

Eliminating Variables in General Constraint Logic*

Rolf Haenni

Computer Science Department

University of California, Los Angeles, CA, 90095

Email: rolf.haenni@gmx.net

Web Site: haenni.shorturl.com

Abstract

Drawing inferences from a set of general constraint clauses is known as a difficult problem. A general approach is based on the idea of eliminating some or all variables involved. In the particular case of propositional logic, this approach leads to a simple procedure that incorporates the well-known resolution principle. The purpose of this paper is to show how the resolution principle can be extended to constraint logic where the knowledge is given as a set of constraint clauses. The result is a general variable elimination method. The paper shows that the elimination problem can always be reduced to the problem of eliminating the variable from a (conjunctive) set of atomic constraints. Variable elimination has a number of possible applications such as satisfiability testing, hypotheses testing, constraint solving, argumentative reasoning, and many others.

1 Introduction

The theory of *information algebras* and *information systems* [25, 27] defines a general theoretical framework for representing abstract pieces of information. This theory is based on a small set of axioms over the two basic operations of *combination* (aggregation of information) and *marginalization* (focusing to a more specific sub-question). This leads

*Research supported by scholarship No. 8220-061232 of the Swiss National Science Foundation.

to a generic architecture for solving different types of inference problems. The architecture implements the idea of distributed *local computations* which has been discovered for valuation systems [30, 33] and is known as *join tree propagation*. A similar method that puts the focus on variable elimination is called *fusion algorithm* [31, 32]. The same idea is also called *bucket elimination* [10].

One particular case of an information system is obtained from the framework of *constraint logic*. Constraints are statements about groups of variables, each variable having an individual set of possible values. Every constraint restricts the possible configurations of a group of variables. Constraint logic is the language obtained from using constraints together with the classical logical connectives. It includes the classical propositional logic as a special case [26]. Other important special cases are *set constraint logic* [3, 14, 15, 16] (also known as *multivalent logic* [20, 21] or signed logic [17, 18]), as well as general systems of *linear* or *non-linear equations* or *inequalities*.

Constraint logic is a general and convenient tool for representing many different types of information. Usually, the given knowledge is expressed as a conjunctive set of relatively small constraint formulae. The difficult operation is then the marginalization of the given information to a subset of variables. A general solution for the marginalization problem is the idea of eliminating successively the corresponding irrelevant variables. The basic operation is then the *variable elimination* (also known as *quantifier elimination* in first-order logic [28] or more generally as *fusion*[31, 32] and *bucket elimination* [10]). In propositional logic, eliminating a variable means essentially computing a number of new resolvents according to the well-known *resolution principle* [26]. Similar methods are known for (multivalent) set constraint logic [3, 8, 18, 20, 21] and for some simple cases of linear equations and inequalities [4, 5, 6, 7, 11, 19, 22, 28, 29, 34, 35]. Variable elimination has a number of possible applications such as satisfiability testing, hypotheses testing, constraint solving, argumentative reasoning under uncertainty (which is the author's main research topic [1, 2, 13, 14, 12]), and many others.

The purpose of this paper is to present a general variable elimination method for constraint logic. This is a possible way of understanding the classical resolution principle from a more general point of view. The paper is organized as follows: Section 2 introduces constraint logic and describes the importance of variable elimination; Section 3 presents a general variable elimination procedure; Section 4 discusses two important special cases of constraint logic; finally, Section 5 contains some concluding remarks.

2 Constraint Logic

Propositional logic can be seen as a formal language for describing statements about binary variables. This is sufficient for expressing a certain class of problems. However, describing the world on the basis of binary variables is sometimes not sufficient. For that reason, propositional logic can be generalized to *constraint logic* (CL). The idea is that arbitrary variables are allowed, each of them having an individual set of possible values. Constraints about the true values of some variables are then the atomic expressions of the language. In this way, the expressiveness of classical logic is extended significantly. Note that constraint logic includes propositional logic, set constraint logic, and systems of linear and non-linear equations and inequalities as special cases.

Constraint logic is based on a finite set V of variables. Each variable $x \in V$ has a set Θ_x of possible values. Exactly one value $\theta \in \Theta_x$ is supposed to be the true value of the variable x . The set of possible values Θ_x is called *frame* of x . If $X \subseteq V$ is a group of variables, then the product space

$$\Theta_X = \prod_{x \in X} \Theta_x \quad (1)$$

is called *product frame* of X . Each *configuration* (or *interpretation*) $i \in \Theta_X$ assigns a value to each variable $x \in X$. A *constraint* C is a description of a subset of $I_X(C) \subseteq \Theta_X$ of configurations. $Vars(C) = X$ denotes the corresponding group of variables involved. Furthermore, \mathcal{C}_X denotes the set of all possible constraints over X . At the moment, the language \mathcal{C}_X is not further specified (see Subsection 2.1 for examples).

A constraint $C \in \mathcal{C}_X$ can be considered as predicate that evaluates to *true* for a given configuration $i \in \Theta_X$, if $i \in I_X(C)$. Otherwise, C evaluates to *false*. The symbol \top always evaluates to *true*. Furthermore, \perp always evaluates to *false*. Constraints together with the symbols \perp and \top can be used to build compound *constraint formulae*:

- (1) constraints, \perp and \top are constraint formulae;
- (2) if γ is a constraint formula, then $\neg\gamma$ is a constraint formula;
- (3) if γ and δ are constraint formulae, then $(\gamma \wedge \delta)$, $(\gamma \vee \delta)$, $(\gamma \rightarrow \delta)$, and $(\gamma \leftrightarrow \delta)$ are constraint formulae.

Often, unnecessary parentheses can be omitted. The set \mathcal{L}_V of all constraint formulae over V is called *constraint language* over V . The group of variables involved in a constraint formula $\gamma \in \mathcal{L}_V$ is denoted by $Vars(\gamma)$. A formula γ is *relevant* for a variable $x \in V$, if $x \in Vars(\gamma)$. Otherwise, the formulae γ is *irrelevant* for x .

Let $i \in \Theta_V$ be a configuration that determines the truth values of the constraints contained in $\gamma \in \mathcal{L}_V$. The truth value of the formula itself can then be determined in the same way as in propositional logic. $I_V(\gamma) \subseteq \Theta_V$ denotes the set of all configurations for which a formula γ evaluates to *true*. A constraint formula γ *entails* a constraint formula δ (denoted by $\gamma \models \delta$), if and only if δ evaluates to *true* under all configurations for which γ evaluates to *true*, that is if $I_V(\gamma) \subseteq I_V(\delta)$. Furthermore, γ and δ are *equivalent* (denoted by $\gamma \equiv \delta$), if and only if the truth values of γ and δ are the same under all possible configurations, that is if $I_V(\gamma) = I_V(\delta)$. Note that equivalent constraint formulae represent exactly the same information.

A constraint $C \in \mathcal{C}_X$ is called *regular*, if $C \not\equiv \top$ and $C \not\equiv \perp$. Note that non-regular constraints appearing in a constraint formula can always be used to simplify the formula. For example, $\gamma \wedge \top \equiv \gamma$, $\gamma \wedge \perp \equiv \perp$, $\gamma \vee \top \equiv \top$, $\gamma \vee \perp \equiv \gamma$, $\neg \top \equiv \perp$, $\neg \perp \equiv \top$, etc.

A *constraint clause* is a disjunction $C_1 \vee \dots \vee C_m$ of regular constraints. A constraint clause D is called *minimal*, if no constraint in D represents a subset of configurations of another constraint in D . A special case of a minimal constraint clause is the *empty clause* \perp . The set of all minimal constraint clauses is denoted by \mathcal{D}_V . Note that non-minimal constraint clauses can always be transformed into equivalent minimal constraint clauses by dropping the corresponding constraints.

A set $\Sigma = \{D_1, \dots, D_s\}$ of minimal constraint clauses is considered as the *knowledge base*. The clauses in Σ are interpreted as a conjunction $D_1 \wedge \dots \wedge D_s$. Therefore, Σ can be seen as a conjunctive normal form (CNF) without negations. Note that such a CNF can always be obtained from any arbitrary formula $\gamma \in \mathcal{L}_V$, if every negated constraint $\neg C$ of a constraint $C \in \mathcal{C}_X$ can be replaced by an equivalent constraint $C' \equiv \neg C$ such that $C' \in \mathcal{C}_X$.

The knowledge base Σ is called *minimal*, if no constraint clause in Σ entails another constraint clause in Σ . Again, non-minimal knowledge bases can always be transformed into equivalent minimal knowledge bases by dropping the corresponding constraint clauses. If Σ is not minimal, then $\mu\Sigma$ denotes the corresponding minimal knowledge base. $I_V(\mu\Sigma) = I_V(\Sigma) = I_V(D_1 \wedge \dots \wedge D_s)$ is the set of configurations that evaluate to *true* with respect to Σ .

2.1 Examples of Constraints

In order to illustrate the wide-spread applicability of constraint logic, consider the following examples of different classes of constraints:

- *Propositional Constraints:* If a variable $x \in V$ has only two possible values, $|\Theta_x| = 2$, then only two different regular constraints are possible. Let $\Theta_x = \{0, 1\}$, for example, be the frame of x , then $\langle x = 1 \rangle$ and $\langle x = 0 \rangle$ are the only regular constraints. In such a situation, $\langle x = 1 \rangle$ is often abbreviated by x and $\langle x = 0 \rangle$ by $\neg x$, or vice versa.
- *Set constraints:* Let Θ_x be an arbitrary set of possible values of x . A set constraint is an expression of the form $\langle x \in F \rangle$, where F is a non-empty subset of Θ_x . *Examples:* $\Theta_x = \{a, b, c, d, e\}$
 - $\langle x \in \{a, b\} \rangle$,
 - $\langle x \in \{b, c, d\} \rangle$,
 - $\langle x \in \{e\} \rangle$,
 - etc.

Clearly, propositional constraints are special cases of set constraints.

- *Interval constraints:* Let $\Theta_x = \mathbb{R}$ be the frame of the variable x . An interval constraint is an expression of the form $\langle x \in I \rangle$, where I is an interval of the form $[a, b]$, $[a, b)$, $(a, b]$ or (a, b) with $a, b \in \mathbb{R}$. *Examples:*
 - $\langle x \in [2, 5] \rangle$,
 - $\langle x \in [-2, 7) \rangle$,
 - $\langle x \in (6, \infty] \rangle$.
 - etc.

General interval constraints are expressions of the form $\langle x \in I_1 \cup \dots \cup I_n \rangle$ with intervals I_i as described above.

- *Linear and non-linear equations and inequalities:* Let $R \subseteq V$ be a set of variables with $\Theta_x = \mathbb{R}$ for all $x \in R$. Every equation or inequality over the variables R can be considered as a constraint. *Examples:*

- $\langle x = 17 \rangle$,
- $\langle x - 3y + 2z = 6 \rangle$,
- $\langle x \leq 25 \rangle$,
- $\langle 4x - 3y + z > 6 \rangle$,
- $\langle x^2 + y^2 = 25 \rangle$,
- $\langle 3x^2 - 2\sin(y) < 4 \rangle$,
- $\langle \frac{\sqrt{2x-y}}{4z^2} \geq 7 \rangle$,
- etc.

2.2 Variable Elimination

Eliminating variables has been discovered as an important basic operation for solving problems in the domain of propositional logic [9, 23, 26], in the framework of set constraint logic [3, 8, 18, 20, 21], as well as in systems of linear equations and inequalities [5, 6, 7, 19, 22, 24, 28, 34, 35]. General techniques that use variable elimination is Shenoy's *fusion algorithm* [31, 32] and Dechter's *bucket elimination* [10].

The same idea will now be developed for the case of general constraint logic. The basic idea is the following. Let $\Sigma = \{D_1, \dots, D_s\}$ the available knowledge base and $x \in V$ a variable to be eliminated. The problem then is to find a new minimal set $\Sigma' = \{D'_1, \dots, D'_s\}$ of minimal constraint clauses $D'_i \in \mathcal{C}_{V'}$, $V' = V \setminus \{x\}$, such that $I_{V'}(\Sigma')$ is the projection of the set $I_V(\Sigma)$ from V to V' . Such an operation is denoted by

$$\Sigma' = \text{Elim}_x(\Sigma). \quad (2)$$

Note that there may be different (but equivalent) sets Σ' satisfying the above condition.

Similarly, if $X \subseteq V$ is a set of variables to be eliminated, then the problem is to find a new set $\Sigma' = \{D'_1, \dots, D'_s\}$ of new minimal constraint clauses $D'_i \in \mathcal{C}_{V'}$, $V' = V \setminus X$, such that $I_{V'}(\Sigma')$ is the projection of the set $I_V(\Sigma)$ from V to V' . Such an operation is denoted by

$$\Sigma' = \text{Elim}_X(\Sigma). \quad (3)$$

This problem can be solved by successively eliminating all the variables in X . More precisely, if $X = \{x_1, \dots, x_q\}$ is the set of variables to be eliminated, then a possible

solution is given by

$$Elim_X(\Sigma) = Elim_{x_q} \circ \dots \circ Elim_{x_1}(\Sigma). \quad (4)$$

Note that any of the $q!$ elimination sequences can be used for this process. However, different elimination sequences may result in different (but equivalent) sets Σ' . Variable elimination is useful for different applications. The following list shows some important examples:

- *Testing for Satisfiability*: One of the most important problems in the framework of constraint logic is to decide whether Σ is satisfiable or not, that is to find out whether $I_V(\Sigma)$ is empty or not. In fact, Σ is satisfiable if and only if $Elim_V(\Sigma) = \emptyset$.
- *Hypotheses Testing*: Another important problem is to decide whether a hypothesis $H \in \mathcal{L}_V$ follows from the given knowledge base Σ or not. Let Σ_H be a set of clauses such that Σ_H and $\neg H$ are equivalent. Then H follows from Σ if and only if $Elim_V(\Sigma \cup \Sigma_H) = \{\perp\}$.
- *Constraint Solving*: If Σ is satisfiable, then another problem consists in finding one or several configurations $i \in I_V(\Sigma)$. Such a configuration can be constructed by a sequence of variable eliminations. First, $\Sigma_{x_1} = Elim_{V \setminus \{x_1\}}(\Sigma)$ is computed. Σ_{x_1} describes the set of possible values of x_1 . One of these values is arbitrarily selected and added to Σ . Then a value for x_2 is arbitrarily selected from $\Sigma_{x_2} = Elim_{V \setminus \{x_2\}}(\Sigma)$ and added to Σ , and so on. Finally, the values selected for all the variables $x_i \in V$ form a configuration $i \in I_V(\Sigma)$.
- *Constraint Projection*: If $X \subseteq V$ is a set of variables, the constraint projection problem consists in finding a new set Σ' of constraint clauses $D \in \mathcal{D}_X$ such that $I_X(\Sigma')$ is the projection of the set $I_V(\Sigma)$ from V to X . This problem is equivalent to the problem of computing $Elim_{V \setminus X}(\Sigma)$.
- *Computing Arguments for Hypotheses* [1, 2, 12, 13, 14]: If a subset $A \subseteq V$ of variables is declared as *assumptions*, then the problem is to find arguments α (conjunctive sets of constraints over assumptions) such that a hypothesis $H \in \mathcal{L}_V$ follows from $\alpha \cup \Sigma$. Such arguments for H can be derived from $Elim_{V \setminus A}(\Sigma \cup \Sigma_H)$, where Σ_H is a set of clauses such that Σ_H and $\neg H$ are equivalent.

All the problems of the above list can therefore be solved by eliminating variables. The basic operation, that is the elimination of a single variable $x \in V$, will be discussed in the following subsection. The result will be a resolution procedure that contains the resolution principle of propositional and set constraint logic as special cases.

3 The Variable Elimination Procedure

Let $x \in V$ be the variable to be eliminated from $\Sigma = \{D_1, \dots, D_s\}$. The first step of the elimination procedure is to decompose Σ into two disjoint subsets Σ_X and Σ_R :

$$\Sigma = \underbrace{\{D_1, \dots, D_r\}}_{x \in \text{Vars}(D_i)} \cup \underbrace{\{D_{r+1}, \dots, D_s\}}_{x \notin \text{Vars}(D_i)} = \Sigma_X \cup \Sigma_R. \quad (5)$$

The set $\Sigma_X = \{D_1, \dots, D_r\}$ contains the constraint clauses which are relevant for x . These are the important clauses for the elimination process. Let $D = C_1 \vee \dots \vee C_m$ be such a constraint clause of Σ_X . Then, in a similar way as above for Σ , the clause D can be decomposed into two sub-clauses X and R :

$$D = \underbrace{C_1 \vee \dots \vee C_k}_{x \in \text{Vars}(C_i)} \vee \underbrace{C_{k+1} \vee \dots \vee C_m}_{x \notin \text{Vars}(C_i)} = \begin{cases} X \vee R, & \text{if } k < m, \\ X, & \text{if } k = m. \end{cases} \quad (6)$$

Note that the first sub-clause X is never empty. In contrast, the second sub-clause R may be empty in some cases. Therefore, the set $\Sigma_X = \{D_1, \dots, D_r\}$ of relevant clauses has in general the following form:

$$\Sigma_X = \{X_1 \vee R_1, \dots, X_q \vee R_q, X_{q+1}, \dots, X_r\}. \quad (7)$$

Example 1 Let $\Sigma = \{D_1, D_2, D_3, D_4, D_5\}$ be a set of five minimal constraint clauses over a set $V = \{f, g, h, q, r, x, y, z\}$ of variables with $\Theta_f = \Theta_g = \Theta_h = \{0, 1\}$, $\Theta_q = \Theta_r = \{a, b, c, d\}$, and $\Theta_x = \Theta_y = \Theta_z = \mathbb{R}$. Suppose that

$$\begin{aligned} D_1 &= \langle x \in [3, 5] \rangle \vee f \vee \neg g, \\ D_2 &= \langle x - y < 4 \rangle \vee \neg h \vee \langle q \in \{b, c\} \rangle, \\ D_3 &= \langle x + 3y \geq 2 \rangle \vee \langle 3x - z = 3 \rangle \vee g, \\ D_4 &= \langle x - z > 1 \rangle, \\ D_5 &= \langle z \in [-2, 5] \rangle \vee \neg f \vee \langle r \in \{a, c, d\} \rangle, \end{aligned}$$

are the constraint clauses where f and $\neg f$, for example, are abbreviations for the constraints $\langle f = 1 \rangle$ and $\langle f = 0 \rangle$, respectively. If x is the variable to be eliminated, then Σ can be decomposed into $\Sigma_X = \{D_1, D_2, D_3, D_4\}$ and $\Sigma_R = \{D_5\}$. Furthermore, Σ_X can be written as $\Sigma_X = \{X_1 \vee R_1, X_2 \vee R_2, X_3 \vee R_3, X_4\}$ with

$$X_1 = \langle x \in [3, 5] \rangle, \quad R_1 = f \vee \neg g,$$

$$\begin{aligned}
X_2 &= \langle x - y < 4 \rangle, & R_2 &= \neg h \vee \langle q \in \{b, c\} \rangle, \\
X_3 &= \langle x + 3y \geq 2 \rangle \vee \langle 3x - z = 3 \rangle, & R_3 &= g, \\
X_4 &= \langle x - z > 1 \rangle.
\end{aligned}$$

To summarize, we have a set $\Sigma = \{X_1 \vee R_1, X_2 \vee R_2, X_3 \vee R_3, X_4, D_5\}$ with $q = 3$, $r = 4$ and $s = 5$.

Theorem 1 *Let Σ be a set of minimal constraint clauses. If Σ can be decomposed into Σ_X and Σ_R as described above, then*

$$Elim_x(\Sigma) = Elim_x(\Sigma_X \cup \Sigma_R) = \mu(Elim_x(\Sigma_X) \cup \Sigma_R) \quad (8)$$

defines the elimination of the variable x .

This theorem shows that irrelevant constraint clauses are not really needed for the elimination process.

3.1 General Constraint Resolution

The remaining problem is to eliminate the variable $x \in V$ from the set of relevant clauses Σ_X . Recall that Σ_X is always a set as shown in (7) with $0 \leq q \leq r \leq s$. If $I = \{1, \dots, q\}$ and $I^* = \{q+1, \dots, r\}$ are the sets of indices of the clauses in Σ_X , $J \subseteq I \cup I^*$, then $X_J = \mu\{X_i : i \in J\}$ denotes the corresponding minimal set of sub-clauses X_i . Furthermore, $R_J = \mu\{R_i : i \in J\}$ denotes the corresponding minimal set of sub-clauses R_i for any $J \subseteq I$.

Theorem 2 *Let Σ_X be a set of relevant clauses for x . If I and I^* are sets of indices as defined above, then*

$$Elim_x(\Sigma_X) = \mu\left(\bigcup_{\emptyset \subseteq J \subseteq I} \{\delta \vee (\vee R_J) : \delta \in Elim_x(X_{J \cup I^*})\}\right) \quad (9)$$

defines the elimination of the variable x .

This theorem describes the main principle of the variable elimination process. The resulting clauses $\delta \vee (\vee R_J)$ are called *resolvents* of Σ_X . The idea is to compute all possible resolvents for every minimal subset of sub-clauses. This procedure will become more clear by looking at the following example.

Example 2 Let Σ_X be a set of relevant clauses for x with $q = 3$ and $r = 4$. Thus, the situation is the same as in Example 1. $I = \{1, 2, 3\}$ and $I^* = \{4\}$ are the corresponding sets of indices. We assume that $X_{J \cup I^*} = \mu X_{J \cup I^*}$ and $R_J = \mu R_J$ for all $J \subseteq I$. The result of the previous theorem can then be written as

$$\begin{aligned}
Elim_x(\Sigma_X) &= Elim_x(\{X_1 \vee R_1, X_2 \vee R_2, X_3 \vee R_3, X_4\}) \\
&= \mu(\{\delta : \delta \in Elim_x(\{X_4\})\}) \\
&\quad \cup \{\delta \vee R_1 : \delta \in Elim_x(\{X_1, X_4\})\} \\
&\quad \cup \{\delta \vee R_2 : \delta \in Elim_x(\{X_2, X_4\})\} \\
&\quad \cup \{\delta \vee R_3 : \delta \in Elim_x(\{X_3, X_4\})\} \\
&\quad \cup \{\delta \vee R_1 \vee R_2 : \delta \in Elim_x(\{X_1, X_2, X_4\})\} \\
&\quad \cup \{\delta \vee R_1 \vee R_3 : \delta \in Elim_x(\{X_1, X_3, X_4\})\} \\
&\quad \cup \{\delta \vee R_2 \vee R_3 : \delta \in Elim_x(\{X_2, X_3, X_4\})\} \\
&\quad \cup \{\delta \vee R_1 \vee R_2 \vee R_3 : \delta \in Elim_x(\{X_1, X_2, X_3, X_4\})\}.
\end{aligned}$$

The above example indicates that Theorem 2 includes the classical resolution principle for propositional logic as a special case in which only pairs of clauses (instead of subsets of clauses) are combined (see Subsection 4.1).

Finally, the remaining problem of the variable elimination process is the computation of $Elim_x(X_{J \cup I^*})$, where $X_{J \cup I^*}$ is a minimal set of clauses in which the variable x is contained in every constraint. A clause $\delta \in Elim_x(X_{J \cup I^*})$ is called a *residue* of $X_{J \cup I^*}$. The problem of computing residues will be discussed in the following subsection. However, two important particular situations are possible:

- If $Elim_x(X_{J \cup I^*}) = \emptyset$, i.e. there are no residues, then

$$\{\delta \vee (\vee R_J) : \delta \in Elim_x(X_{J \cup I^*})\} = \emptyset. \quad (10)$$

- If $Elim_x(X_{J \cup I^*}) = \{\perp\}$, i.e. the only residue is the empty clause \perp , then

$$\{\delta \vee (\vee R_J) : \delta \in Elim_x(X_{J \cup I^*})\} = \{\vee R_J\}. \quad (11)$$

Note that the second case implies $Elim_x(X_{J' \cup I^*}) = \{\perp\}$ for all $J' \supset J$. More generally, if $\delta \in Elim_x(X_{J \cup I^*})$, then $\delta \in Elim_x(X_{J' \cup I^*})$ implies that $\delta \vee (\vee R_J)$ entails $\delta \vee (\vee R_{J'})$ for all $J' \supset J$. This observation tells us that for a set J of indices only residues are to be considered which are not residues for any smaller set $J' \subset J$. The set of such residues can be defined as

$$Elim_x^*(X_{J \cup I^*}) = Elim_x(X_{J \cup I^*}) \setminus \left(\bigcup_{\emptyset \subseteq J' \subset J} Elim_x(X_{J' \cup I^*}) \right). \quad (12)$$

In this way, the elimination procedure of Theorem 2 can be reformulated as follows:

Corollary 1 *Let Σ_X be a set of relevant clauses for x . If I and I^* are sets of indices as defined above, then*

$$Elim_x(\Sigma_X) = \mu \left(\bigcup_{\emptyset \subseteq J \subseteq I} \{ \delta \vee (\vee R_J) : \delta \in Elim_x^*(X_{J \cup I^*}) \} \right) \quad (13)$$

defines the elimination of the variable x .

3.2 Computing Residues

Let $X_J = \{X_1, \dots, X_m\}$ be an arbitrary minimal set of clauses $X_i = C_{i,1} \vee \dots \vee C_{i,m_i}$ with $x \in Vars(C_{i,j})$. The remaining problem is then to eliminate x from X_J . As mentioned in the previous subsection, clauses $\delta \in Elim_x(X_J)$ are called *residues* of X_J . The general idea for computing residues is to transform X_J into an equivalent disjunctive normal form (DNF) and back. Transforming X_J into a DNF can be seen as computing the Cartesian product over the clauses $X_i \in X_J$ by regarding them as sets of constraints. The result

$$DNF(X_J) = \prod_{i=1}^m X_i = \{T_1, \dots, T_M\} \quad (14)$$

is a set of terms (conjunctions) of constraints with $M = \prod_{i=1}^m |X_i|$. Every term $T_i \in DNF(X_J)$ contains m constraints, one from each clause in X_J .

Example 3 *Let $X_J = \{X_1, X_2, X_3\}$ be the set of constraint clauses for $m = 3$ and with $X_1 = C_1 \vee C_2$, $X_2 = C_3 \vee C_4 \vee C_5$, and $X_3 = C_6$. Therefore, $DNF(X_J) = \{T_1, T_2, T_3, T_4, T_5, T_6\}$ contains the following $M = 6$ terms:*

$$\begin{aligned} T_1 &= \{C_1, C_3, C_6\}, & T_2 &= \{C_1, C_4, C_6\}, & T_3 &= \{C_1, C_5, C_6\}, \\ T_4 &= \{C_2, C_3, C_6\}, & T_5 &= \{C_2, C_4, C_6\}, & T_6 &= \{C_2, C_5, C_6\}. \end{aligned}$$

Let $T_i = \{C_{i,1}, \dots, C_{i,m}\}$ be a term obtained from $DNF(X_J)$. Clearly, every constraint $C_{i,j} \in T_i$ can be considered as a constraint clause of length 1. Therefore, $T'_i = Elim_x(T_i)$ denotes the new set of constraints obtained from eliminating x from T_i . Let $T' = \{T'_1, \dots, T'_M\}$ be the new set of such terms. Transforming T' back into a CNF can be seen as computing the Cartesian product over the terms $T'_i \in T'$. The resulting minimal set

$$CNF(T') = \mu \{ \vee D : D \in \prod_{i=1}^M T'_i \} \quad (15)$$

is a new set of constraint clauses in which the variable x does not appear any more.

Theorem 3 If $X_J = \{X_1, \dots, X_m\}$ is set of constraint clauses $X_i = C_{i,1} \vee \dots \vee C_{i,m_i}$ with $x \in \text{Vars}(C_{i,j})$ for $1 \leq i \leq m$ and $1 \leq j \leq m_i$, then

$$\text{Elim}_x(X_J) = \text{CNF}(\{\text{Elim}_x(T_i) : T_i \in \text{DNF}(X_J)\}) \quad (16)$$

defines the elimination of the variable x .

Example 4 Consider the situation from Example 3. Suppose that the elimination of the variable x produces the following new sets of constraints:

$$\begin{aligned} T'_1 &= \text{Elim}_x(T_1) = \{C'_1, C'_2\}, & T'_2 &= \text{Elim}_x(T_2) = \{\perp\}, \\ T'_3 &= \text{Elim}_x(T_3) = \{C'_3\}, & T'_4 &= \text{Elim}_x(T_4) = \{C'_4, C'_5\}, \\ T'_5 &= \text{Elim}_x(T_5) = \{\perp\}, & T'_6 &= \text{Elim}_x(T_6) = \{C'_6\}. \end{aligned}$$

Note that $x \notin \text{Vars}(C'_i)$ for $1 \leq i \leq 6$. The result of the elimination process is then a set of four residues, that is $\text{Elim}_x(X_J) = \{\delta_1, \delta_2, \delta_3, \delta_4\}$ with

$$\begin{aligned} \delta_1 &= C'_1 \vee C'_3 \vee C'_4 \vee C'_6, & \delta_2 &= C'_1 \vee C'_3 \vee C'_5 \vee C'_6, \\ \delta_3 &= C'_2 \vee C'_3 \vee C'_4 \vee C'_6, & \delta_4 &= C'_2 \vee C'_3 \vee C'_5 \vee C'_6. \end{aligned}$$

The remaining problem now is the elimination of the variable x from sets $T_i = \{C_{i,1}, \dots, C_{i,m}\}$ of constraints with $x \in \text{Vars}(C_{i,j})$ for $1 \leq i \leq M$ and $1 \leq j \leq m$. This problem depends strongly on the type of constraints involved. Some particular types of constraints will be discussed in Section 4. In all cases, two important particular situations are possible:

- If there is a term $T_i \in \text{DNF}(X_J)$ such that $\text{Elim}_x(T_i) = \emptyset$, then

$$\text{Elim}_x(X_J) = \emptyset.$$

- If $\text{Elim}_x(T_i) = \{\perp\}$ for all terms $T_i \in \text{DNF}(X_J)$, then

$$\text{Elim}_x(X_J) = \{\perp\}.$$

In propositional logic, for example, these two particular cases are the only possible situations (see Subsection 4.1).

4 Special Cases

The variable elimination method of the previous section is a general procedure that works for sets of arbitrary constraint clauses. The main steps of the procedure are described by Theorem 2 and Theorem 3. The purpose of this section is to investigate the elimination procedure when the constraints are restricted to some special cases.

4.1 Propositional Logic

Suppose that the variable $x \in V$ has only two possible values, that is $|\Theta_x| = 2$. In such a case, only two different regular constraints over x are possible. Let $\Theta_x = \{0, 1\}$, for example, be the frame of x , then $\langle x = 1 \rangle$ and $\langle x = 0 \rangle$ are the only regular constraints (often abbreviated by x and $\neg x$, respectively). Let $\Sigma_X = \{D_1, \dots, D_r\} \subseteq \Sigma$ be the set of relevant constraint clauses for x . Recall that every $D \in \Sigma$ is supposed to be minimal. Therefore, if $D = C_1 \vee \dots \vee C_m$ is a constraint clause of Σ_X , then

$$D = \underbrace{C_1}_{x \in \text{Vars}(C_1)} \vee \underbrace{C_2 \vee \dots \vee C_m}_{x \notin \text{Vars}(C_i)} = \begin{cases} \left. \begin{array}{l} x \vee R \\ \neg x \vee R \end{array} \right\} & \text{if } m > 1, \\ \left. \begin{array}{l} x \\ \neg x \end{array} \right\} & \text{if } m = 1, \end{cases} \quad (17)$$

are the only four possibilities for D (compare with Equation 6). Consequently, there are only four possibilities for the minimal sets $X_{J \cup I^*}$ in Theorem 2:

$$\begin{aligned} X_{J \cup I^*} = \emptyset & \Rightarrow \text{Elim}_x(X_{J \cup I^*}) = \emptyset, \\ X_{J \cup I^*} = \{x\} & \Rightarrow \text{Elim}_x(X_{J \cup I^*}) = \emptyset, \\ X_{J \cup I^*} = \{\neg x\} & \Rightarrow \text{Elim}_x(X_{J \cup I^*}) = \emptyset, \\ X_{J \cup I^*} = \{x, \neg x\} & \Rightarrow \text{Elim}_x(X_{J \cup I^*}) = \{\perp\}. \end{aligned}$$

Therefore, the discussion in Subsection 3.2 about computing residues is irrelevant when x is a binary (propositional) variable. Moreover, we have always one of the two special cases of Equation 10 and Equation 11. The interesting case, where resolvents are generated, is the case where $\text{Elim}_x(X_{J \cup I^*}) = \{\perp\}$. Such a case appears as soon as a pair of clauses is taken such that the variable x once appears as a positive and once as a negative literal. Therefore, in Theorem 2 only pairs of clauses must be considered (instead of subsets of clauses). Let Σ_X^+ and Σ_X^- be the sets of clauses in Σ_X in which the variable x appears as a positive and as a negative literal, respectively. Equation 9 in Theorem 2 can then be reformulated as

$$\begin{aligned} \text{Elim}_x(\Sigma_X) &= \text{Elim}_x(\Sigma_X^+ \cup \Sigma_X^-) \\ &= \mu \{R_1 \vee R_2 : x \vee R_1 \in \Sigma_X^+, \neg x \vee R_2 \in \Sigma_X^-\}. \end{aligned} \quad (18)$$

This is the usual variable elimination method of propositional logic. Obviously, because only pairs of clauses are to be considered, this particular case is much more efficient than the general case, where subsets of clauses are considered.

4.2 Set Constraint Logic

Set constraint logic can be regarded as a generalization of propositional logic [3, 14, 15, 16]. Note that the same concept is also known as *signed logic* [17, 18] or multivalent logic [20, 21]. A set constraint $\langle x \in F \rangle$ restricts the possible values of a variable $x \in V$ to a non-empty subset $F \subseteq \Theta_x$. Logical expressions over set constraints can always be simplified by the following rules:

$$\begin{aligned}\neg\langle x \in F \rangle &\equiv \langle x \in \Theta_x \setminus F \rangle, \\ \langle x \in F_1 \rangle \vee \langle x \in F_2 \rangle &\equiv \langle x \in F_1 \cup F_2 \rangle, \\ \langle x \in F_1 \rangle \wedge \langle x \in F_2 \rangle &\equiv \langle x \in F_1 \cap F_2 \rangle.\end{aligned}$$

Every clause $D \in \Sigma_X$ can therefore be written such that the variable x appears in only one set constraint. Therefore, only two cases are possible (compare with Equation 6):

$$D = \underbrace{C_1}_{x \in \text{Vars}(C_1)} \vee \underbrace{C_2 \vee \dots \vee C_m}_{x \notin \text{Vars}(C_i)} = \begin{cases} \langle x \in F \rangle \vee R, & \text{if } m > 1, \\ \langle x \in F \rangle, & \text{if } m = 1. \end{cases} \quad (19)$$

Every set $X_{J \cup I^*}$ in Theorem 2 is therefore a set $\{C_1, \dots, C_{k_J}\}$ of constraints $C_i = \langle x \in F_i \rangle$ with $\text{Vars}(C_i) = \{x\}$. The problem of computing the residues for $X_{J \cup I^*}$ can then be reduced to the following two cases:

$$\text{Elim}_x(X_{J \cup I^*}) = \begin{cases} \{\perp\}, & \text{if and only if } \bigcap_{i=1}^{k_J} F_i = \emptyset, \\ \emptyset, & \text{otherwise.} \end{cases} \quad (20)$$

The problem therefore is to find minimal sets $X_{J \cup I^*}$ with an empty intersection of the corresponding sets F_i . The resolvents are then the corresponding clauses $\vee R_J$. Thus, if $\Sigma_X = \{D_1, \dots, D_r\}$ is the set of relevant clauses for x with $D_i = \langle x \in F_i \rangle \vee R_i$, then the variable elimination can be described by

$$\text{Elim}_x(\Sigma_X) = \mu \{ \vee R_J : J \subseteq \{1, \dots, r\}, \bigcap_{i \in J} F_i = \emptyset \}. \quad (21)$$

An efficient way of computing all such resolvents is described in [3]. Note that the above formula corresponds to the results found in [17, 18, 20, 21] and is closely related to the method described in [8] about optimal k -consistency algorithms.

5 Conclusion

This paper presents a general procedure for eliminating variables from arbitrary sets of constraint clauses. The procedure generalizes the resolution principle and includes

the classical resolution-based methods for propositional logic and set constraint logic as special cases. In the general case, the paper shows that the elimination problem for a set of clauses can always be reduced to the problem of eliminating the variable from a (conjunctive) set of atomic constraints.

Future work will focus on complexity studies and on the investigation of further special cases. Among others, systems of linear equations and inequalities are certainly of particular importance. Other important special cases are systems of interval constraints with their wide-spread applicability in the domain of temporal and spatial reasoning.

References

- [1] B. Anrig, R. Bissig, R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems: Introduction to assumption-based modeling with ABEL. Technical Report 99-1, Institute of Informatics, University of Fribourg, 1999.
- [2] B. Anrig, R. Haenni, and N. Lehmann. ABEL – a new language for assumption-based evidential reasoning under uncertainty. Technical Report 97-01, University of Fribourg, Institute of Informatics, 1997.
- [3] B. Anrig, N. Lehmann, and R. Haenni. Reasoning with finite set constraints. Technical Report 97-11, Institute of Informatics, University of Fribourg, 1997.
- [4] P. Barth. *Logic-Based 0-1 Constraint Programming*. Kluwer Academic Publishers, Norwell, 1996.
- [5] H.-J. Bürkert. A resolution principle for constrained logics. *Artificial Intelligence*, 66(2):235–271, 1994.
- [6] V. Chandru. Variable elimination in linear constraints. *The Computer Journal*, 36(5):463–472, 1993.
- [7] V. Chandru and J. N. Hooker. *Optimization Methods for Logical Inference*. Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., 1999.
- [8] M. C. Cooper. An optimal K-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1989.
- [9] M. Davis and H. Putnam. A Computation Procedure for Quantification Theory. *Journal of ACM*, 7:201–215, 1960.

- [10] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.
- [11] R. Duffin. On Fourier’s analysis of linear inequality systems. *Mathematical Programming Study*, 1:71–95, 1974.
- [12] R. Haenni. Cost-bounded argumentation. *Int. Journal of Approximate Reasoning*, 26(2):101–127, 2001.
- [13] R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems. In J. Kohlas and S. Moral, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 5: Algorithms for Uncertainty and Defeasible Reasoning*. Kluwer, Dordrecht, 2000.
- [14] R. Haenni and N. Lehmann. Assumption-based reasoning with finite set constraints. In *IPMU’98, Proceedings of the seventh international conference, Paris, France*, pages 1289–1295, 1998.
- [15] R. Haenni and N. Lehmann. Reasoning with finite set constraints. In *ECAI’98, Workshop W17: Many-valued logic for AI application*, pages 1–6, 1998.
- [16] R. Haenni and N. Lehmann. Buidling argumentation systems on set constraint logic. In B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, editors, *Information, Uncertainty and Fusion*, pages 393–406. Kluwer Academic Publishers, 2000.
- [17] R. Hähnle. Short conjunctive normal forms in finitely-valued logics. *Journal of Logic and Computation*, 4(6):905–927, 1994.
- [18] R. Hähnle and G. Escalada-Imaz. Deduction in many-valued logics: a survey. *Mathware & Soft Computing*, IV(2):69–97, 1997.
- [19] J. N. Hooker. Generalized resolution for 0–1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.
- [20] J. N. Hooker. Logic-based methods for optimization. *Lecture Notes in Computer Science*, 874:336–349, 1994.
- [21] J. N. Hooker and M. A. Osorio. Mixed logical linear programming. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 96, 1999.
- [22] J.-L. Imbert. Variable elimination for generalized linear constraints. In David S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 499–516, Budapest, Hungary, 1993. The MIT Press.

- [23] K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56(2-3):301–353, 1992.
- [24] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19 & 20:503–582, May 1994.
- [25] J. Kohlas. Computational theory for information systems. Technical Report 97–07, University of Fribourg, Institute of Informatics, 1997.
- [26] J. Kohlas, R. Haenni, and S. Moral. Propositional information systems. *Journal of Logic and Computation*, 9 (5):651–681, 1999.
- [27] J. Kohlas and R. Stärk. Information algebras and information systems. Technical Report 96–14, University of Fribourg, Institute of Informatics, 1996.
- [28] C. Lassez and J.-L. Lassez. Quantifier elimination for conjunction of linear constraints by a convex hull algorithm. Technical report, IBM T. J. Watson Research Center, 1991.
- [29] J. L. Lassez. Parametric queries, linear constraints and variable elimination. In A. Miola, editor, *Proceedings of the Conference on Design and Implementation of Symbolic Computation Systems*, volume 429 of *Lecture Notes in Computer Science*, pages 164–173. Springer-Verlag, 1990.
- [30] P. P. Shenoy. Valuation-based systems: A framework for managing uncertainty in expert systems. In L. A. Zadeh and J. Kacprzyk, editors, *Fuzzy Logic for the Management of Uncertainty*, pages 83–104. John Wiley and Sons, New York, 1992.
- [31] P. P. Shenoy. Binary join trees. In E. Horvitz and F. V. Jensen, editors, *Uncertainty in Artificial Intelligence*, pages 492–499. Morgan Kaufmann, 1996.
- [32] P. P. Shenoy. Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of Approximate Reasoning*, 17(2–3):239–263, 1997.
- [33] P. P. Shenoy and G. Shafer. Axioms for probability and belief functions propagation. In R.D. Shachter and al., editors, *Uncertainty in Artificial Intelligence 4*. North Holland, 1990.
- [34] M. E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.
- [35] H. P. Williams. Fourier-Motzkin elimination extension to integer programming problems. *Journal of Combinatorial Theory (A)*, 21:118–123, 1976.