A Logical Analysis of Boolean Constraints

Krzysztof R. Apt

Abstract

In Apt (1998) we provided a proof theoretic account of constraint programming. Here we show how it can be used to analyse Boolean constraints. More precisely, we show here how a Boolean constraint solver based on the look-ahead search strategy can be defined in a purely logical way. To this end we characterize arc consistency for Boolean constraints by proof theoretic means. As a byproduct we clarify the status of the proof rules introduced in Codognet & Diaz (1996) that form a basis of their Boolean constraint solver. These considerations lead to a simple Boolean constraint solver that generates all solutions to a given set of Boolean constraints. It performs well on various benchmarks.

Contents

1 Introduction

The research on constraint satisfaction problems led to an identification of several important and useful techniques that can be put to practice to solve various combinatorial and optimization problems.

These techniques include algorithms for various forms of local consistency, several powerful search techniques and formalization of the problems under consideration by means of customized constraint primitives. For an overview of this area see Tsang (1993).

The aim of this paper is to show how in case of Boolean constraints some of these techniques can be explained by purely logical means. To this end we use the proof theoretic approach to constraint programming introduced in Apt (1998) and customize it to Boolean constraints. Then we provide an axiomatic characterization of arc consistency for Boolean constraint satisfaction problems. Also, we compare this axiomatization with a similar one presented in Codognet & Diaz (1996) and used there as a basis for an efficient implementation of a Boolean constraint solver.

This, when combined with a use of proof trees, allows us to explain in logical terms the look-ahead search strategy generalized to n-ary constraints. These ideas lead to a natural implementation of a Boolean constraint solver that performs well on various benchmarks.

The use of local consistency methods to deal with Boolean constraints essentially goes back to Stallman & Sussman (1977). Some more recent articles can be found in Benhamou & Colmerauer (1993). The method discussed in this paper is known in the literature as *unit propagation* (see, e.g. Dalal (1992)). However, to our knowledge, no systematic study of it in purely logical terms has been provided.

2 Preliminaries

2.1 Constraint Satisfaction Problems

Consider a finite sequence of variables $X := x_1, \ldots, x_n$ where $n \geq 0$, with respective domains $\mathcal{D} := D_1, \ldots, D_n$ associated with them. So each variable x_i ranges over the domain D_i. By a *constraint* C on X we mean a subset of $D_1 \times \ldots \times D_n$. If C equals $D_1 \times \ldots \times D_n$ then we say that C is *solved*.

In the boundary case when $n = 0$ we admit two constraints, denoted by $\top$ and $\bot$, that denote respectively the *true constraint* (for example $0 = 0$) and the *false constraint* (for example $0 = 1$).

Now, by a *constraint satisfaction problem*, CSP in short, we mean a finite sequence of variables $X := x_1, \ldots, x_n$ with respective domains $\mathcal{D} := D_1, \ldots, D_n$, together with a finite set $\mathcal{C}$ of constraints, each on a subsequence of X. We write such a CSP as $\langle \mathcal{C} \ ; \ \mathcal{DE} \rangle$, where $\mathcal{DE} := x_1 \in D_1, \ldots, x_n \in D_n$ and call each construct of the form $x \in D$ a *domain expression*. To simplify the notation from now on we omit the "{ }" brackets when presenting specific sets of constraints $\mathcal{C}$.

Consider a CSP $\langle \mathcal{C} \ ; \ \mathcal{DE} \rangle$ with $\mathcal{DE} := x_1 \in D_1, \ldots, x_n \in D_n$. We say that an n-tuple $(d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n$ *is a solution to* $\langle \mathcal{C} \ ; \ \mathcal{DE} \rangle$ if for every constraint $C \in \mathcal{C}$ on the variables $x_{i_1}, \ldots, x_{i_m}$ we have

$$(d_{i_1}, \ldots, d_{i_m}) \in C.$$

We call a CSP *solved* if it is of the form $\langle \emptyset \ ; \ \mathcal{DE} \rangle$ where no domain in $\mathcal{DE}$ is empty, and *failed* if some of its domains is empty. Two CSP's with the same sequence of variables are called *equivalent* if they have the same set of solutions.

Finally, given a constraint c on the variables $x_1, \ldots, x_n$ with respective domains $D_1, \ldots, D_n$, and a sequence of domains $D'_1, \ldots, D'_n$ such that for $i \in [1..n]$ we have $D'_i \subseteq D_i$, we say that c' is the *result of restricting c to the domains* $D'_1, \ldots, D'_n$ if $c' = c \cap (D'_1 \times \ldots \times D'_n)$.

2.2 Boolean Constraints

In this paper we focus on Boolean constraint satisfaction problems. They deal with Boolean variables and constraints on them defined by means of Boolean connectives and equality. Let us recall the definitions.

By a *Boolean variable* we mean a variable which ranges over the domain which consists of two values: 0 denoting **false** and 1 denoting **true**. By a *Boolean domain expression* we mean an expression of the form $x \in D$ where $D \subseteq \{0, 1\}$. In what follows we write the Boolean domain expression $x \in \{1\}$ as $x = 1$ and $x \in \{0\}$ as $x = 0$. By a *Boolean expression* we mean an expression built out of Boolean variables using three connectives: $\neg$ (*negation*), $\wedge$ (*conjunction*) and $\vee$ (*disjunction*).

Next, by a *Boolean constraint* we mean a formula of the form $s = t$, where s, t are Boolean expressions. In presence of Boolean domain expressions each Boolean constraint uniquely determines a constraint on the sequence of its variables, so from now on we identify each Boolean constraint with the constraint determined by it.

Finally, by a *Boolean constraint satisfaction problem*, in short *Boolean CSP*, we mean a CSP with Boolean domain expressions and all constraints of which are Boolean constraints.

Note that in our framework the Boolean constants, **true** and **false**, are absent. They can be easily modeled by using two predefined variables, say x_T and x_F, with the Boolean domain expressions $x_T = 1$ and $x_F = 0$.

In the sequel x, y, z denote different Boolean variables. We call a Boolean constraint *simple* if it is in one of the following form:

- $x = y$; we call it the *equality* constraint,
- $\neg x = y$; we call it the *NOT* constraint,
- $x \wedge y = z$; we call it the *AND* constraint,
- $x \vee y = z$; we call it the *OR* constraint.

By introducing auxiliary variables it is straightforward to transform each Boolean CSP into an equivalent one all constraints of which are simple. (More precisely, to prove such an equivalence result, the notion of equivalence between CSP's has to be extended to deal with CSP's with different sequences of variables. This was done in Apt (1998).) So we assume in the sequel that all Boolean constraints are simple.

2.3 A Proof Theoretic Framework

Next, we recall briefly the proof theoretic framework introduced in Apt (1998). In this paper two types of proof rules for CSP's were introduced: *deterministic* and *splitting*. Here we focus on the deterministic ones; the splitting rules will be introduced in Section 6.

The deterministic rules are of the form

$$\frac{\phi}{\psi}$$

where ϕ and ψ are CSP's. We assume here that ϕ is not failed and its set of constraints is non-empty. Depending on the form of the conclusion ψ we distinguish two cases.

- *Domain reduction rules*, or in short *reduction rules*. They are of the form

  $$\frac{\langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n \rangle}{\langle \mathcal{C}' \ ; \ x_1 \in D'_1, \ldots, x_n \in D'_n \rangle}$$

  where for $i \in [1..n]$ we have $D'_i \subseteq D_i$ and where $\mathcal{C}'$ is the result of restricting each constraint in $\mathcal{C}$ to the corresponding subsequence of the domains $D'_1, \ldots, D'_n$.

  When all constraints in $\mathcal{C}'$ are solved, we call such a rule a *solving rule*.
- *Transformation rules*. These rules are not domain reduction rules. In this paper they are of the following form:

  $$\frac{\langle \mathcal{C} \ ; \ \mathcal{DE} \rangle}{\langle \mathcal{C}' \ ; \ \mathcal{DE} \rangle}$$

Now that we have defined the proof rules, we define the result of applying a proof rule to a CSP. Intuitively, we just replace in a given CSP the part that coincides with the premise by the conclusion and restrict the "old" constraints to the new domains.

Because of variable clashes we need to be more precise. So assume a CSP of the form $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$ and consider a rule of the form

$$\frac{\langle \mathcal{C}_1 \ ; \ \mathcal{DE}_1 \rangle}{\langle \mathcal{C}_2 \ ; \ \mathcal{DE}_2 \rangle} \tag{1}$$

Call a variable that appears in the conclusion but not in the premise an *introduced* variable of the rule. By appropriate renaming we can assume that no introduced variable of this rule appears in $\langle \mathcal{C} \ ; \ \mathcal{DE} \rangle$.

Let now $\mathcal{C}'$ be the result of restricting each constraint in $\mathcal{C}$ to the domains in $\mathcal{DE} \cup \mathcal{DE}_2$. We say that rule (1) *can be applied* to $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$ and call

$$\langle \mathcal{C}' \cup \mathcal{C}_2 \ ; \ \mathcal{DE} \cup \mathcal{DE}_2 \rangle$$

the *result of applying rule* (1) *to* $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$.

To discuss the effect of an application of a proof rule to a CSP we introduce the following notions.

Definition 2.1 Consider two CSP's ϕ and ψ and a deterministic rule R.

- We call ϕ a *reformulation* of ψ if the removal of solved constraints from ϕ and ψ yields the same CSP.
- Suppose that ψ is the result of applying the rule R to the CSP ϕ. If ψ is not a reformulation of ϕ, then we call this a *relevant application* of R to ϕ.
- Suppose that the rule R cannot be applied to ϕ or no application of it to ϕ is relevant. Then we say that ϕ is *closed under the applications of R*. $\Box$

For example, the Boolean CSP $\phi := \langle x \wedge y = z \ ; \ x = 1, y = 0, z = 0 \rangle$ is closed under the applications of the transformation rule

$$\frac{\langle x \wedge y = z \ ; \ x = 1, y \in D_y, z \in D_z \rangle}{\langle y = z \ ; \ x = 1, y \in D_y, z \in D_z \rangle}$$

Indeed, this rule can be applied to ϕ; the outcome is $\psi := \langle z = y \ ; \ x = 1, y = 0, z = 0 \rangle$. After the removal of solved constraints from ϕ and ψ we get in both cases the solved CSP $\langle \emptyset \ ; \ x = 1, y = 0, z = 0 \rangle$.

In contrast, the Boolean CSP $\phi := \langle x \wedge y = z \ ; \ x = 1, y \in \{0, 1\}, z \in \{0, 1\} \rangle$ is not closed under the applications of the above rule because $\langle z = y \ ; \ x = 1, y \in \{0, 1\}, z \in \{0, 1\} \rangle$ is not a reformulation of ϕ.

To conclude this brief review we need two more definitions. Given two CSP's ϕ and ψ with the same variables, we say that ϕ *is smaller than* ψ if it is obtained from ψ by an application of a reduction rule.

Finally, a proof rule

$$\frac{\phi}{\psi}$$

is called *equivalence preserving* if ϕ and ψ are equivalent.

All the rules discussed in this paper are easily seen to be equivalence preserving. The following lemma explains why the equivalence preserving rules are important.

Lemma 2.2 (Equivalence) *Suppose that the CSP ψ is the result of applying an equivalence preserving rule to the CSP ϕ. Then ϕ and ψ are equivalent.* $\Box$

3 A Characterization of Arc Consistency

The following notion was introduced in Mohr & Masini (1988). The original definition for binary constraints is due to Mackworth (1977).

Definition 3.1

- A constraint C is called *arc consistent* if for every variable of it each value in its domain participates in a solution to C.
- A CSP is called *arc consistent* if every constraint of it is. $\Box$

We now characterize arc consistency for Boolean CSP's by means of proof rules. In what follows we write these rules in a simplified form. We illustrate it by means of three representative examples.

We write the solving rule

$$\frac{\langle \neg x = y \ ; \ x \in D_x, y = 0 \rangle}{\langle \ ; \ x \in D_x \cap \{1\}, y = 0 \rangle}$$

as

$$\neg x = y, y = 0 \rightarrow x = 1,$$

the already mentioned transformation rule

$$\frac{\langle x \wedge y = z \ ; \ x = 1, y \in D_y, z \in D_z \rangle}{\langle y = z \ ; \ x = 1, y \in D_y, z \in D_z \rangle}$$

as

$$x \wedge y = z, x = 1 \rightarrow z = y,$$

and the solving rule

$$\frac{\langle x \vee y = z \ ; \ x = 0, y \in D_y, z = 1 \rangle}{\langle \ ; \ x = 0, y \in D_y \cap \{1\}, z = 1 \rangle}$$

as

$$x \vee y = z, x = 0, z = 1 \rightarrow y = 1.$$

Using this convention we now introduce 20 solving rules presented in Table 1 and call the resulting proof system *BOOL*.

To read properly such formulas it helps to remember that 0 and 1 are domain elements, so atomic formulas of the form $x = 0$ and $x = 1$ are domain expressions while all other atomic formulas are constraints.

Observe, however, that our interpretation of these formulas as shorthands for the corresponding rules amounts to a different, *dynamic* interpretation of them. For example *AND 1* should be interpreted as: $x \wedge y = z, x = 1$ and $y = 1$ imply that z *becomes* 1. Additionally, as a side effect of applying such a rule, the constraint — here $x \wedge y = z$ — is deleted.

Note also that each of these rules can yield a failed CSP — take for instance the *NOT 2* rule applied to $\langle \neg x = y \ ; \ x = 0, y = 0 \rangle$.

We now establish the following result.

EQU 1	$x = y, x = 1 \to y = 1$
EQU 2	$x = y, y = 1 \to x = 1$
EQU 3	$x = y, x = 0 \to y = 0$
EQU 4	$x = y, y = 0 \to x = 0$
NOT 1	$\neg x = y, x = 1 \to y = 0$
NOT 2	$\neg x = y, x = 0 \to y = 1$
NOT 3	$\neg x = y, y = 1 \to x = 0$
NOT 4	$\neg x = y, y = 0 \to x = 1$
AND 1	$x \wedge y = z, x = 1, y = 1 \to z = 1$
AND 2	$x \wedge y = z, x = 1, z = 0 \to y = 0$
AND 3	$x \wedge y = z, y = 1, z = 0 \to x = 0$
AND 4	$x \wedge y = z, x = 0 \to z = 0$
AND 5	$x \wedge y = z, y = 0 \to z = 0$
AND 6	$x \wedge y = z, z = 1 \to x = 1, y = 1$
OR 1	$x \vee y = z, x = 1 \to z = 1$
OR 2	$x \vee y = z, x = 0, y = 0 \to z = 0$
OR 3	$x \vee y = z, x = 0, z = 1 \to y = 1$
OR 4	$x \vee y = z, y = 0, z = 1 \to x = 1$
OR 5	$x \vee y = z, y = 1 \to z = 1$
OR 6	$x \vee y = z, z = 0 \to x = 0, y = 0$

Table 1: Proof system *BOOL*

Theorem 3.2 (Arc Consistency) *A non-failed Boolean CSP is arc consistent iff it is closed under the applications of the rules of the proof system BOOL.*

Proof. Let ϕ be the CSP under consideration. Below $C := x \wedge y = z$ is some *AND* constraint belonging to ϕ. We view it as a constraint on the variables x, y, z in this order. Let D_x, D_y and D_z be respectively the domains of x, y and z.
$(\Rightarrow)$ We need to consider each rule in turn. We analyse here only the *AND* rules. For other rules the reasoning is similar.

AND 1 rule.

Suppose that $D_x = \{1\}$ and $D_y = \{1\}$. If $0 \in D_z$, then by the arc consistency for some $d_1 \in D_x$ and $d_2 \in D_y$ we have $(d_1, d_2, 0) \in C$, so $(1, 1, 0) \in C$ which is a contradiction.

This shows that $D_z = \{1\}$ which means that ϕ is closed under the applications of this rule.

AND 2 rule.

Suppose that $D_x = \{1\}$ and $D_z = \{0\}$. If $1 \in D_y$, then by the arc consistency for some $d_1 \in D_x$ and $d_2 \in D_z$ we have $(d_1, 1, d_2) \in C$, so $(1, 1, 0) \in C$ which is a contradiction.

This shows that $D_y = \{0\}$ which means that ϕ is closed under the applications of this rule.

AND 3 rule.

This case is symmetric to that of the *AND 2* rule.

AND 4 rule.

Suppose that $D_x = \{0\}$. If $1 \in D_z$, then by the arc consistency for some $d_1 \in D_x$ and $d_2 \in D_y$ we have $(d_1, d_2, 1) \in C$, so $(1, 1, 1) \in C$ which is a contradiction.

This shows that $D_z = \{0\}$ which means that ϕ is closed under the applications of this rule.

AND 5 rule.

This case is symmetric to that of the *AND 4* rule.

AND 6 rule.

Suppose that $D_z = \{1\}$. If $0 \in D_x$, then by the arc consistency for some $d_1 \in D_y$ and $d_2 \in D_z$ we have $(0, d_1, d_2) \in C$, so $0 \in D_z$ which is a contradiction.

This shows that $D_x = \{1\}$. By a symmetric argument also $D_y = \{1\}$ holds. This means that ϕ is closed under the applications of this rule.

$(\Leftarrow)$ Consider the *AND* constraint C. We have to analyze six cases.

Case 1. Suppose $1 \in D_x$.

Assume that neither $(1, 1) \in D_y \times D_z$ nor $(0, 0) \in D_y \times D_z$. Then either $D_y = \{1\}$ and $D_z = \{0\}$ or $D_y = \{0\}$ and $D_z = \{1\}$.

If the former holds, then by the *AND 3* rule we get $D_x = \{0\}$ which is a contradiction. If the latter holds, then by the *AND 5* rule we get $D_z = \{0\}$ which is a contradiction.

We conclude that for some d we have $(1, d, d) \in C$.

Case 2. Suppose $0 \in D_x$.

Assume that $0 \notin D_z$. Then $D_z = \{1\}$, so by the *AND 6* rule we get $D_x = \{1\}$ which is a contradiction. Hence $0 \in D_z$. Let now d be some element of D_y. We then have $(0, d, 0) \in C$.

Case 3. Suppose $1 \in D_y$.

This case is symmetric to Case 1.

Case 4. Suppose $0 \in D_y$.

This case is symmetric to Case 2.

Case 5. Suppose $1 \in D_z$.

Assume that $(1, 1) \notin D_x \times D_y$. Then either $D_x = \{0\}$ or $D_y = \{0\}$. If the former holds, then by the *AND 4* rule we conclude that $D_z = \{0\}$. If the latter holds, then by the *AND 5* rule we conclude that $D_z = \{0\}$. For both possibilities we reached a contradiction. So both $1 \in D_x$ and $1 \in D_y$ and consequently $(1, 1, 1) \in C$.

Case 6. Suppose $0 \in D_z$.

Assume that both $D_x = \{1\}$ and $D_y = \{1\}$. By the rule *AND 1* rule we conclude that $D_z = \{1\}$ which is a contradiction. So either $0 \in D_x$ or $0 \in D_y$ and consequently for some d either $(0, d, 0) \in C$ or $(d, 0, 0) \in C$.

An analogous reasoning can be spelled out for the equality, *OR* and *NOT* constraints and is omitted. $\Box$

Note that the restriction to non-failed CSP's is necessary: the failed CSP $\langle x \wedge y = z \ ; \ x \in \emptyset, y \in \{0,1\}, z \in \{0,1\}\rangle$ is not arc consistent but it is closed under the applications of the rules of *BOOL*.

It is also easy to check that all the rules of the *BOOL* system are needed, that is, this result does not hold when any of these 20 rules is omitted. For example, if the rule *AND 4* is left out, then the CSP $\langle x \wedge y = z \ ; \ x = 0, y \in \{0,1\}, z \in \{0,1\}\rangle$ is closed under the applications of all remaining rules but is not arc consistent.

The above theorem shows that in order to reduce a Boolean CSP to an equivalent one that is either failed or arc consistent it suffices to close it under the applications of the rules of the *BOOL* system. We shall return to this matter after having discussed a related proof system.

4 The Proof System of Codognet and Diaz

In Codognet & Diaz (1996) a slightly different proof system was introduced to deal with Boolean constraints. It consists of the proof rules given in Table 2. We call the resulting proof system *BOOL'*.

EQU 1 − 4	as in the system *BOOL*
NOT 1 − 4	as in the system *BOOL*
AND 1′	$x \wedge y = z, x = 1 \rightarrow z = y$
AND 2′	$x \wedge y = z, y = 1 \rightarrow z = x$
AND 3′	$x \wedge y = z, z = 1 \rightarrow x = 1$
AND 4	as in the system *BOOL*
AND 5	as in the system *BOOL*
AND 6′	$x \wedge y = z, z = 1 \rightarrow y = 1$
OR 1	as in the system *BOOL*
OR 2′	$x \vee y = z, x = 0 \rightarrow z = y$
OR 3′	$x \vee y = z, y = 0 \rightarrow z = x$
OR 4′	$x \vee y = z, z = 0 \rightarrow x = 0$
OR 5	as in the system *BOOL*
OR 6′	$x \vee y = z, z = 0 \rightarrow y = 0$

Table 2: Proof system *BOOL'*

To be precise, the rules *EQU 1–4* are not present in Codognet & Diaz (1996). Instead, the constraints $0 = 0$ and $1 = 1$ are adopted as axioms. Note that the rules *AND 1', AND 2', OR 2'* and *OR 3'* are transformation rules.

The main difference between *BOOL* and *BOOL'* lies in the fact that the rules *AND 1–3* of *BOOL* are replaced by the rules *AND 1'* and *AND 2'* of *BOOL'* and the rules *OR 2–4* of *BOOL* are replaced by the rules *OR 2'* and

OR 3' of *BOOL'*. (The fact that the rule *AND 6* of *BOOL* is split in *BOOL'* into two rules, *AND 3'* and *AND 6'* and analogously for the rules *OR 6* of *BOOL* and *OR 3'* and *OR 6'* of *BOOL'* is of no importance.)

A natural question arises whether the proof systems *BOOL* and *BOOL'* are equivalent. The precise answer is "sometimes". First, observe that the following result holds.

Theorem 4.1 *If a non-failed Boolean CSP is closed under the applications of the rules of the proof system BOOL', then it is arc consistent.*

Proof. The proof relies on the following immediate observation.

Claim 1 *Consider a Boolean CSP ϕ containing the AND constraint $x \wedge y = z$ on the variables x, y, z with respective domains D_x, D_y and D_z. If ϕ is closed under the applications of the AND 1' rule, then $D_x = \{1\}$ implies $D_y = D_z$. If ϕ is closed under the applications of the AND 2' rule, then $D_y = \{1\}$ implies $D_x = D_z$.* □

Suppose now that the CSP in question contains the *AND* constraint $x \wedge y = z$ on the variables x, y, z with respective domains D_x, D_y and D_z. We present the proof only for the cases where the argument differs from the one given in the proof of the Arc Consistency Theorem 3.2.

Case 1. Suppose $1 \in D_x$.

Assume that neither $(1,1) \in D_y \times D_z$ nor $(0,0) \in D_y \times D_z$. Then either $D_y = \{1\}$ and $D_z = \{0\}$ or $D_y = \{0\}$ and $D_z = \{1\}$.

If the former holds, then by Claim 1 $D_y = D_z$, which is a contradiction. If the latter holds, then by the *AND 5* rule $D_z = \{0\}$ which is also a contradiction. We conclude that for some d we have $(1, d, d) \in C$.

Case 6. Suppose $0 \in D_z$.

Assume that both $D_x = \{1\}$ and $D_y = \{1\}$. By Claim 1 $D_y = D_z$, which is a contradiction. So either $0 \in D_x$ or $0 \in D_y$ and consequently for some d either $(0, d, 0) \in C$ or $(d, 0, 0) \in C$.

The reasoning for the *OR* and *NOT* constraints is analogous and omitted. □

In constrast to the case of the *BOOL* system the converse result does not hold. Indeed, just take the CSP $\phi := \langle x \wedge y = z \ ; \ x = 1, y \in \{0,1\}, z \in \{0,1\}\rangle$. Note that ϕ is arc consistent but it is not closed under the applications of the *AND 1'* rule.

This brings us to the following definition.

Definition 4.2 We call a Boolean CSP *limited* if none of the following four CSP's forms a subpart of it:

- $\langle x \wedge y = z \ ; \ x = 1, y \in \{0,1\}, z \in \{0,1\}\rangle$,
- $\langle x \wedge y = z \ ; \ x \in \{0,1\}, y = 1, z \in \{0,1\}\rangle$,
- $\langle x \vee y = z \ ; \ x = 0, y \in \{0,1\}, z \in \{0,1\}\rangle$,

- $\langle x \vee y = z \; ; \; x \in \{0,1\}, y = 0, z \in \{0,1\}\rangle$. $\Box$

We can now prove the following result.

Theorem 4.3 *If a non-failed Boolean CSP is limited and arc consistent, then it is closed under the applications of the rules of the proof system BOOL'.*

Proof. In view of the Arc Consistency Theorem 3.2 we only have to consider the rules of *BOOL'* that are absent in *BOOL*. We present here an argument for one representative rule.

AND 1' rule.

Suppose that $D_x = \{1\}$. If $0 \in D_y$, then by the arc consistency for some $d \in D_z$ we have $(1, 0, d) \in C$, which means that $0 \in D_z$. Conversely, if $0 \in D_z$, then by the arc consistency for some $d \in D_y$ we have $(1, d, 0) \in C$, so $0 \in D_y$. By a similar argument we get that $1 \in D_y$ iff $1 \in D_z$. This shows that $D_y = D_z$.

By assumption ϕ is limited, so either $D_y \neq \{0,1\}$ or $D_z \neq \{0,1\}$. Hence either $D_y = D_z = \{1\}$ or $D_y = D_z = \{0\}$. In both cases The CSP under consideration is closed under the applications of the *AND 1'* rule. $\Box$

5 Enforcing Arc Consistency

The Arc Consistency Theorem 3.2 suggests an obvious way of transforming a Boolean CSP into one that is failed or arc consistent: it just suffices to compute its closure under the rules of the *BOOL* system.

This brings us to the question whether such a closure is uniquely defined. Before we proceed, it is useful to point out that a naive argument based on commutativity of the considered rules does not hold here.

Indeed, consider the CSP

$$\phi := \langle x \wedge y = z, \neg x = u \; ; \; u = 1, x \in \{0,1\}, y \in \{0,1\}, z \in \{0,1\}\rangle.$$

By applying to it the *NOT 3* rule we get

$$\psi := \langle x \wedge y = z \; ; \; u = 1, x = 0, y \in \{0,1\}, z \in \{0,1\}\rangle.$$

An application of the *AND 4* now yields the solved CSP

$$\langle \emptyset \; ; \; u = 1, x = 0, y \in \{0,1\}, z = 0\rangle.$$

However, when we apply these two rules in the reversed order, we "only" get ψ. The reason is that the *AND 4* rule cannot be applied to ϕ, or to put it in other words, ϕ is closed under the applications of the *AND 4* rule.

Let us define now the notion of a closure formally.

Definition 5.1 Consider two CSP's ϕ and ψ and a finite set of proof rules $\mathcal{R}$. We say that ψ is a *closure of ϕ under the rules from $\mathcal{R}$* if

- ψ is obtained from ϕ by a finite number of consecutive applications of the rules from $\mathcal{R}$,

- ψ is closed under the applications of the rules from $\mathcal{R}$. □

Each proof rule can be viewed as a function on CSP's by assuming that it is the identity function on CSP's to which it cannot be applied. Therefore the following definition is meaningful.

Definition 5.2 We say that a proof rule R is *monotonic* if, when viewed as a function, it is monotonic, w.r.t. the "smaller than" relation. □

Now, the general result proved in Apt (1997) (more precisely, the Domain Reduction Theorem on page 47) implies the following.

Theorem 5.3 (Closure) *Consider a CSP ϕ with finite domains and a finite set of reduction rules $\mathcal{R}$. Suppose that all rules in $\mathcal{R}$ are equivalence preserving and monotonic. Then*

- *a closure of ϕ under the rules from $\mathcal{R}$ exists and is equivalent to ϕ,*
- *if a closure of ϕ under the rules from $\mathcal{R}$ is non-failed, then it is uniquely defined and it is the largest CSP closed under the rules from $\mathcal{R}$ that is smaller than ϕ.* □

The above theorem applies to our situation because all proof rules of the system *BOOL* are easily seen to satisfy the conditions of this theorem. This shows that a closure of a Boolean CSP ϕ under the rules of *BOOL* exists and is equivalent to ϕ. Moreover, if it is non-failed, then it is uniquely defined and is the largest arc consistent Boolean CSP smaller than ϕ.

It is useful to see that the closure under the rules of *BOOL* does not have to be unique. Indeed, take the CSP

$$\phi := \langle x \wedge y = z, x \wedge y = u \ ; \ u = 0, x = 1, y = 1, z = 0\rangle.$$

Then the *AND 1* rule can be applied in two ways, so both

$$\langle x \wedge y = u \ ; \ u = 0, x = 1, y = 1, z \in \emptyset\rangle$$

and

$$\langle x \wedge y = z \ ; \ u \in \emptyset, x = 1, y = 1, z = 0\rangle$$

are closures of ϕ, since by definition no rule can be applied to a failed CSP. On the other hand, all closures under the rules of *BOOL* are obviously equivalent.

The Closure Theorem 5.3 also allows us to study confluence of the rules of the *BOOL* system.

Definition 5.4 Let $\rightarrow$ be a binary relation on a set A and $\rightarrow^*$ its transitive reflexive closure. An element a of A is called *confluent w.r.t.* $\rightarrow$ if for all $b, c \in A$ the fact that $a \rightarrow^* b$ and $a \rightarrow^* c$ implies that for some $d \in A$ we have $b \rightarrow^* d$ and $c \rightarrow^* d$. □

We now have the following immediate consequence of the Closure Theorem 5.3 that can be applied to the *BOOL* system.

Theorem 5.5 *Consider ϕ and $\mathcal{R}$ as in the Closure Theorem 5.3. For two CSP's ψ_1 and ψ_2 let $\psi_1 \rightarrow_{\mathcal{R}} \psi_2$ denote the fact that ψ_2 is obtained from ψ_1 by a single application of a rule from $\mathcal{R}$.*

If a closure of ϕ under the rules from $\mathcal{R}$ is non-failed, then ϕ is confluent w.r.t. $\rightarrow_{\mathcal{R}}$. □

The above Boolean CSP $\langle x \wedge y = z, x \wedge y = u \ ; \ u = 0, x = 1, y = 1, z = 0\rangle$ shows that the restriction to CSP's with non-failed closure is necessary here.

Without going into details let us mention that analogous results also hold for the system *BOOL'* though the argument is more involved as *AND 1', AND 2', OR 2'* and *OR 3'* are not reduction rules. To deal with this complication an appropriate modification of the Constraint Reduction Theorem of Apt (1997, page 49) can be used.

The obvious question that comes now to one's mind is how to schedule the proof rules of the *BOOL* system in a meaningful way when computing a closure.

One possibility is to apply one of the generic chaotic iteration algorithms of Apt (1997). Such algorithms schedule the rules in a way analogous to the one used in the AC-3 and PC-2 algorithms of Mackworth (1977). As pointed out in Apt (1997), this scheduling strategy is employed in several constraint propagation algorithms proposed in the literature.

Here, however, because all rules are solving, a more natural strategy is to put all constraints in a ring and repeatedly advance around it, each time trying to apply some rule. In this approach the constraints and not the rules are scheduled.

6 From Arc Consistency to a Boolean Constraint Solver

Of course, reducing a Boolean CSP to an equivalent one that is failed or arc consistent is not sufficient to find all solutions to it, let alone to solve it. Take for example the simple constraints $x \wedge y = z, x \wedge v = z, \neg y = v$ with all variable domains equal $\{0, 1\}$. This CSP is arc consistent but to find its two solutions, $x = 0, y = 1, z = 0, v = 0$ and $x = 0, y = 0, z = 0, v = 1$, some additional techniques are needed.

One, widely used, method is the *look-ahead* search strategy (see Tsang (1993)). Look-ahead is usually defined for binary constraints only. Here we use a natural generalization of it to arbitrary constraints. Look-ahead for arbitrary CSP's combines backtracking, triggered by instantiation of some variable with non-singleton domain, with enforcement of the arc consistency on the resulting CSP. If during this enforcement process the domain of a variable becomes empty, a failure arises.

To explain this technique for Boolean CSP's in logical terms we return to Apt (1998) and extend our proof theoretic framework by adding to it splitting rules.

The splitting rules are of the form

$$\frac{\phi}{\psi_1 \mid \psi_2}$$

where ϕ, ψ_1 and ψ_2 are CSP's with the same sequence of variables. As for deterministic rules we assume here that ϕ is not failed and its set of constraints is non-empty.

These rules allow us to replace one CSP by two CSP's. They are counterparts of the deterministic rules, so we distinguish two cases.

- *Reduction splitting rules.* These are rules such that both $\frac{\psi}{\phi_1}$ and $\frac{\psi}{\phi_2}$ are reduction rules.
- *Transformation splitting rules.* These are rules such that both $\frac{\psi}{\phi_1}$ and $\frac{\psi}{\phi_2}$ are transformation rules.

Consider now a CSP of the form $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$ and a splitting rule of the form

$$\frac{\langle \mathcal{C}_1 \ ; \ \mathcal{DE}_1 \rangle}{\langle \mathcal{C}_2 \ ; \ \mathcal{DE}_2 \rangle \mid \langle \mathcal{C}_3 \ ; \ \mathcal{DE}_3 \rangle} \tag{2}$$

We then say that rule (2) *can be applied* to $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$ and call

$$\langle \mathcal{C} \cup \mathcal{C}_2 \ ; \ \mathcal{DE} \cup \mathcal{DE}_2 \rangle \mid \langle \mathcal{C} \cup \mathcal{C}_3 \ ; \ \mathcal{DE} \cup \mathcal{DE}_3 \rangle \tag{3}$$

the *result of applying it to* $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$. If neither $\langle \mathcal{C} \cup \mathcal{C}_2 \ ; \ \mathcal{DE} \cup \mathcal{DE}_2 \rangle$ nor $\langle \mathcal{C} \cup \mathcal{C}_3 \ ; \ \mathcal{DE} \cup \mathcal{DE}_3 \rangle$ is a reformulation of $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$, then we say that (3) is the result of a *relevant application of rule* (2) *to* $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{DE} \cup \mathcal{DE}_1 \rangle$.

Finally, we introduce the notion of a proof tree.

Definition 6.1 Assume a set of proof rules. A *proof tree* is a tree the nodes of which are CSP's. Further, each node has at most two direct descendants and for each node ϕ the following holds:

- If ϕ is a leaf, then no application of a rule to ψ is relevant;
- If ϕ has precisely one direct descendant, say ψ, then ψ is the result of a relevant application of a proof rule to ϕ;
- If ϕ has precisely two direct descendants, say ψ_1 and ψ_2, then $\psi_1 \mid \psi_2$ is the result of a relevant application of a proof rule to ϕ. □

The idea behind the above definition is that we consider in the proof trees only those applications of the proof rules that cause some change. Note also that more proof rules can be applicable to a given CSP, so a specific CSP can be a root of several proof trees.

In what follows we consider only one splitting rule, namely

INSTANTIATION

$$\frac{\langle \mathcal{C} \ ; \ \mathcal{DE}, x \in \{0,1\} \rangle}{\langle \mathcal{C}' \ ; \ x = 1 \rangle \mid \langle \mathcal{C}'' \ ; \ x = 0 \rangle}$$

where $\mathcal{C}'$ is the result of restricting each constraint in $\mathcal{C}$ to the corresponding subsequence of the domains $D'_1, \ldots, D'_n$ domains in $\mathcal{DE} \cup \{x = 1\}$ and analogously with $\mathcal{C}''$.

We add it to *BOOL* and call the resulting system *BOOL+*. We then have the following result that characterizes the set of all solutions to a Boolean CSP.

Theorem 6.2 *Consider a Boolean CSP* $\phi := \langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n \rangle$ *and a proof tree* $\mathcal{T}$ *for the proof rules of BOOL+ with the root* ϕ*. Then an* n*-tuple* $(d_1, \ldots, d_n)$ *is a solution to* ϕ *iff the solved CSP* $\langle \emptyset \ ; \ x_1 = d_1, \ldots, x_n = d_n \rangle$ *is a leaf in* $\mathcal{T}$*.*

Proof. It is an immediate consequence of the fact that all rules of *BOOL* are equivalence preserving and that given the *INSTANTIATION* rule $\frac{\phi}{\psi_1 | \psi_2}$, every solution to ϕ is a solution to ψ_1 or to ψ_2 and every solution to ψ_i $(i \in [1, 2])$ is a solution to ϕ. □

The above result provides us with a way of computing all solutions to a Boolean CSP but leaves us with a considerable degree of freedom concerning the choice of the proof tree. A natural, widely used, heuristic consists of delaying the applications of the splitting rule *INSTANTIATION* as much as possible. This leads to the proof trees that embody the look-ahead search strategy. Indeed, the proof trees are defined in such a way that the internal nodes are non-failed CSP's. Hence by the Arc Consistency Theorem 3.2 the *INSTANTIATION* rule can be applied to a CSP only if it is non-failed and arc consistent.

Let us mention also that the other well-known search strategy, forward checking, can be described in a similar way. Recall that forward checking differs from look-ahead in that after the instantiation of some variable the arc consistency is enforced only on the set of constraints that contain this variable.

In our approach forward checking for Boolean constraints translates into a strategy according to which after each application of the *INSTANTIATION* rule to some variable, say x, a closure under the rules of *BOOL* is computed only for the set of constraints that contain the variable x.

In practise it seems that for Boolean CSP's forward checking is inferior to look-ahead.

The above considerations lead to a natural implementation of a Boolean constraint solver. In fact, a student of ours, Antal Godkewitsch, implemented such a Boolean constraint solver in C, both based on the rules of the *BOOL+* system and on a similar system that deals with simple Boolean constraints built out of literals instead of variables only. This, when combined with the well-known heuristic of instantiating the most constrained variable, led to a constraint solver that performs on various benchmarks 1.3 to 7 times slower than the apparently fastest known Boolean constraint solver, `clp(B)` of Codognet & Diaz (1996). This is not too bad, having in mind that the latter solver was implemented by means of a highly optimized and dedicated extension of WAM.

A truly straightforward way of implementing the above approach consists of using the constraint handling rules (CHRs) of Frühwirth (1995) that are part of the ECL^{i}PS^e system (see A. Aggoun et al. (1995)). In fact, Boolean constraints form a prime example for an effective use of CHRs.

The CHRs allow us to add various deterministic (in our terminology) rules to a constraint logic program. The application of these rules has a priority over the logic programming resolution step. Rewriting the rules of the *BOOL* system as so-called simplification rules of CHR and adding the labeling to simulate the *INSTANTIATION* rule effectively results in a Boolean constraint solver that realizes the look-ahead search strategy discussed here.

In fact, in Frühwirth (1998, page 113) a Boolean constraint solver can be found that consists of CHRs that amount to the rules of the *BOOL'* system.

It should be remarked though that this approach comes with a fixed strategy of selecting the constraints and rules because the scheduler of CHRs is hardwired within the ECL^{i}PS^e system.

Acknowledgements

We would like to thank Antal Godkewitsch for useful discussions on the implementation of a Boolean constraint solver based on the approach here discussed.

References

A. Aggoun et al. (1995), *ECL^{i}PS^e 3.5 User Manual*, Munich, Germany.

Apt, K. R. (1997), From chaotic iteration to constraint propagation, *in* P. Degano, R. Gorrieri & A. Marchetti-Spaccamela, eds, 'Proceedings of the 24th International Colloquium, ICALP '97', Vol. 1256 of *Lecture Notes in Computer Science*, Springer-Verlag, New York, pp. 36–55. Invited Lecture.

Apt, K. R. (1998), 'A proof theoretic view of constraint programming', *Fundamenta Informaticae* **33**(3), 263–293. Available via `http://www.cwi.nl/~apt`.

Benhamou, F. & Colmerauer, A., eds (1993), *Constraint Logic Programming: Selected Research*, MIT Press, Cambridge, MA.

Codognet, P. & Diaz, D. (1996), 'A simple and efficient Boolean constraint solver for constraint logic programming', *Journal of Automated Reasoning* **17**(1), 97–128.

Dalal, M. (1992), Efficient Propositional Constraint Propagation, *in* 'Proceedings of the 10^{th} National Conference on Artificial Intelligence, AAAI'92', pp. 409–414. San Jose, California.

Frühwirth, T. (1995), Constraint Handling Rules, *in* A. Podelski, ed., 'Constraint Programming: Basics and Trends', LNCS 910, Springer-Verlag, pp. 90–107. (Châtillon-sur-Seine Spring School, France, May 1994).

Frühwirth, T. (1998), 'Theory and practice of constraint handling rules', *Journal of Logic Programming* **37**(1–3), 95–138. Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriot, Eds.).

Mackworth, A. (1977), 'Consistency in networks of relations', *Artificial Intelligence* **8**(1), 99–118.

Mohr, R. & Masini, G. (1988), Good old discrete relaxation, *in* Y. Kodratoff, ed., 'Proceedings of the 8th European Conference on Artificial Intelligence (ECAI)', Pitman Publishers, pp. 651–656.

Stallman, R. M. & Sussman, G. J. (1977), 'Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis', *Artificial Intelligence* **9**, 135–196.

Tsang, E. (1993), *Foundations of Constraint Satisfaction*, Academic Press.