is extended to hyperarc-consistency for n-ary constraints in a straighforward manner. A whole CSP is arc-consistent if its variable domains are arc-consistent for all constraints. To be precise, variable domains are often considered as new unary constraints whose solutions are reduced. Thus, a *search state* is a family $(\mathcal{E}_X)_{X \in V}$ which belongs to the search space $\mathcal{P}(D)^V$. Each $\mathcal{E}_X$ represents the current domain of the variable X and the computed approximation is the cartesian product of this family of domains. A solution is found when every variable domain has been reduced to a singleton, that we call a *singletonic* search state. Conversely, if a variable domain becomes empty, then the CSP has no solution.

All known consistencies can be represented by the common closure of a set R of *reduction* operators, i.e. monotonic, correct and contracting operators [17, 4, 2]. For the arc-consistency with respect to $c(X, Y)$, it can be formalized by the mapping $r(\mathcal{E}) = \{d \in \mathcal{E}_X \mid \exists e \in \mathcal{E}_Y, c(d, e)\}$ and the one obtained by exchanging the role of X and Y. The set of operators associated to these mappings completely defines the expected consistency, which is obtained by computing their greatest common fixed-point. This can be done by defining an aggregate operator $T(\mathcal{E}) = \bigcap_{t \in R} t(\mathcal{E})$ and iterating it, or by a *chaotic iteration* [2], where only one operator of R is applied at each time, fairness of the scheduling ensuring the confluence. Correctness of the operators is required for not missing any solution. However, completeness, which is not required in general, has to hold for singleton sets. It is simply needed for the solver to discriminate solutions. This fact is of particular importance because a solver replaces the original constraints by the set of operators.

In this context, two extensions can be considered. Firstly, the current solutions of constraints of arbitrary arity can be reduced. This scheme is called the constraint reduction scheme [17, 2], of which the domain reduction scheme is a particular case. Secondly, arbitrary operators can be used, provided that they are correct, monotonic and contractant. All known consistencies can be modeled by such operators, but their expression in a given language may be problematic. Nevertheless, they can always be defined at least by a set of ground rules. We call R-consistency the stability by such an arbitrary set of rules R.

2.2 CLP

CLP(D) consists in replacing in Logic Programming unification by constraint solving on a domain D (pure LP is viewed as CLP(*Herbrand*) with the constraint =). This feature consists in defining a language to express the constraints and its integration into Logic Programming [11]. The constraint language is classically defined on a set of variables V and a signature containing a set Σ of function symbols and a set Π_C of constraint predicate symbols. A primitive constraint is an atomic formula $c(t_1, \dots t_n)$ where $c \in \Pi_C$ and the t_i's are (Σ, V)-terms. A constraint is a particular case of first-order (Π_C, Σ, V)-formula. The pre-interpretation of the constraints is given by a (Π_C, Σ)-structure $\mathcal{D}$ on the domain D.

Let Π_P be a set of program predicates symbols. An atom is an atomic (Π_P, Σ, V)-formula. The interpretation of a predicate p is given by a set of pseudo-atom called D-atom $p(d_1, \dots, d_n)$ where the d_i's are elements of D. The D-base is the set of all D-atoms and we denote it by $\mathcal{B}$. A $\mathcal{D}$-interpretation is identified to an arbitrary subset of the D-base. A valuation is a mapping from V to D.

A definite clause is a formula $h \leftarrow c, b_1, \dots, b_n$ where h and the b_i's are atoms and c is a constraint. A $\mathcal{D}$-instance of such a clause is a rule obtained by instanciation by a valuation satisfying the constraint c. A definite program is a set of definite clauses and is identified to the set of $\mathcal{D}$-instances of its clauses. To a given program P is associated an immediate consequence operator $T_P : \mathcal{P}(\mathcal{B}) \rightarrow \mathcal{P}(\mathcal{B})$ defined by $T_P(I) = \{h \mid \exists h \leftarrow B \in P, B \subseteq I\}$. This operator is monotonic and continuous and its least fixed point (in short lfp) is the least $\mathcal{D}$-model of P [11].

Following [11], we call definition of a predicate p the formula

$$\forall X_1, \dots, X_n \;\; p(X_1, \dots, X_n) \leftrightarrow B_1 \vee \dots \vee B_m$$

where for all clause $p(t_1, \dots, t_n) \leftarrow L_1, \dots, L_k$ defining p in P, B_i is the formula

$$\exists Y_1, \dots Y_j (\bigwedge_{i=1..n} X_i = t_i \;\wedge \bigwedge_{i=1..k} L_i)$$

where $Y_1, \ldots Y_j$ are the variables of the clause. The completion of P, denoted by P^*, is the set of the definitions of P's predicates. The greatest fixed point (in short gfp) of T_P is also the greatest $\mathcal{D}$-model of P^*.

A litteral is an atom a or the negation of an atom $\neg a$. If I is an interpretation, then $\neg I$ denotes the set $\{\neg a \mid a \in I\}$ and $\overline{I} = \mathcal{B} - I$. A normal clause is a clause whose body is composed of litterals instead of atoms. From this follows the notion of normal program. Various semantics for normal programs have been considered. We are particularily interested by two of them: the well-founded semantics [24] and the stable model semantics [10].

The well-founded semantics assigns to a program P a unique 3-valued model, denoted by WFS(P). Originally defined in terms of unfounded sets [24], we rather use a formulation [5] defined with an operator Ψ_P called *stability* operator. Again, P is identified with the set of its $\mathcal{D}$-instances. A partial (or 3-valued) interpretation I is a subset of $\mathcal{B} \cup \neg\mathcal{B}$ such that $I^+ = I \cap \mathcal{B}$ and $I^- = \{a \mid \neg a \in I\}$ are disjoint. In this context, the mapping T_P defined by $T_P(I) = \{h \mid \exists h \leftarrow B \in P, B \subseteq I\}$ is no longer an operator. Hence we define, for $I, J \subseteq \mathcal{B}$, the operator $T_{P,J}(I) = T_P(I \cup \neg J)$. Then the operator $\Psi_P : \mathcal{P}(\mathcal{B} \cup \neg\mathcal{B}) \rightarrow \mathcal{P}(\mathcal{B} \cup \neg\mathcal{B})$ can be defined:

$$\Psi_P(I) = \mathrm{lfp}(T_{P,I^-}) \cup \neg\, \overline{\mathrm{lfp}(T_{P,\overline{I^+}})}$$

The well-founded semantics WFS(P) is the least fixed-point lfp(Ψ_P).

The stable model semantics is based on the Gelfond-Lifschitz (or GL-) transformation of the program P with respect to a set of atoms $I \subseteq \mathcal{B}$. Let $GL(P, I)$ be the program obtained by removing from P any clause containing a litteral $\neg a$ with $a \in I$ and by removing all negative litterals from the body of the remaining clauses. The interpretation I is a stable model of P if I is the least model of $GL(P, I)$. In this case, I can be viewed as a consistent set of beliefs concerning the program P. In general, a program has more than one stable model, each of them being interesting. In [15], this is presented as a novel — very declarative — way of programming called *Stable Logic Programming*. A program is a collection of formulas whose purpose is to specify (or constrain) some class of models. For example, to have the atom g in every stable model with no perturbation of the semantics of g already defined by other clauses, we shall write the clause $f \leftarrow \neg g, \neg f$ where f is a new predicate symbol not appearing in P (and not by adding the fact $g \leftarrow$ which could interfere with other predicates in some stable models). This is usually written in this framework as the shorter integrity constraint $\leftarrow \neg g$ [15, 18]. Stable models and the well-founded semantics have strong connexions as the former is the least 3-valued stable model [22].

3 A logical formulation of CSP approximations

3.1 A common framework

We shall first proceed towards an integration of the two frameworks. To achieve this, we first express CSPs in the language of CLP. A CSP search state $\mathcal{E}$ is composed of a subset of the domain for each variable. In order to move towards a Logic Programming framework, we put in Π_P a *program* predicate symbol x for each variable X — and not a constraint predicate, which would remain at the constraint level —. With this bijection $x \leftrightarrow X$, we can identify the search state $\mathcal{E} = (\mathcal{E}_X)_{X \in V} \in \mathcal{P}(D)^V$ with the D-interpretation $\{x(d) \mid d \in \mathcal{E}_X\}$. For example, with two variables X and Y, if $\mathcal{E}$ is the search state $(X \mapsto \{0,1\}, Y \mapsto \{1,2\})$, then this family of domains is identified to the D-interpretation $\{x(0), x(1), y(1), y(2)\}$, also denoted by $\mathcal{E}$.

Since search states and interpretations are now related, we define what it means for an interpretation to be R-consistent. Let c be a constraint. A *local consistency rule* for c is a pair $r = (c, X)$ where $X \in var(c)$. It expresses that a value d of X's domain is supported by the constraint c according to other values present in the domain of the other variables. However, it does not prevent this value to be unsupported by another constraint. From now on, we consider that consistency operators are expressed by consistency rules and $R \subseteq \{(c, X) \mid c \in C, X \in var(c)\}$ and postpone the discussion on the use of more general operators.

Let C be a CSP and R be a set of local consistency rules, then we denote by $R(X) = \{r \in R \mid r = (c, X)\}$ the set of rules associated to X. Hence we have $R = \bigcup_{X \in V} R(X)$ and we assume that:

- $\forall X \in V, R(X) \neq \emptyset$, that is every variable appears at least in a rule.
- $\forall c \in C, \exists r = (c, X) \in R$, that is every constraint appears at least in a rule. Therefore, R fully determines the CSP C.

Definition 1 (r-consistent search state).
A search state $\mathcal{E}$ is r-consistent for $r = (c, X) \in R$, if:

$$\forall d \in \mathcal{E}_X, \exists s \in sol(c), d = s(X) \text{ and } \forall Y \in var(c), s(Y) \in \mathcal{E}_Y$$

A search state is R-consistent if r-consistent for each $r \in R$.

Definition 2 (Hyperarc-consistency).
The set of rules for hyperarc-consistency is $AC = \{(c, X) \mid c \in C, X \in var(c)\}$.

This coincides with arc-consistency for binary CSPs and with node-consistency for unary constraints. Note that our definition of R-consistency is weaker since not all rules may be present in R.

Example 3 (Arc-consistency). For example, let C be the following CSP composed of two constraints $k_1(X, Y)$ given by its solutions $\{(0,1), (1,2), (2,1)\}$ and $k_2(Y, Z)$ given by $\{(0,2), (1,1)\}$. Then AC is composed of the four following consistency rules $r_1 = (k_1, X), r_2 = (k_1, Y), r_3 = (k_2, Y)$ and $r_4 = (k_2, Z)$. The search state $\mathcal{E} = (X \mapsto \{0,1,2\}, Y \mapsto \{0,1,2\}, Z \mapsto \{1,2\})$ is not R-consistent because the value 0 in Y's domain does not meet r_2-consistency and the value 2 r_3-consistency. However, the search state $\mathcal{F} = (X \mapsto \{0,2\}, Y \mapsto \{1\}, Z \mapsto \{1\})$ is.

A local consistency rule is also a CLP($\mathcal{D}$) clause. Nevertheless, the set of such clauses defining the operators is not the expected program, but before stating it, we shall now be more precise concerning the used constraint language. Here and whatever is its exact syntactic definition in the constraint language, a constraint is only known by i) its symbol c, ii) its arity $var(c)$ and iii) its solutions $sol(c)$.

To a consistency rule $r = (c, X)$ may be associated a clause:

$$x(X) \leftarrow c \wedge \bigwedge_{Y \in var(c)} y(Y)$$

in the spirit of the clause representation of graphs used in [17], for which it is mentioned that it does not enjoy a logic program semantics. In fact, in order to model consistency, we need to get the intersection of the consequences of the rules, and not their union. This is done by introducing for each consistency rule r a new program predicate symbol of the same name r. This should be no confusing since they are not used in the same context. The D-base $\mathcal{B}$ now becomes $\{x(d) \mid X \in V, d \in D\} \cup \{r(d) \mid r \in R, d \in D\}$.

To a D-interpretation I is associated $I^{VAR} = (I_X^{VAR})_{X \in V} \in \mathcal{P}(D)^V$ where $I_X^{VAR} = \{d \in D \mid x(d) \in I\}$. For example, with $I = \{x(3), foo(1), y(0), y(4)\}$, we get $I^{VAR} = (X \mapsto \{3\}, Y \mapsto \{0, 4\})$. From now on, we shall use the letter I to denote an arbitrary D-interpretation and the letter $\mathcal{E}$ to denote a family of domains included in $\mathcal{B}^{VAR}$. According to its identification with an interpretation, $\mathcal{E}_X$ represents the notion of X's domain formalized by $\{d \mid x(d) \in \mathcal{E}_X\}$.

3.2 The program P and its semantics

In order to get the program P corresponding to the R-consistency, we define:

– for each $r = (c, X) \in R$,

$$\text{rule-clause}(r) \quad := \quad r(X) \leftarrow c \wedge \bigwedge_{Y \in var(c)} y(Y)$$

– for each variable $X \in V$,

$$\text{var-clause}(X) \quad := \quad x(X) \leftarrow \bigwedge_{r \in R(X)} r(X)$$

Definition 4 (CLP program associated to a consistency). *The program P associated to the set of consistency rules R is the CLP($\mathcal{D}$) program:*

$$\{\textit{rule-clause}(r) \mid r \in R\} \cup \{\textit{var-clause}(X) \mid X \in V\}$$

Intuitively, for $r = (c, X)$, rule-clause(r) collects the values maintained in X's domain according to c, and var-clause(X) allows this value only if maintained by all constraints. Note that since there is no fact (because a constraint involves at least one variable and because $R(X) \neq \emptyset$), the least $\mathcal{D}$-model of P is always empty, and thus of no interest. Let $T_P : \mathcal{P}(\mathcal{B}) \longrightarrow \mathcal{P}(\mathcal{B})$ be the monotonic operator associated to P.

Example 5 (example 3 continued).
The program P associated to the CSP C and arc-consistency is:

$$\begin{array}{lcl} rc_1 & := & r_1(X) \leftarrow k_1(X,Y), x(X), y(Y). \\ rc_2 & := & r_2(Y) \leftarrow k_1(X,Y), x(X), y(Y). \\ rc_3 & := & r_3(Y) \leftarrow k_2(Y,Z),\ y(Y),\ z(Z). \\ rc_4 & := & r_4(Z) \leftarrow k_2(Y,Z),\ y(Y),\ z(Z). \end{array} \qquad \begin{array}{lcl} vc_1 & := & x(X) \leftarrow r_1(X). \\ vc_2 & := & y(Y) \leftarrow r_2(Y),\ r_3(Y). \\ vc_3 & := & z(Z) \leftarrow r_4(Z). \end{array}$$

Lemma 6 (r-consistent interpretation).
Let $\mathcal{E} \subseteq \mathcal{B}^{VAR}$, $r = (c, X) \in R$. The interpretation $\mathcal{E}$ is r-consistent if

$$\forall d \in \mathcal{E}_X, r(d) \in T_P(\mathcal{E})$$

Proof. By rule-clause(r).

This can be viewed as an alternative definition of r-consistency. We denote by $R \downarrow \mathcal{E}$ the downward closure of $\mathcal{E}$ by R, which is the greatest R-consistent set included in $\mathcal{E}$. Such a set exists (take $\bigcup\{\mathcal{F} \subseteq \mathcal{E} \mid \mathcal{F}$ R-consistent$\}$). This closure is the approximation computed by the consistency.

Example 7 (example 3 continued). The interpretation $\mathcal{E}$ is not R-consistent since $r_3(2) \notin T_P(\mathcal{E})$.

Here follows two technical lemmas:

Lemma 8. *Let $I \subseteq \mathcal{B}$. If $I \subseteq T_P(I)$, then I^{VAR} is R-consistent.*

Proof. Assume that $I \subseteq T_P(I)$. Proving R-consistence consists in proving r-consistence for all $r \in R$. Let $r = (c, X) \in R$ and let $d \in I_X^{VAR}$. We have $x(d) \in I \subseteq T_P(I)$. Then, by var-clause($X$), we have $\{r(d) \mid r \in R(X)\} \subseteq I$. Hence $r(d) \in I \subseteq T_P(I)$. But, by rule-clause($r$), $r(d) \in T_P(I) \Longrightarrow r(d) \in T_P(I^{VAR})$.

Lemma 9. *Let $\mathcal{E} \subseteq \mathcal{B}^{VAR}$ and $\mathcal{E}$ R-consistent. Then $\mathcal{E} \cup T_P(\mathcal{E}) \subseteq T_P(\mathcal{E} \cup T_P(\mathcal{E}))$.*

Proof. First, we prove that $\mathcal{E} \subseteq T_P(T_P(\mathcal{E}))$. Let $x(d) \in \mathcal{E}$ and $r \in R(X)$. Since $\mathcal{E}$ is R-consistent, we have $r(d) \in T_P(\mathcal{E})$. Therefore, $\{r(d) \mid r \in R(X)\} \subseteq T_P(\mathcal{E})$ and, by var-clause(X), we have $x(d) \in T_P(T_P(\mathcal{E}))$. By monotony of T_P, $\mathcal{E} \subseteq T_P(T_P(\mathcal{E})) \subseteq T_P(\mathcal{E} \cup T_P(\mathcal{E}))$ and $T_P(\mathcal{E}) \subseteq T_P(\mathcal{E} \cup T_P(\mathcal{E}))$. Hence the lemma.

The following theorem states that the approximation computed by the consistency starting from the full domain for each variable is the greatest fixed-point of T_P.

Theorem 10.

$$R \downarrow \mathcal{B}^{VAR} = (gfp(T_P))^{VAR}$$

Proof. We use the fact that $\text{gfp}(T_P)$ is the greatest supported set by T_P. Thus, to prove that $E \subseteq \text{gfp}(T_P)$, it is sufficient to have $E \subseteq T_P(E)$.

- $\subseteq$: Let $J = R \downarrow \mathcal{B}^{VAR}$. By definition, J is R-consistent and $J \subseteq \mathcal{B}^{VAR}$. By lemma 9, we have $J \cup T_P(J) \subseteq T_P(J \cup T_P(J))$, hence $J \cup T_P(J) \subseteq \text{gfp}(T_P)$ and therefore $J \subseteq \text{gfp}(T_P)$. But $J \subseteq \mathcal{B}^{VAR}$ and so, $J \subseteq (\text{gfp}(T_P))^{VAR}$.
- $\supseteq$: Let $I = \text{gfp}(T_P)$. I is the greatest set such that $I \subseteq T_P(I)$. Hence, by lemma 8, I^{VAR} is R-consistent, so $I^{VAR} \subseteq R \downarrow \mathcal{B}^{VAR}$.

Example 11 (example 3 continued). Let $D = \{0, 1, 2\}$. The downward iterations of T_P are detailed as follows, where the notation $p(a; b)$ stands for $p(a), p(b)$:

- $T \downarrow^1 \mathcal{B} = \mathcal{B}^{VAR} \cup \{r_1(0;1;2), r_2(1;2), r_3(0;1), r_4(1;2)\}$
- $T \downarrow^2 \mathcal{B} = \{x(0;1;2), y(1), z(1;2)\} \cup \{r_1(0;1;2), r_2(1;2), r_3(0;1), r_4(1;2)\}$
- $T \downarrow^3 \mathcal{B} = \{x(0;1;2), y(1), z(1;2)\} \cup \{r_1(0;2), r_2(1), r_3(1), r_4(1)\}$
- $T \downarrow^4 \mathcal{B} = \{x(0;2), y(1), z(1)\} \cup \{r_1(0;2), r_2(1), r_3(1), r_4(1)\} = \text{gfp}(T_P)$.

In this example, $\text{gfp}(T_P))^{VAR} = \{x(0), x(2), y(1), z(1)\}$ which is exactly the values obtained by arc-consistency.

3.3 Relation with the CSP's solutions

The usual solving process of a CSP does not end as the first closure is found. Usually, a variable is chosen and its domain is partitionned in two (domain splitting) or its values are enumerated (labeling) until a solution is found. We do not model directly the labeling but the following result aims at characterizing the approximation obtained when starting from an arbitrary search point encountered during the search. We recall that a solution for a CSP is a single tuple satisfying all constraints. In the domain reduction framework, it is represented by a *singletonic* search state or interpretation:

Definition 12. *An interpretation $\mathcal{E} \subseteq \mathcal{B}^{VAR}$ is* singletonic *if for all variable $X \in V$, there exists an unique $x(d) \in \mathcal{E}$.*

Thus, for a singletonic interpretation $\mathcal{E}$, there exists a tuple $s_{\mathcal{E}} : V \to D$ such that $\mathcal{E} = \{x(s_{\mathcal{E}}(X)) \mid X \in V\}$. The following lemma states that R is a complete solver for singletonic interpretations:

Lemma 13. *Let $\mathcal{E}$ be a singletonic interpretation. Then $\mathcal{E}$ is R-consistent if and only if $s_{\mathcal{E}} \in sol(C)$.*

In theory, this property could hold for larger search states s such that $\Pi s \subseteq sol(c)$ but usually solvers are unable to detect this case. This justify the interest in singletonic search states and interpretations.

We denote by $T_P \downarrow I = \max\{J \subseteq I \mid J \subseteq T_P(J)\}$ the downward closure of $I \subseteq \mathcal{B}$ by T_P. After having computed a closure, the domain is further reduced by the search mechanism and a new consistency calculation is launched. The following theorem states that the closure computed by consistency starting from an arbitrary search state is also modeled by the downward closure of T_P starting from a suitable interpretation. This interpretation is defined from the search state to which it is added a part for the atoms of the form $r(d)$.

Theorem 14.
For $\mathcal{E} \subseteq \mathcal{B}^{VAR}$,

$$R \downarrow \mathcal{E} = [T_P \downarrow (\mathcal{E} \cup T_P(\mathcal{E}))]^{VAR}$$

Proof.

- $\subseteq$: Let $J = R \downarrow \mathcal{E}$. By definition, $J \subseteq \mathcal{B}^{VAR}$, $J \subseteq \mathcal{E}$ and J is R-consistent. By lemma 9, we have $J \cup T_P(J) \subseteq T_P(J \cup T_P(J))$. Besides, $J \cup T_P(J) \subseteq \mathcal{E} \cup T_P(\mathcal{E})$. Therefore, by definition of $T_P \downarrow$, we have $J \cup T_P(J) \subseteq T_P \downarrow (\mathcal{E} \cup T_P(\mathcal{E}))$. Since $J \subseteq \mathcal{B}^{VAR}$, we have $J \subseteq [T_P \downarrow (\mathcal{E} \cup T_P(\mathcal{E}))]^{VAR}$.
- $\supseteq$: Let $I = T_P \downarrow (\mathcal{E} \cup T_P(\mathcal{E}))$. By definition, $I \subseteq \mathcal{E} \cup T_P(\mathcal{E})$. But $I^{VAR} \cap T_P(\mathcal{E}) = \emptyset$ because $\mathcal{E} \subseteq \mathcal{B}^{VAR}$ and so $I^{VAR} \subseteq \mathcal{E}$. Besides, by definition of downward closure, $I \subseteq T_P(I)$. Hence, by lemma 8, I^{VAR} is R-consistent. Therefore, by definition of $R \downarrow \mathcal{E}$, we have $I^{VAR} \subseteq R \downarrow \mathcal{E}$.

According to the fixed-point theory, this closure is not always a fixed-point, and thus cannot be obtained by an iteration of T_P. Indeed, it can be obtained by iterating the contractant operator associated to T_P, i.e. T'_P defined by $T'_P(I) = I \cap T_P(I)$. The following example illustrates this theorem.

Example 15. Let C be a CSP composed of $c_1(X,Y)$ given by $\{(0,0),(0,1),(1,1),(1,2)\}$ and $c_2(Y,Z)$ by $\{(1,2),(2,1),(2,2)\}$. The initial domain is $D = \{0,1,2\}$ and we name the rules $(c_1,X),(c_1,Y),(c_2,Y)$ and (c_2,Z) respectively by xc_1, yc_1, yc_2, zc_2.

A first closure gives the search state $(X \mapsto \{0,1\}, Y \mapsto \{1,2\}, Z \mapsto \{1,2\})$. If we further restrict the domain of Y to 1 by labeling, we can compute the closure of this new domain $\mathcal{E}$. We have $\mathcal{E}' = \mathcal{E} \cup T_P(\mathcal{E}) = \{x(0;1), y(1), z(1;2)\} \cup \{xc_1(0;1), yc_1(1), yc_2(1), zc_2(2)\}$.

$$T \downarrow^1 \mathcal{E}' = \{x(0;1), y(1), z(2)\} \cup T_P(\mathcal{E}) = T_P \downarrow (\mathcal{E} \cup T_P(\mathcal{E}))$$

Again, this is the closure computed by arc-consistency.

3.4 Discussion

The interest for a declarative representation of consistency in LP style was already a concern of [17] since they use a clause notation to represent graph operations. The conclusion of this paper is illuminating since it is said that "the correspondence [between relaxation and Logic Programming] is not complete [...]. For example, a set of relaxation rules for a network does not constitute a logic program from a semantics point of view; [...]". This comes from the fact that all their rules with the same head contribute to a common reduction and are thus linked by a "and" instead of the "or" of Logic Programs".

If we only want to model the monotonic deleting of values in the variables' domains, then these rules, or the monotonic operator associated to an arbitrary consistency can be exploited in the following way. First, note that for any operator $T : \mathcal{P}(\mathcal{B}) \to \mathcal{P}(\mathcal{B})$, there exists its dual [1] $\widetilde{T} : \mathcal{P}(\mathcal{B}) \to \mathcal{P}(\mathcal{B})$ defined by

$\widetilde{T}(I) = \overline{T(\overline{I})}$. If T is monotonic, so is $\widetilde{T}$ and $\mathrm{lfp}(\widetilde{T}) = \overline{\mathrm{gfp}(T)}$. Then for each rule $r = (c, X) \in R$, we can associate an operator T_r defined by $T_r(\mathcal{E}) = \{x(d) \mid \exists s \in sol(c), d = s(X)$ and $\forall Y \in var(c), y(s(Y)) \in \mathcal{E}\} \cup \{z(d) \mid z \neq x, d \in D\}$. Intuitively, T_r is the correct and monotonic operator associated to a consistency rule r.

Lemma 16.
$$\mathcal{E} \subseteq T_r(\mathcal{E}) \Longleftrightarrow \mathcal{E} \textit{ is } r\textit{-consistent}$$

The dual operator $\widetilde{T_r}$ models the values deleted from the variable's domain. Let T_R be the operator associated to the whole set of rules R: $T_R(\mathcal{E}) = \bigcap_{r \in R} T_r(\mathcal{E})$. It follows that $\mathcal{E} \subseteq T_R(\mathcal{E})$ means that $\mathcal{E}$ is R-consistent.

Lemma 17.
$$\widetilde{T_R}(I) = \bigcup_{r \in R} \widetilde{T_r}(I)$$

Proof.
$$\widetilde{T_R}(I) = \overline{T_R(\overline{I})} = \overline{\bigcap_{r \in R} T_r(\overline{I})} = \bigcup_{r \in R} \overline{T_r(\overline{I})} = \bigcup_{r \in R} \widetilde{T_r}(I)$$

We can characterize the computed approximation using T_R:

Lemma 18.
$$R \downarrow B^{VAR} = \mathit{gfp}(T_R)$$

Proof. By definition of $R \downarrow B^{VAR}$ and T_R.

It follows that $\overline{R \downarrow B^{VAR}} = \mathrm{lfp}(\widetilde{T_R})$, and therefore the negative information is completely characterized by the operators $\widetilde{T_r}$.

Because the operators $\widetilde{T_r}$ express the direct cause of a removal, they are interesting for the implementation of solvers. But for more advanced uses such as to model a computation trace [6], they have a major drawback: they are not expressed in general by a simple system of rules. In fact, they can always be defined by a system of ground rules but not by rule schemata like in our formalism. Hence they do not enjoy a non-trivial logical reading as our program P. Moreover, their definition makes use of choice functions across the cartesian product of the rules' body indexed by the rules' name and thus a rule's body may as large as the domain's size. For example some rule bodies associated to the constraint $X < Y$ are as large as the domain's size. Their use for debugging has nevertheless been tackled in [8].

4 Approximations and well-founded model

The greatest fixed-point semantics of P allows to express the computed approximation. In this section, we enhance this framework by introducing a program with negation which allows to characterize more directly the values removed by consistency, takes into account the CSP's solutions and benefits of a more intuitive reading.

4.1 The program P_{neg} and its semantics

Let us transform the program P into P_{neg} in order to isolate the contribution of each rule r. For this, we introduce a new program predicate symbol nx for each variable X. Its meaning is that $nx(d)$ is true if the value d has been removed from the domain of x.

Definition 19 (P_{neg}). *Let P_{neg} be the following CLP($\mathcal{D}$) program:*

$$\begin{array}{ll} x(X) \leftarrow \neg nx(X). & \textit{for each } X \in V \\ nx(X) \leftarrow \neg r(X). & \textit{for each } r \in R(X) \\ \textit{rule-clause}(r) & \textit{for each } r \in R \end{array}$$

The intuitive meaning of this program is closer to the idea of consistency: if a value d for a variable X is unsupported by some constraint c, then this value has to be removed. It follows from P_{neg} that $nx(d)$ is true and $x(d)$ false. Because of the semantics of programs with negation, the semantics of this program deserves a study. The well-founded semantics expresses nicely the useful notion of "false if the negation is proved".

Example 20 (example 3 continued). The program P_{neg} associated to the CSP C and arc-consistency is the following:

$$\begin{array}{ll} x(X) \leftarrow \neg\, nx(X). & \\ y(Y) \leftarrow \neg\, ny(Y). & \\ z(Z) \leftarrow \neg\, nz(Z). & r_1(X) \leftarrow k_1(X,Y), x(X), y(Y). \\ nx(X) \leftarrow \neg\, r_1(X). & r_2(Y) \leftarrow k_1(X,Y), x(X), y(Y). \\ ny(Y) \leftarrow \neg\, r_2(Y). & r_3(Y) \leftarrow k_2(Y,Z),\ y(Y),\ z(Z). \\ ny(Y) \leftarrow \neg\, r_3(Y). & r_4(Z) \leftarrow k_2(Y,Z),\ y(Y),\ z(Z). \\ nz(Z) \leftarrow \neg\, r_4(Z). & \end{array}$$

Let $\mathcal{B}_{neg} = \mathcal{B} \cup \{nx(d) \mid X \in V, d \in D\}$. As the least fixed-point of the operator Ψ, the well-founded semantics $WFS(P_{neg})$ is the least set $W \subseteq \mathcal{B}_{neg} \cup \neg\mathcal{B}_{neg}$ such that $\Psi(W) \subseteq W$. Following the definition of P_{neg}, $\Psi(W) \subseteq W$ is equivalent to:

$$\begin{array}{rlr} x(d) \in W^+ & \Leftarrow nx(d) \in W^- & (1) \\ nx(d) \in W^+ & \Leftarrow \exists r \in R(X), r(d) \in W^- & (2) \\ r(d) \in W^+ & \Leftarrow \exists s \in sol(c), d = s(X), \forall Y \in var(c), y(s(Y)) \in W^+ & (3) \\ x(d) \in \overline{W^-} & \Rightarrow nx(d) \in \overline{W^+} & (4) \\ nx(d) \in \overline{W^-} & \Rightarrow \exists r \in R(X), r(d) \in \overline{W^+} & (5) \\ r(d) \in \overline{W^-} & \Rightarrow \exists s \in\in sol(c), d = s(X), \forall Y \in var(c), y(s(Y)) \in \overline{W^-} & (6) \end{array}$$

Moreover, since it is a fixed-point, $WFS(P_{neg})$ also verifies the conditions obtained by replacing $\Leftarrow$ and $\Rightarrow$ by $\Leftrightarrow$. Since this program is relatively simple, we are able to fully describe its well-founded semantics. Let us consider the candidate $W \subseteq \mathcal{B}_{neg} \cup \neg\mathcal{B}_{neg}$ defined as follows:

$$\begin{aligned} W^+ &= \{nx(d) \in \mathcal{B}_{neg} \mid x(d) \notin \mathrm{gfp}(T_P)\} \\ W^- &= \{x(d) \in \mathcal{B}_{neg} \mid x(d) \notin \mathrm{gfp}(T_P)\} \cup \\ &\quad \{r(d) \in \mathcal{B}_{neg} \mid r(d) \notin \mathrm{gfp}(T_P)\} \end{aligned}$$

The two following lemmas and the theorem state that W is the well-founded semantics of P_{neg}.

Lemma 21.

$$\Psi(W) \subseteq W$$

Proof. W verifies the six above conditions:

1. there is no $nx(d) \in W^-$.
2. for $r \in R(X)$, we have $x(d) \in \mathrm{gfp}(T_P) \Rightarrow r(d) \in \mathrm{gfp}(T_P)$ because of var-clause(X).
3. there is no $x(d) \in W^+$.
4. trivial since $x(d) \in \mathrm{gfp}(T_P) \Rightarrow x(d) \in \mathrm{gfp}(T_P)$.
5. we have $\forall X, R(X) \neq \emptyset$. Hence $nx(d) \in \overline{W^-}$ and $r(d) \in \overline{W^+}$ are always true.
6. we have $r(d) \in \mathrm{gfp}(T_P) \Rightarrow \exists s \in\in sol(c), d = s(X), \forall Y \in var(c), y(s(Y)) \in \mathrm{gfp}(T_P)$ because of rule-clause(r).

Corollary 22.

$$WFS(P_{neg}) \subseteq W$$

Lemma 23.

$$\mathcal{B} \cap \overline{WFS(P_{neg})^-} \subseteq T_P(\mathcal{B} \cap \overline{WFS(P_{neg})^-})$$

Proof. We have two cases to consider:

– consider $x(d) \in \overline{WFS(P_{neg})^-}$. By condition 4, $nx(d) \in \overline{WFS(P_{neg})^+}$. Hence by the equivalence of condition 2, which holds because $WFS(P_{neg})$ is a fixed-point, we have, $\forall r \in R(X)$, $r(d) \in \overline{WFS(P_{neg})^-}$ and therefore $x(d) \in T_P(\mathcal{B} \cap \overline{WFS(P_{neg})^-})$.
– consider $r(d) \in \overline{WFS(P_{neg})^-}$. By condition 6, and rule-clause(r), we have $r(d) \in T_P(\mathcal{B} \cap \overline{WFS(P_{neg})^-})$.

Corollary 24.

$$\mathcal{B} \cap \overline{WFS(P_{neg})^-} \subseteq \mathit{gfp}(T_P)$$

Theorem 25 (Well-founded semantics of P_{neg}).

$$WFS(P_{neg}) = W$$

Proof.

- $WFS(P_{neg})^+ = W^+$. By corollary 22, we have $WFS(P_{neg})^+ \subseteq W^+$. To prove the reverse inclusion, consider d such that $x(d) \notin \mathrm{gfp}(T_P)$. By var-clause($X$), we have $x(d) \in \mathrm{gfp}(T_P) \Leftrightarrow \forall r \in R(X), r(d) \in \mathrm{gfp}(T_P)$. Thus, if $x(d) \in \mathrm{gfp}(T_P), \exists r \in R(X)$ such that $r(d) \notin \mathrm{gfp}(T_P)$, or $r(d) \in WFS(P_{neg})^-$. Then, because $WFS(P_{neg})$ verifies the condition 2, $nx(d) \in WFS(P_{neg})^+$.
- $WFS(P_{neg})^- = W^-$. By corollary 22, we have $WFS(P_{neg})^- \subseteq W^-$. By corollary 24, we have $x(d) \notin WFS(P_{neg})^- \Rightarrow x(d) \in \mathrm{gfp}(T_P)$ and $r(d) \notin WFS(P_{neg})^- \Rightarrow r(d) \in \mathrm{gfp}(T_P)$. Hence $x(d) \notin \mathrm{gfp}(T_P) \Rightarrow x(d) \in WFS(P_{neg})^-$ and the same for $r(d)$.

Corollary 26.

$$\mathcal{B} \cap \overline{WFS(P_{neg})^-} = \mathit{gfp}(T_P)$$

4.2 Discussion

The transformation from P to P_{neg} is quite simple, it can be viewed as an application of De Morgan laws. If there were an universal quantifier instead of a and, the body of var-clause(X) could also be transformed by the Lloyd-Topor transformation [13]. Also, P_{neg} has several properties, namely call-consistency (see [3], def. 4.5) and its stable models coincide with the Herbrand models of its completed (see [3], cor. 6.19).

From a logical point of view, the definite program P allows to express the consistency as the greatest $\mathcal{D}$-model of P^*. But the well-founded semantics allows to express more explicitely that the approximation consists in deleting values from the variable domains. For a litteral l, being in $WFS(P_{neg})$ means to be proved, either in a positive or negative way. This is meaningful since:

- $nx(d)$ is proved means that $\neg\, x(d)$ is proved, which holds if there exists a rule $r \in R(X)$ such that $\neg\, r(d)$ is proved, or in other words, if the value d has been rejected from X's domain because of r.
- $\neg\, r(d)$ is proved means that for all solution s of c, such that $d = s(X)$, there exists $Y \in var(c)$ such that $\neg\, y(s(Y))$ is proved. This reflects exactly constraint propagation.

As for the definite program P, the CSP's solutions are expressed by models of P_{neg}. These models are the singletonic stable models, which extend in the information ordering the well-founded model. This property is stated in lemma 27 hereafter. Let W be a bi-valued D-interpretation such that $\Psi(W) = W$. Then, by reading the conditions 1 and 2, we get:

$$x(d) \in W^+ \Leftrightarrow nx(d) \notin W^+ \Leftrightarrow \forall r \in R(X), r(d) \in W^+$$

Let $\mathcal{E}(W) = \mathcal{B}^{VAR} \cap W^+$ be the part of W which concerns the variables' domains.

Lemma 27.

1. *Let W be a bi-valued interpretation such that $\Psi(W) = W$. If $\mathcal{E}(W)$ is singletonic, then $\mathcal{E}(W)$ is solution of C.*

2. *Let $\mathcal{E}$ be a singletonic D-interpretation solution of C. Then W defined by $W^+ = \mathcal{E} \cup \{nx(d) \mid x(d) \notin \mathcal{E}\} \cup \{r(d) \mid r = (c, X), x(d) \in \mathcal{E}\}$ and $W^- = \overline{W^+}$ is such that $\Psi(W) = W$.*

Proof.

1. by condition 3.
2. W verifies the 6 equivalences corresponding to condition 1 to 6.

Since stable models are bi-valued models and stable by Ψ, this proves the correspondence.

On the other side, there is a considerable recent work in the area of stable models in order to model CSPs either directly or from a knowledge representation point of view [19, 15, 18, 20, 21]. For example, [18] proposes a translation from a CSPs to a normal logic program whose stable models are the CSP's solutions. Our program shares the same property with the additional one of following the approximation mechanism. This last feature proves the strength of this approach, if necessary. With the translation of [18] and on the examples we tried, the well-founded semantics of the translated programs is empty. It follows not surprinsingly from this cross-fertilization that some CSP techniques are shared with the ones used in stable model systems such SMODELS [19].

4.3 Implementation

We have implemented in Gnu-Prolog [7] a small translator called `csp2smodels` from CSPs to normal logic programs in order to feed the SMODELS [19] system. Here we present and illustrate experimentally our results.

A CSP is written in a simple syntax: each variable X must have a domain, which is expressed as an interval by `domain(x,`*min*`,`*max*`)` or given by a list of values `domain(x,[`v_1`, ..., `v_n`])`. A constraint may be described by a set of solution tuples by `constraint(`*name*`,[`*list of variables*`],[`*list of tuples*`])` or may be a SMODELS built-in, for example `constraint(eq,[x,y])` denotes $X = Y$.

Example 28 (example 3 continued).
Our running example is written as follows:

```
domain(x,0,2).
domain(y,0,2).
domain(z,0,2).

constraint(k1,[x,y],[[0,1],[1,2],[2,1]]).
constraint(k2,[y,z],[[0,2],[1,1]]).
```

Here is an excerpt from the output program. First variable ranges are declared for SMODELS' grounding procedure:

```
domx(0..2).
#domain domx(X).
#hide domx(X).
```

User constraints are stated in extension:

```
k1(0,1).
k1(1,2).
k1(2,1).
#hide k1(X,Y).
```

Then the main program is written out, with for example rule-clause(k_1, X):

```
k1x(X) :- k1(X,Y), x(X), y(Y).
#hide k1x(X).
```

The variable Y is involved in two constraints appears negatively in two rules:

```
ny(Y) :- not k1y(Y).
ny(Y) :- not k2y(Y).
#hide ny(Y).
```

Then the last type of clause which ensure the stability of the models:

```
x(X) :- not nx(X).
```

Note that auxiliary names are hidden to focus on the useful information in the models, namely the predicates $x(X)$. The well-founded semantics, computed with `smodels -w` is the approximation computed by arc-consistency:

```
smodels version 2.26. Reading...done
Well-founded model:
Positive part:
Negative part: x(1) y(2) z(0) z(2) y(0)
```

The SMODELS system also finds the CSP's solutions as singletonic interpretations (this is computed with `smodels 0`):

```
smodels version 2.26. Reading...done
Answer: 1
Stable Model:
Answer: 2
Stable Model: x(2) y(1) z(1)        // singletonic
Answer: 3
Stable Model: x(0) x(2) y(1) z(1)
Answer: 4
Stable Model: x(0) y(1) z(1)        // singletonic
False
```

On an other example, the n-queens problem, the computed well-founded semantics reflects that this CSP is already arc-consistent. The translator may be downloaded from the authors' web page.

5 Conclusion

In this paper, we presented a CLP logical scheme which allows to reflect the approximation made by arc-consistency in a declarative way. We first provide a definite CLP program and proved that the greatest fixed-point of its associated operator is the computed approximation. Then we transform it into an intuitive normal program and we show that the values removed by consistency are in the negative part of its well-founded semantics. Moreover, the original CSP's solutions are given by a certain class of stable models we call singletonic.

There are two possible generalisations to this framework: the extension to general constraint reduction is straightforward since the $\mathcal{D}$-base style we borrowed from CLP generalizes to n-ary predicates easily. It can be used to represent stronger consistencies like path-consistency. The second one concerns the use of more general operators to model weaker consistencies like the partial arc-consistency, which contracts only the bounds of the variables' domains represented as an interval [7]. Since any monotonic operator can be represented by a system of (ground) rules, we can consider using these rules instead of rule-clause(r). Our theory still holds in this context, but we loose the compact representation of clauses as the body of var-clause(X) would contain one atom by ground rule.

Let us recall the strong interest in propagation analysis and debugging which motivate our work. By using the program P, or better P_{neg} as a support for propagation analysis, we currently investigate the topic of building an user-understandable explanation of some unexpected success or failure in the set of computed solutions of a CSP. Moreover, by showing connections between CSP approximations and 3-valued stable models, this work contributes to bridge the gap between traditional ways of solving CSPs by consistency and the knowledge representation paradigm of logic programming with stable model semantics.

References

1. Peter Aczel. *An introduction to inductive definitions*, chapter 7, pages 739–782. Handbook of mathematical logic. North Holland, 1977.
2. K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999.
3. K.R. Apt and R.N. Bol. Logic programming and negation : a survey. *Journal of Logic Programming*, 19/20:9–71, 1994.
4. Frédéric Benhamou. Heterogeneous constraint solving. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *Algebraic and Logic Programming, 5th International Conference, ALP'96*, volume 1139 of *lncs*, pages 62–76, Aachen, Germany, sep 1996. Springer.
5. Staffan Bonnier, Ulf Nilsson, and Torbjörn Näslund. A simple fixed point characterization of three-valued stable model semantics. *Information Processing Letters*, 40(2):73–78, 1991.
6. Pierre Deransart, Manuel V. Hermenegildo, and Jan Maluszynski, editors. *Analysis and Visualization Tools for Constraint Programming, Constraint Debugging (DiSCiPl project)*. Number 1870 in LNCS. Springer, 2000.

7. Daniel Diaz and Philippe Codognet. The GNU Prolog system and its implementation. In *ACM Symposium on Applied Computing*, volume 2, pages 728–732, Villa Olmo, Como, Italy, 2000.
8. G. Ferrand, W. Lesaint, and A. Tessier. Theoretical foundations of value withdrawal explanations in constraints solving by domain reduction. Research Report RR-LIFO-2001-05, LIFO, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, BP 6759, F-45067 Orléans Cedex 2, 2001.
9. Thom Früwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, 1998.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *International Joint Conference and Symposium on Logic Programming*, pages 1070–1080, Seattle, 1988. IEEE.
11. Joxan Jaffar, Michael Maher, Kim Marriott, and Peter Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
12. François Laburthe and the OCRE project. Choco: implementing a CP kernel. In *TRICS, Techniques foR Implementing Constraint programming Systems, CP 2000 post-conference workshop*, Technical report TRA9/00, Singapore, sep 2000.
13. John W. Lloyd and Rodney W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.
14. Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
15. Victor W. Marek and Mirosław Truszczyński. *Stable Models and an Alternative Logic Programming Paradigm*, pages 375–378. Artificial Intelligence. Springer-Verlag, 1999.
16. Kim Marriott and Peter Stuckey. *Programming with Constraints: an introduction*. MIT Press, 1998.
17. Ugo Montanari and Francesca Rossi. Constraint relaxation may be perfect. *Artificial Intelligence*, 48:143–170, 1991.
18. Ilkka Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
19. Ilkka Niemelä and Patrick Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *International Conference on Logic Programming and Non Monotonic Reasoning*, pages 420–429, Dagstuhl, Germany, July 1997. Springer-Verlag.
20. Nikolay Pelov and Maurice Bruynooghe. Extending constraint logic programming with open functions. In Maurizio Gabbrielli and Frank Pfenning, editors, *Principles and Practice of Declarative Programming*, pages 235–244, Montreal, Canada, 2000. ACM.
21. Nikolay Pelov, Emmanuel De Mot, and Marc Denecker. Logic programming approaches for representing and solving constraint satisfaction problems : A comparison. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning*, volume 1955 of *LNCS*, pages 225–239, Reunion Island, France, November 11-12 2000. Springer.
22. T. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae*, XIII:445–463, 1990.
23. Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
24. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
25. P. van Hentenryck, V. Saraswat, and Y. Deville. Constraint processing in cc(fd). draft, 1991.

26. Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.