Some Remarks on Boolean Constraint Propagation

Krzysztof R. Apt
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
University of Amsterdam, The Netherlands

Abstract

We study here the well-known propagation rules for Boolean constraints. First we propose a simple notion of completeness for sets of such rules and establish a completeness result. Then we show an equivalence in an appropriate sense between Boolean constraint propagation and unit propagation, a form of resolution for propositional logic.

Subsequently we characterize one set of such rules by means of the notion of hyper-arc consistency introduced in Mohr & Masini (1988). Also, we clarify the status of a similar, though different, set of rules introduced in Simonis (1989) and more fully in Codognet & Diaz (1996).

1 Introduction

1.1 Motivation

Boolean constraints form a special case of constraint satisfaction problems in which the constraints are defined by means of Boolean formulas. The most common representation uses basic constraints that represent the typical connectives, such as *and, not* etc.

To reason about Boolean constraints one often uses rules such as:

$$\text{“for } x \wedge y = z\text{, if } z \text{ holds, then both } x \text{ and } y \text{ hold”} \tag{1}$$

or

$$\text{“for } x \vee y = z\text{, if } x \text{ does not hold, then } y = z \text{ holds.”} \tag{2}$$

These rules allow us to propagate the information that some values in a Boolean constraint are known. This type of inferences have been used for a long time. In McAllester (1980) they are explained informally; in McAllester (1990) they are called *Boolean constraint propagation*. In Simonis (1989) such rules are formulated explicitly and used to propagate known values through the circuit when generating tests for combinatorial circuits. More recently, these rules were used in Codognet & Diaz (1996) as a basis for an efficient implementation of a Boolean constraint solver.

In this paper we put together various simple observations concerning Boolean constraint propagation. The main difficulty lies in a proper setting up of the framework. Once this is done the results easily follow.

To start with, in Section 2, we answer the question in what sense a set of such rules can be complete. To this end we introduce a notion of completeness based on the notions of minimal rules and valid rules and show completeness for one set of such rules. In Section 3 we relate Boolean constraint propagation to unit propagation, a form of resolution for propositional logic, by explaining in what sense each method can be simulated by the other.

Next, in Section 4 we introduce proof rules that act on CSP's. This allows us to provide in Section 5 an alternative characterization for one set of rules by means of the notion of hyper-arc consistency of Mohr & Masini (1988) (we use here the terminology of Marriott & Stuckey (1998)). In Section 6 we clarify the status of another, more commonly used, set of such rules given for the *and* constraint in Simonis (1989) and for other connectives in Codognet & Diaz (1996). In the final section we relate Boolean constraint propagation to the `CHR` language of Frühwirth (1995).

1.2 Preliminaries

We review here the notions used in the sequel.

Consider a finite sequence of variables $Y := y_1, \ldots, y_k$ where $k \geq 0$, with respective domains $\mathcal{D} := D_1, \ldots, D_k$ associated with them. So each variable y_i ranges over the domain D_i. By a *constraint* C on Y we mean a subset of $D_1 \times \ldots \times D_k$. If C equals $D_1 \times \ldots \times D_k$, then we say that C is *solved*.

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 now an element $d := d_1, \ldots, d_n$ of $D_1 \times \ldots \times D_n$ and a subsequence $Y := x_{i_1}, \ldots, x_{i_\ell}$ of X. Then we denote by $d[Y]$ the sequence $d_{i_1}, \ldots, d_{i_\ell}$. By the *domain of* Y we mean the set of all tuples from $D_{i_1} \times \cdots \times D_{i_\ell}$. By a *solution* to $\langle \mathcal{C} \; ; \; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ we mean an element $d \in D_1 \times \ldots \times D_n$ such that for each constraint $C \in \mathcal{C}$ on a sequence of variables X we have $d[X] \in C$.

Next, we call a CSP *failed* if some of its domains is empty. Given two CSP's ϕ and ψ, we call ϕ a *reformulation* of ψ if the removal of solved constraints from ϕ and ψ yields the same CSP. We call two CSP's with the same sequence of variables *equivalent* if they have the same set of solutions. Clearly, two CSP's such that one is a reformulation of another are equivalent.

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' equals *c restricted to the domains* $D'_1, \ldots, D'_n$ if $c' = c \cap (D'_1 \times \ldots \times D'_n)$.

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 introduce the relevant 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 $x = 0$.

In the sequel x, y, z denote different Boolean variables. We distinguish four Boolean constraints:

- $x = y$; 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,

and interpret them in the expected way.

Finally, by a *Boolean constraint satisfaction problem*, in short *Boolean CSP*, we mean a CSP with Boolean domain expressions and each constraint of which is a Boolean constraint restricted to the adopted domains.

For example, the Boolean CSP

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

can be alternatively written as

$$\langle C_1, C_2 \; ; \; x = 1, y \in \{0, 1\}, z \in \{0, 1\} \rangle,$$

where $C_1 = \{(1, 1, 1), (1, 0, 0\}$ is a constraint on x, y, z and $C_2 = \{(1, 0)\}$ is a constraint on x, y.

In this paper we shall relate Boolean constraints to clauses as used in the resolution method. The relevant notions are the following ones.

A *literal* is a Boolean variable or its negation; a *clause* is a (possibly empty) disjunction of different literals. We denote the complement of the literal u by $\bar{u}$. A clause with a single literal is called a *unit clause*. We write $u \vee Q$ to denote a clause that contains the literal u as a disjunct; Q is the disjunction of the remaining literals.

2 The Proof System *BOOL* and Its Completeness

The rules such as the ones given in Section 1.1 can be naturally interpreted as implications over the constraint formed by the truth table of the connective in question. For instance, rule (1) can be viewed as the implication

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

over the *AND* constraint on the variables x, y, z determined by the table:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

With this interpretation "completeness" of a set of such rules can be naturally interpreted as the question whether the set implies all other valid rules. These concepts can be made precise as follows (see essentially Apt & Monfroy (1999)).

Definition 2.1 Consider a constraint C on a sequence of variables VAR, two disjoint non-empty subsequences X and Y of VAR, a tuple s of elements from the domain of X and a tuple t of elements from the domain of Y. We call $X = s \rightarrow Y = t$ a *rule* (*for* C).

- We say that $X = s \rightarrow Y = t$ is *valid* (*for* C) if for every tuple $d \in C$ the equality $d[X] = s$ implies the equality $d[Y] = t$.
- We say that $X = s \rightarrow Y = t$ is *feasible* (*for* C) if for some tuple $d \in C$ the equality $d[X] = s$ holds.

Suppose that a sequence of variables Z extends X and a tuple of elements u from the domain of Z extends s. We say then that $Z = u$ *extends* $X = s$. We now say that the rule $Z = u \rightarrow U = v$ *is implied* by the rule $X = s \rightarrow Y = t$ if $Z = u$ extends $X = s$ and $Y = t$ extends $U = v$.

We call a rule *minimal* if it is feasible and is not properly implied by a valid rule. Finally, we call a set of rules $\mathcal{R}$ for a constraint C *complete* if it consists of all minimal valid rules for C. $\square$

Take for example the *AND* constraint. The rule $z = 1 \rightarrow y = 1$ is implied by the rule $z = 1 \rightarrow x = 1, y = 1$. Since both of them are valid, the former rule is not minimal. Both rules are feasible, whereas the rule $z = 0, x = 1 \rightarrow y = 0$ is not. One can check that the rule $z = 1 \rightarrow x = 1, y = 1$ is minimal.

Consider now the set of rules presented in Table 1, where for the sake of clarity we attached to each implication the Boolean constraint in question. Call the resulting proof system *BOOL*.

A natural question arises whether some rules have been omitted in the proof system *BOOL*. Observe for example that no rule is introduced for $x \wedge y = z$ when $z = 0$. In this case either $x = 0$ or $y = 0$ holds, but $x = 0 \vee y = 0$ is not a legal conclusion of a rule. Alternatively, either $x = z$ or $y = z$ holds, but $x = z \vee y = z$ is not a legal conclusion of a rule either. The same considerations apply to $x \vee y = z$ when $z = 1$.

Also, we noted already that rule *AND 6* corresponds to rule (1). In contrast, no rule corresponds to rule (2). The following simple result clarifies the situation.

Theorem 2.2 (Completeness) *For each Boolean constraint the corresponding set of rules given in the proof system BOOL is complete.*

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

Table 1: Proof system *BOOL*

Proof. The claim follows by a straightforward exhaustive analysis of the valid rules for each considered Boolean constraint. Clearly, such an argument can be mechanized by generating all minimal rules for each Boolean constraint. This was done in Apt & Monfroy (1999) for the case of arbitrary finite constraints and rules of the form $X = s \rightarrow Y \neq t$ that have an obvious interpretation. Now, for the case of Boolean constraints each domain has two elements, so each rule of the form $X = s \rightarrow Y \neq t$ has a "dual" of the form $X = s \rightarrow Y = t'$, where t' is obtained from t by a bitwise complement. □

It is useful to mention that the abovementioned program, implemented in ECL^{i}PS^e, generated the appropriate rules for the *AND* constraint in 0.02 seconds and similarly for the other three Boolean constraints.

3 Relation to Unit Propagation

The considerations of the previous section clarify the matter of completeness. We still should explain how the rules of the proof system *BOOL* are supposed to be applied. To this end we consider finite sets of Boolean constraints and literals and interpret the rules as proof rules applied to such sets. We illustrate it by means of an example.

Consider *OR 3* rule. We interpret it as the following proof rule:

$$\frac{x \vee y = z, \neg x, z}{\neg x, y, z}$$

We define now the result of applying a rule of *BOOL* to a finite set of Boolean constraints and literals as expected: an application of the rule results in the replacement of (the subset corresponding to) the premise by (the subset corresponding to) the conclusion. This interpretation of the rules of *BOOL* allows us to derive conclusions that coincide with the informal use of such rules. In the case of *OR 3* rule the constraint $x \vee y = z$ is dropped as no other inference using it can be made, while the literal z is retained as other inferences using it are still possible.

In this section we relate so interpreted proof system $BOOL$ to unit propagation, a form of propositional resolution (see, e.g. Zhang & Stickel (1996)) that is a component of the Davis-Putnam algorithm for the satisfiability problem (see Davis & Putnam (1960)).

We consider two types of operations on a set of clauses:

- *unit resolution* (w.r.t. the literal u): given a unit clause u and a clause $\bar{u} \vee Q$ replace $\bar{u} \vee Q$ by Q,
- *unit subsumption* (w.r.t. the literal u): given a unit clause u and a clause $u \vee Q$ delete $u \vee Q$.

By *unit propagation* we mean one of the above two operations.

We now translate each Boolean constraint to a set of clauses as follows. We replace

- each equality constraint $x = y$ by the clauses $x \vee \neg y, \neg x \vee y$,
- each NOT constraint $\neg x = y$ by the clauses $x \vee y, \neg x \vee \neg y$,
- each AND constraint $x \wedge y = z$ by the clauses $\neg x \vee \neg y \vee z, x \vee \neg z, y \vee \neg z$,
- each OR constraint $x \vee y = z$ by the clauses $\neg x \vee z, \neg y \vee z, x \vee y \vee \neg z$.

Given a finite set of Boolean constraints and literals $\mathcal{S}$ we denote by $\phi_{\mathcal{S}}$ the resulting translation of this set into a set of clauses. It is straightforward to see that this translation maintains equivalence.

In what follows, given two sets of Boolean constraints and literals $\mathcal{S}_1$ and $\mathcal{S}_2$, we write $\mathcal{S}_1 \vdash_{BOOL} \mathcal{S}_2$ to denote the fact that $\mathcal{S}_2$ is obtained by a single application of a rule of the $BOOL$ system to $\mathcal{S}_1$, and $\mathcal{S}_1 \vdash^{\leq i}_{BOOL} \mathcal{S}_2$ to denote the fact that $\mathcal{S}_2$ is obtained by up to i applications of the rules of the $BOOL$ system to $\mathcal{S}_1$.

Analogously, given two sets of clauses ϕ_1 and ϕ_2, we write $\phi_1 \vdash_{UNIT} \phi_2$ to denote the fact that ϕ_2 is obtained by a single application of the unit propagation to ϕ_1, and $\phi_1 \vdash^{\leq i}_{UNIT} \phi_2$ to denote the fact that ϕ_2 is obtained by up to i applications of the unit propagation to ϕ_1.

The following result relates the proof system $BOOL$ to unit propagation.

Theorem 3.1 (Reduction 1) *Consider two finite sets of Boolean constraints and literals $\mathcal{S}_1$ and $\mathcal{S}_2$. Suppose that $\mathcal{S}_1 \vdash_{BOOL} \mathcal{S}_2$. Then $\phi_{\mathcal{S}_1} \vdash^{\leq 4}_{UNIT} \phi_{\mathcal{S}_2}$.*

Proof. We need to analyze each of the 20 rules of $BOOL$. We illustrate the argument on one, arbitrary selected rule, *OR 3*.

Suppose that $\mathcal{S}_2$ is the result of applying rule *OR 3* to $\mathcal{S}_1$. Recall that this rule is interpreted as

$$\frac{x \vee y = z, \neg x, z}{\neg x, y, z}$$

The assumption of this rule translates to the following set of clauses:

$$\{\neg x \vee z, \neg y \vee z, x \vee y \vee \neg z, \neg x, z\}.$$

By the application of the unit resolution w.r.t. z we obtain the set

$$\{\neg x \vee z, \neg y \vee z, x \vee y, \neg x, z\},$$

from which by two applications of the unit subsumption w.r.t. z we obtain the set

$$\{x \vee y, \neg x, z\}.$$

By the application of the unit resolution w.r.t. $\neg x$ we now obtain the set

$$\{\neg x, y, z\}$$

which corresponds to the conclusion of rule *OR 3*.

For other rules the argument is equally straightforward. □

The converse relation is a bit more complicated since to translate clauses to Boolean constraints we need to use auxiliary variables. First, we translate each expression of the form $Q = z$, where Q is a clause and z a variable, to a finite set of Boolean constraints and literals. We proceed by induction on the number of literals in Q.

If Q is a unit clause, then $Q = z$ is either an equality constraint or a NOT constraint and we put $trans(Q = z) := \{Q = z\}$. Otherwise Q is of the form $u \vee Q_1$ and we define

$$trans(x \vee Q_1 = z) := \{x \vee y = z\} \cup trans(Q_1 = y),$$

where y is a fresh variable,

$$trans(\neg x \vee Q_1 = z) := \{\neg x = v, v \vee y = z\} \cup trans(Q_1 = y),$$

where v and y are fresh variables.

Finally, we put for a unit clause u

$$trans(u) := \{u\},$$

and for a non-unit clause Q

$$trans(Q) := \{z\} \cup trans(Q = z),$$

where z is a fresh variable.

Note that for a non-unit clause Q the resulting finite set of Boolean constraints and literals $trans(Q)$ depends on the order in which the literals of Q are selected and on the specific choice of the fresh variables, so it is not uniquely determined. However, it is clear that for each such translation $trans(Q)$, the clause Q is equivalent to $\exists \bar{z}\, trans(Q)$, where $\bar{z}$ is the sequence of the introduced fresh variables.

Given now a finite set of clauses ϕ we translate each of its clauses separately and call thus obtained finite set of Boolean constraints and literals a *translation of ϕ to a finite set of Boolean constraints and literals.*

Below, given two sets of Boolean constraints and literals $\mathcal{C}$ and $\mathcal{S}$ we say that $\mathcal{C}$ *semantically follows from a set* $\mathcal{S}$ if every valuation that satisfies $\mathcal{S}$ can be extended to a valuation that satisfies $\mathcal{C}$.

We then have the following result.

Theorem 3.2 (Reduction 2) *Consider two finite sets of clauses ϕ_1 and ϕ_2. Suppose that $\phi_1 \vdash_{UNIT} \phi_2$. Then for some translations $\mathcal{S}_1$ and $\mathcal{S}_2$ of ϕ_1 and ϕ_2 to finite sets of Boolean constraints and literals and some set of Boolean constraints and literals $\mathcal{C}$ we have $\mathcal{S}_1 \vdash^{\leq 3}_{BOOL} \mathcal{S}_2 \cup \mathcal{C}$, where $\mathcal{C}$ semantically follows from $\mathcal{S}_2$.*

Informally, the reduction from $\mathcal{S}_1$ to $\mathcal{S}_2$ yields additionally some redundant set of Boolean constraints and literals $\mathcal{C}$.

Proof. Consider first the unit resolution. It leads to a replacement of $\bar{u} \vee Q$ by Q in presence of the unit clause u. Suppose that u is a Boolean variable x. Then $\bar{u}$ is $\neg x$.

We now have for some fresh variables v, y and z

$$trans(\neg x \vee Q) = \{z\} \cup \{\neg x = v, v \vee y = z\} \cup trans(Q = y).$$

So the clauses x and $\neg x \vee Q$ translate to the set of Boolean constraints and literals

$$\{x, z, \neg x = v, v \vee y = z\} \cup trans(Q = y).$$

By the application of the *NOT 1* rule we now obtain the set

$$\{x, z, \neg v, v \vee y = z\} \cup trans(Q = y),$$

from which by the *OR 3* rule we obtain

$$\{x, z, \neg v, y\} \cup trans(Q = y). \tag{3}$$

Now, if Q is a unit clause, then the set (3) equals

$$\{x, z, \neg v, y, Q = y\}$$

from which we get by the *EQU 2* rule

$$\{x, z, \neg v, y, Q\},$$

i.e., the set $trans(x) \cup trans(Q) \cup \{z, \neg v, y\}$. Since v, y and z are fresh, $\{z, \neg v, y\}$ semantically follows from $trans(x) \cup trans(Q)$.

If Q is a non-unit clause we can assume that

$$trans(Q) = \{y\} \cup trans(Q = y),$$

so the set (3) equals $trans(x) \cup trans(Q) \cup \{z, \neg v\}$. Since v and z are fresh, $\{z, \neg v\}$ semantically follows from $trans(x) \cup trans(Q)$.

The argument in case u is negation of a Boolean variable is even more straightforward.

Consider now the unit subsumption. It leads to a deletion of the clause $u \vee Q$ in presence of the unit clause u. Suppose that u is $\neg x$ for some Boolean variable x. We have for some fresh variables v, y and z

$$trans(\neg x \vee Q) = \{z, \neg x = v, v \vee y = z\} \cup trans(Q = y).$$

So the clauses $\neg x$ and $\neg x \vee Q$ translate to the set of Boolean constraints and literals

$$\{\neg x, z, \neg x = v, v \vee y = z\} \cup trans(Q = y).$$

By the application of the *NOT 2* rule we now obtain the set

$$\{\neg x, z, v, v \vee y = z\} \cup trans(Q = y),$$

from which by the *OR 1* rule we obtain

$$\{\neg x, z, v\} \cup trans(Q = y).$$

It is now easy to see that the set $\{z, v\} \cup trans(Q = y)$ semantically follows from $\{\neg x\}$. Indeed, a straightforward proof by induction shows that for any clause Q the set $trans(Q = y)$ is satisfiable. □

The above two results clarify the relationship between Boolean constraint propagation and unit propagation. They show that each method can be simulated by another in constant time, albeit the simulation of the unit propagation by means of the Boolean constraint propagation leads to a generation of redundant constraints.

A relation between Boolean constraint propagation and the Davis-Putnam algorithm was already mentioned in McAllester (1980, page 1), where it is stated without any further explanation that "propositional [i.e., Boolean] constraint propagation [...] was originally described, in essence, by Davis & Putnam (1960) ". But to our knowledge this connection was not made precise.

4 A Proof Theoretic Framework

We now proceed towards another characterization of the proof system *BOOL* in constraint processing terms. In the previous section we considered finite sets of Boolean constraints and literals. We now need to translate them into Boolean CSP's by interpreting in an appropriate way the literals belonging to such a set.

Given a Boolean variable x there are four sets of literals concerning x. We interpret each of them as a Boolean domain expression, as follows:

- $\emptyset$ by $x \in \{0, 1\}$,
- $\{x\}$ by $x \in \{1\}$,
- $\{\neg x\}$ by $x \in \{0\}$,
- $\{x, \neg x\}$ by $x \in \emptyset$.

This interpretation entails a translation of finite sets of Boolean constraints and literals to Boolean CSP's. For example, the set $\{x \vee y = z, \neg x, z\}$ (that corresponds to the premise of *OR 3* rule) translates to the Boolean CSP

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

It is straightforward to see that this translation preserves equivalence in the sense that $(d_1, \ldots, d_n)$ is a solution to a Boolean CSP $\mathcal{P} := \langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n\rangle$ iff the assignment $(x_1/d_1, \ldots, x_n/d_n)$ satisfies the original set of Boolean constraints and literals.

This translation also leads to another interpretation of the rules of the proof system *BOOL*. We interpreted them as rules on the finite sets of Boolean constraints and literals. By means of the above translation they become rules on Boolean CSP's.

Note that for a set L of literals concerning x that translates into the Boolean domain expression $x \in D$, the set $L \cup \{x\}$ translates into the Boolean domain expression $x \in D \cap \{1\}$, and similarly for the literal $\neg x$. Consequently, the rule

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

translates into

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

and the rule

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

translates into

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

In addition the rule

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

that naturally corresponds to rule (2) of in Section 1.1, translates into

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

This brings us to the proof theoretic framework introduced in Apt (1998). We briefly recall the relevant definitions. The crucial concept that we need is that of a CSP being closed under the applications of a proof rule. In the above paper we introduced two types of proof rules for CSP's: *deterministic* and *splitting*. Here we only use the deterministic ones. These rules are of the form

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

where ϕ and ψ are CSP's.

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

$$\frac{\langle \mathcal{C}_1 \ ; \ \mathcal{D}_1\rangle}{\langle \mathcal{C}_2 \ ; \ \mathcal{D}_2\rangle} \tag{4}$$

We then say that rule (4) *can be applied* to $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{D} \cup \mathcal{D}_1\rangle$ and call

$$\langle \mathcal{C} \cup \mathcal{C}_2 \ ; \ \mathcal{D} \cup \mathcal{D}_2\rangle$$

the *result of applying it to* $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{D} \cup \mathcal{D}_1\rangle$. If $\langle \mathcal{C} \cup \mathcal{C}_2 \ ; \ \mathcal{D} \cup \mathcal{D}_2\rangle$ is not a reformulation of $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{D} \cup \mathcal{D}_1\rangle$, then we say that it is the result of a *relevant application of rule* (4) *to* $\langle \mathcal{C} \cup \mathcal{C}_1 \ ; \ \mathcal{D} \cup \mathcal{D}_1\rangle$.

Finally, given a CSP ϕ and a deterministic rule R, we say that ϕ is *closed under the applications of* R if either R cannot be applied to ϕ or no application of it to ϕ is relevant.

Take for example the Boolean CSP $\phi := \langle x \wedge y = z \ ; \ x = 1, y = 0, z = 0\rangle$. This CSP is closed under the applications of the 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 $\psi := \langle y = z \ ; \ x = 1, y = 0, z = 0\rangle$. 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 $\langle y = z \ ; \ x = 1, y \in \{0,1\}, z \in \{0,1\}\rangle$ is not a reformulation of ϕ.

In what follows we identify the rules of the proof system *BOOL* with their counterparts that act on Boolean CSP's. At this stage we introduced two interpretations of the rules of the proof system *BOOL*: one on the finite sets of Boolean constraints and literals and the other on Boolean CSP's. It is straightforward to check that these interpretations correspond in the following sense. Consider two finite sets of Boolean constraints and literals $\mathcal{S}_1$ and $\mathcal{S}_2$ that translate respectively to the Boolean CSP's $\mathcal{P}_1$ and $\mathcal{P}_2$ and a rule r of *BOOL*. Then in the first interpretation r transforms $\mathcal{S}_1$ into $\mathcal{S}_2$ iff in the second interpretation it transforms $\mathcal{P}_1$ into $\mathcal{P}_2$.

It is worthwhile to note that the Characterization Theorem 5.2 can be proved indirectly by using the theoretical results established in Apt & Monfroy (1999) together with the output of the already mentioned in Section 2 program that automatically generates proof rules from the truth tables, or more generally, from a table representing a finite constraint.

5 Relation to Hyper-arc Consistency

We now return to CSP's. In Mohr & Masini (1988) a generalization of the notion of arc consistency of Mackworth (1977) from binary constraints to arbitrary constraints was introduced. Let us recall the definition.

Definition 5.1

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

The following result characterizes the proof system *BOOL* in terms of the notion of hyper-arc consistency for Boolean CSP's.

Theorem 5.2 (Characterization) *A non-failed Boolean CSP is closed under the applications of the rules of the proof system BOOL iff it is hyper-arc consistent.*

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. Let D_x, D_y and D_z be respectively the domains of x, y and z.
$(\Rightarrow)$ 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 *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 $(d,0,0) \in C$.

($\Leftarrow$) 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 hyper-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 hyper-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 hyper-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 hyper-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.

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

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 hyper-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 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 hyper-arc consistent.

In view of the fact that all considered proof rules preserve equivalence, the above theorem shows that to reduce a Boolean CSP to an equivalent one that is either failed or hyper-arc consistent it suffices to close it under the applications of the rules of the *BOOL* system. This provides a straightforward algorithm for enforcing hyper-arc consistency for Boolean constraints. We shall return to this point in the final section.

6 The Proof System of Codognet and Diaz

Usually, slightly different proof rules are introduced when dealing with Boolean constraints. For example, in Codognet & Diaz (1996) the set of rules given in Table 2 is considered. 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'	$x \wedge y = z, x = 1 \rightarrow y = z$
AND 2'	$x \wedge y = z, y = 1 \rightarrow x = z$
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'	$x \vee y = z, x = 0 \rightarrow y = z$
OR 3'	$x \vee y = z, y = 0 \rightarrow x = z$
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 rules *AND 1'*, *AND 2'*, *OR 2'* and *OR 3'* introduce constraints in their conclusions. *OR 2'* rule corresponds to rule (2) of Section 1.1.

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.)

The *AND* rules of the *BOOL'* system can be found (in a somewhat different format) in Simonis (1989). 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 6.1 *If a non-failed Boolean CSP is closed under the applications of the rules of the proof system BOOL', then it is hyper-arc consistent.*

Proof. The proof relies on the following immediate observation.

Claim *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 hyper-arc consistency Theorem 5.2.

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 other Boolean constraints is analogous and omitted. □

In contrast 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 hyper-arc consistent but it is not closed under the applications of the *AND 1'* rule.

In general, there are four such "problematic" CSP's. In each of them the single *AND* or *OR* constraint can be reduced to an equality constraint. These four CSP's are used in the following definition.

Definition 6.2 We call a Boolean CSP *limited* if none of the following four CSP's forms a subpart of it:

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

The idea is that if we exclude these "problematic" CSP, then hopefully we prevent the situation that a CSP is hyper-arc consistent but is not closed under the applications of the *AND 1'* (respectively *AND 2', OR 2'* or *OR 3'*) rule. This is exactly what the following theorem states.

Theorem 6.3 *If a non-failed Boolean CSP is limited and hyper-arc consistent, then it is closed under the applications of the rules of the proof system BOOL'.*

Proof. In view of the hyper-arc consistency Theorem 5.2 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 hyper-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 hyper-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. □

To summarize: for Boolean CSP's that are limited the respective closures under the rules of the proof systems *BOOL* and *BOOL'* coincide.

7 Relation to the CHR Language

The rules such as the ones given in the proof system *BOOL* can be straightforwardly represented as so-called simplification rules of the CHR language of Frühwirth (1995). The CHR language is part of the ECL^{i}PS^e system (see A. Aggoun et al. (1995)). For a more recent and more complete overview of CHR see Frühwirth (1998). For example *AND 6* rule, so

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

is written in the syntax of CHR as

```
and(X,Y,Z) <=> Z = 1 | X = 1,Y = 1.
```

In fact, such CHR rules for the *AND* constraint can be already found in Frühwirth, Herold, Küchenhoff, Provost, Lim, Monfroy & Wallace (1992). They amount to the corresponding *AND* rules of the *BOOL'* system. Boolean constraints form a prime example for an effective use of CHRs. A CHR program that corresponds to the proof system *BOOL* or *BOOL'* when combined with a labeling procedure constitutes a natural decision procedure for Boolean CSP's. The Characterization Theorem 5.2 shows that the CHR rules corresponding to the *BOOL* system implement hyper-arc consistency.

8 Conclusions

In this paper we collected a number of simple but hopefully useful observations on Boolean constraint propagation rules. First of all, we clarified in what sense one set of such rules is complete. Then we showed that Boolean constraint propagation is in fact equivalent to unit propagation, a form of resolution for propositional logic. The reduction in each direction can be achieved in constant time.

This shows that given a combinatorial problem that can be naturally formalized using Boolean constraints (for example, a problem concerning combinatorial circuits) it is useless to translate it to a clausal form and subsequently employ unit propagation: in such case Boolean constraint propagation achieves the same effect. Conversely, it is useless to translate a clausal form representation to a representation that uses Boolean constraints with the aim of employing Boolean constraint propagation: in this case unit propagation achieves the same effect.

The subsequent characterization of the introduced set of Boolean constraint propagation rules by means of the hyper-arc consistency notion shows that this set of rules is in some sense optimal. The notion of hyper-arc consistency also allowed us to differentiate between two sets of such rules proposed in the literature.

Acknowledgement

We thank Rina Dechter, Thom Frühwirth and the referees for helpful comments.

References

A. Aggoun et al. (1995), *ECL^iPS^e 3.5 User Manual*, Munich, Germany.

Apt, K. R. (1998), 'A proof theoretic view of constraint programming', *Fundamenta Informaticae* **33**(3), 263–293. Available via `http://xxx.lanl.gov/archive/cs/`.

Apt, K. R. & Monfroy, E. (1999), Automatic generation of constraint propagation algorithms for small finite domains, *in* J. Jaffar, ed., 'Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)', Lecture Notes in Computer Science, Springer-Verlag, Alexandra, Virginia, USA. Available via `http://xxx.lanl.gov/archive/cs/`.

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

Davis, M. & Putnam, H. (1960), 'A computing procedure for quantification theory', *Journal of the ACM* **7**(3), 201–215.

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.).

Frühwirth, T., Herold, A., Küchenhoff, V., Provost, T. L., Lim, P., Monfroy, E. & Wallace, M. (1992), Constraint logic programming: An informal introduction, *in* G. Comyn, N. E. Fuchs & M. J. Ratcliffe, eds, 'Logic Programming in Action', LNCS 636, Springer-Verlag, pp. 3–35.

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

Marriott, K. & Stuckey, P. (1998), *Programming with Constraints*, The MIT Press, Cambridge, Massachusetts.

McAllester, D. (1980), 'An outlook on truth maintenance'. MIT, Artificial Intelligence Laboratory, AI Memo No. 551.

McAllester, D. (1990), Truth maintenance, *in* 'AAAI-90: Proceedings 8th National Conference on Artificial Intelligence', pp. 1109–1116.

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.

Simonis, H. (1989), Test generation using the Constraint Logic Programming language CHIP, *in* G. Levi & M. Martelli, eds, 'ICLP'89: Proceedings 6th International Conference on Logic Programming', MIT Press, Lisbon, Portugal, pp. 101–112.

Zhang, H. & Stickel, M. (1996), An efficient algorithm for unit propagation, *in* 'Proc. of the Fourth International Symposium on Artificial Intelligence and Mathematics', Ft. Lauderdale, Florida.